SUPPORT TOOLS FOR OBJECT ORIENTED SOFTWARE DEVELOPMENT

BY

GARETH CARTER

A THESIS SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

IN

COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF IRELAND, MAYNOOTH

HEAD OF DEPARTMENT : Prof. RONAN O’REILLY

THESIS SUPERVISOR : Ms. ROSEMARY MONAHAN

June, 2005
Abstract

This thesis provides an overview of the current state-of-the-art in software development tools with a particular interest in those tailored to the Object Oriented programming paradigm. From examining and experimenting with these tools, the work presents one of the most ambitious and advanced, Perfect Developer, critically evaluating it as an industrial-strength tool for the software development community. The work presents two case studies developed with the support of Perfect Developer to give merit to this evaluation. Finally, a set of recommendations that attempt to solve the short-comings found within Perfect Developer, and support tools in general, is provided.
# Table of Contents

Abstract ii

Table of Contents iii

List of Tables xiii

List of Figures xiv

1 Introduction 1

1.1 Support for Representation .......................... 1
1.2 Support for Implementation .......................... 2
1.3 Support for Reasoning .............................. 3
1.4 Aims of the Thesis ................................ 4
1.5 Structure of the Thesis ............................. 5

2 The Software Development Life-Cycle 8

2.1 Requirements Acquisition .......................... 8
2.1.1 Natural Language ................................ 8
2.1.2 Graphical Exploration ........................... 9
2.1.3 Technical Description ........................... 9

2.2 Software Representation ............................ 10
2.2.1 Model Oriented Specification ..................... 10

2.2.2 Structural Representation ......................... 11

2.2.3 Lightweight Specification ......................... 11

2.3 Software Implementation .......................... 12
2.3.1 Object Oriented Programming Languages ........ 12
2.3.2 Object Oriented Programming Paradigm .......... 13
2.3.3 Object Oriented Programming Challenges ........... 14
2.4 Software Testing ........................................ 15
  2.4.1 Test Case Analysis ............................... 15
  2.4.2 Validation ......................................... 16
  2.4.3 Verification ....................................... 17
2.5 Conclusions ............................................ 17

3 Tool Support for Development ............................. 18
  3.1 Model Verification ................................... 18
    3.1.1 U2B ............................................ 19
    3.1.2 SLAM .......................................... 20
    3.1.3 Alloy Analyzer .................................. 20
  3.2 Specification Animation ............................... 21
    3.2.1 LARCH Prover ................................ 22
    3.2.2 Z/EVES ........................................ 22
    3.2.3 Cafe ............................................ 23
  3.3 Extended Static Checking .............................. 24
    3.3.1 ESC/Java2 ...................................... 24
    3.3.2 The KeY Tool ................................... 25
    3.3.3 Omnibus ........................................ 26
  3.4 Refinement ............................................ 26
    3.4.1 B ............................................... 27
    3.4.2 ProofPower ..................................... 27
    3.4.3 Perfect Developer ............................... 28
3.5 Conclusion ............................................... 28

4 The Perfect Language ...................................... 31
  4.1 Typing ................................................ 31
E.14 StaffBase.pd ........................................... 124
E.15 Subject.pd ........................................... 124
E.16 UserBase.pd ........................................... 125
E.17 nat.pd ................................................. 125
E.18 string.pd ................................................ 125

F  Library Specification  .................................... 126
F.1  Author.pd ............................................. 126
F.2  BookDescription.pd .................................. 126
F.3  BorrowerBase.pd ..................................... 126
F.4  Borrower.pd ........................................... 127
F.5  Borrowing.pd .......................................... 127
F.6  LibraryBookDescription.pd ......................... 127
F.7  LibraryCatalog.pd .................................... 128
F.8  LibraryDB.pd ......................................... 128
F.9  LibraryItem.pd ........................................ 131
F.10 LibraryResultCode.pd ................................. 132
F.11 LibraryStock.pd ...................................... 132
F.12 Person.pd ............................................ 133
F.13 Staff.pd .............................................. 133
F.14 StaffBase.pd .......................................... 134
F.15 Subject.pd ............................................ 134
F.16 UserBase.pd .......................................... 134

G  Library Refinement ..................................... 136
G.1  HashedBucket.pd .................................... 136
G.2  PriorityQueue.pd .................................... 137
G.3  LibraryCatalog.pd ................................... 140
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G.4</td>
<td>LibraryItem.pd</td>
</tr>
<tr>
<td>G.5</td>
<td>LibraryStock.pd</td>
</tr>
<tr>
<td>H</td>
<td>Library Java Implementation</td>
</tr>
<tr>
<td>H.1</td>
<td>Author.java</td>
</tr>
<tr>
<td>H.2</td>
<td>BookDescription.java</td>
</tr>
<tr>
<td>H.3</td>
<td>BorrowerBase.java</td>
</tr>
<tr>
<td>H.4</td>
<td>Borrower.java</td>
</tr>
<tr>
<td>H.5</td>
<td>Borrowing.java</td>
</tr>
<tr>
<td>H.6</td>
<td>HashedBucket.java</td>
</tr>
<tr>
<td>H.7</td>
<td>LibraryAccess.java</td>
</tr>
<tr>
<td>H.8</td>
<td>LibraryBookDescription.java</td>
</tr>
<tr>
<td>H.9</td>
<td>LibraryCatalog.java</td>
</tr>
<tr>
<td>H.10</td>
<td>LibraryDB.java</td>
</tr>
<tr>
<td>H.11</td>
<td>LibraryItem.java</td>
</tr>
<tr>
<td>H.12</td>
<td>LibraryResultCode.java</td>
</tr>
<tr>
<td>H.13</td>
<td>LibraryStock.java</td>
</tr>
<tr>
<td>H.14</td>
<td>Person.java</td>
</tr>
<tr>
<td>H.15</td>
<td>PriorityQueue.java</td>
</tr>
<tr>
<td>H.16</td>
<td>Staff.java</td>
</tr>
<tr>
<td>H.17</td>
<td>StaffBase.java</td>
</tr>
<tr>
<td>H.18</td>
<td>UserBase.java</td>
</tr>
<tr>
<td>I</td>
<td>Resource Manager Specification Code</td>
</tr>
<tr>
<td>I.1</td>
<td>ProcessErrorCode.pd</td>
</tr>
<tr>
<td>I.2</td>
<td>ProcessItem.pd</td>
</tr>
<tr>
<td>I.3</td>
<td>ProcessStateCode.pd</td>
</tr>
<tr>
<td>I.4</td>
<td>ResourceItem.pd</td>
</tr>
<tr>
<td>Section</td>
<td>File Name</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>I.5</td>
<td>ResourceManager.pd</td>
</tr>
<tr>
<td>I.6</td>
<td>Semaphore.pd</td>
</tr>
<tr>
<td>I.7</td>
<td>System.pd</td>
</tr>
<tr>
<td>J</td>
<td>Higher Order Functions with Perfect Developer</td>
</tr>
<tr>
<td>J.1</td>
<td>HOF.pd</td>
</tr>
<tr>
<td>J.2</td>
<td>Double.pd</td>
</tr>
<tr>
<td>J.3</td>
<td>MyInteger.pd</td>
</tr>
<tr>
<td>J.4</td>
<td>QuadInt.pd</td>
</tr>
<tr>
<td>K</td>
<td>Vegetarian Problem in Perfect Developer</td>
</tr>
<tr>
<td>K.1</td>
<td>Food.pd</td>
</tr>
<tr>
<td>K.2</td>
<td>Vegetable.pd</td>
</tr>
<tr>
<td>K.3</td>
<td>Person.pd</td>
</tr>
<tr>
<td>K.4</td>
<td>Vegetarian.pd</td>
</tr>
<tr>
<td>K.5</td>
<td>Main.pd</td>
</tr>
<tr>
<td>L</td>
<td>Vegetarian Solution in Perfect Developer</td>
</tr>
<tr>
<td>L.1</td>
<td>Person.pd</td>
</tr>
<tr>
<td>L.2</td>
<td>Vegetarian.pd</td>
</tr>
</tbody>
</table>
## List of Tables

1. Summary of Support Mechanisms ..................................... 29  
2. Summary of Support Quality ......................................... 30  
3. Controlling Perfect Developer ....................................... 53
# List of Figures

1. Harbour Example - Three Ships docked, Three Ships Queueing  
   - Page 32
2. Perfect Developer GUI  
   - Page 52
3. Harbour UML diagram  
   - Page 54
4. UML Model of the Library Database  
   - Page 65
5. UML Model of the Resource Manager  
   - Page 74
1 Introduction

Software Development projects often result in the late delivery of unreliable software systems that cost much more than originally expected. Such projects reflect negatively on the software development community. Ongoing research in this community aims to correct these problems through improved software development techniques. In particular, this thesis focuses on software development support tools which aim to improve productivity time and software reliability.

Tool support may be delivered to the developer in a number of ways; Support for Representation; Implementation; and Reasoning are of particular interest in this thesis. Representation concerns an abstraction of the software under development, free from its implementation details but retaining the essence of its meaning. Implementation concerns the programming language used to implement the software and its efficiency. Reasoning refers to the mechanisms used to ensure that the resulting software is correct with respect to the abstraction thereof. In this chapter, we outline these three key forms of support and present the aims and structure of the thesis.

1.1 Support for Representation

Representation assists developers in fully understanding the essence of software under construction by providing a clear, abstract description of the required system. This representation is discovered through the software development phase Requirements Acquisition. The Requirements are often stated in natural language, leaving them susceptible to incompleteness, incorrectness and multiple interpretations. Structural Representations may be employed to better document the software abstractly, though these representations are all too often closely related to an implementation language. We believe that an intermediary representation, through Mathematic Specifications, should exist
between requirements and implementation.

Unfortunately, specification languages have had little success in industry, possibly due to their highly mathematical nature. The cost of employing specification languages may be considered too high to merit the development of a specification of software in advance of its implementation. The value of specification development can be increased by promoting a link between specification and implementation through Refinement[1] which will be discussed in detail in Chapter 3.

The specification language must be rich and expressive without being unnecessarily complex. Developers should be able to specify the requirements of software with ease. The language should be easy to learn with its theoretical underpinnings documented. The elements of the language should inhabit a single theory of software development and promote this theory throughout the development life-cycle. It is with respect to these qualities that we will evaluate specification languages presented in this thesis.

1.2 Support for Implementation

The language chosen to implement software is of utmost importance, affecting the efficiency and maintainability of software. This thesis confines itself to the Object Oriented programming paradigm[2] as the approach for software development. The paradigm is well established in industry and academia. Research has shown that software development through Object Oriented programming languages gives rise to an improvement in software quality in the long term[3].

The Object Oriented paradigm was developed to offer a gentle learning curve to developers by building upon its intuitive use of runtime “objects”[4]. It promotes modular software development, allowing developers to work in parallel
or to alter components independently. It supports re-use of software saving
development time[5]. We believe these reasons justify the Object Oriented pro-
gramming paradigm to provide support for implementation.

While the Object Oriented paradigm supports many aspects of software
development, its support of the correctness of software remains an open issue.
The paradigm evolved to solve problems faced by software engineers, rather
than from a single mathematical foundation[6]. Several theories of what it
means for a language to be “Object Oriented” have resulted from the varied and
plentiful research[7]. One example of this arises from the complications in the
Object Oriented message passing mechanisms which has resulted in numerous
treatments of Inheritance and Polymorphism. This thesis will keep the issue of
software correctness to the forefront of its discussions of support for software
implementation.

1.3 Support for Reasoning

In recent years, technological advances in automated reasoning have allowed
tools that support the reasoning of software to be developed. Theoreticians
are harnessing the power of computers to perform complex mathematics that
were never before possible, allowing Verification, the grand challenge of software
development[8], to be a possibility. While full verification of software is still a
distant target, modern tools claim to assist in the discovery of common software
development errors hence helping developers to construct correct and reliable
software.

Research have employed technologies like Symbolic Executors[9, 10, 11],
Model Checkers[12] and Theorem Provers[13, 14, 15, 11]. These tools have
met with some skepticism in industry as they are viewed as too complex to
wield on everyday software problems. They are rarely used apart from during
the development of safety critical systems[16]. We believe that to stimulate in-
dustrial interest in support tools, an integrated tool that provides a verification
of a software representation, its implementation and the relation between them
is needed.

Support for Reasoning must be accessible to developers so that they may
understand and develop this verification. Reasoning tools must be easy to use
while ensuring the correctness of the software to improve development. Many
tools, for example the B-Tool[17], have failed to reach a large audience owing to
the complexity of their operation[18]. The extent to which reasoning support is
automated is crucial to the adoption of the tool.

This thesis will explore the possible support for representation, implementa-
tion and the associated reasoning within a framework that is user friendly and
applicable by the software development community.

1.4 Aims of the Thesis

The ultimate aim of the thesis is to generate a set of recommendations for
a tool that supports software development. The recommendations must be
practical and solve existing problems that are not sufficiently dealt with by the
current selection of tools available. They should provide strong support for the
representation, implementation and reasoning of software by merging the best
of practical and academic tools.

The thesis will present, explore and critically evaluate the Perfect
Developer[19] software development tool to discover the limitations of exist-
ing software development tools. The exploration will involve the construction
of medium scale software that illustrate exemplars of software development.
The evaluation will discuss areas of improvement that should be met by the
recommendations.
The thesis will also survey other tools that support correct software development. This survey stands to catalog the current cutting edge technologies, reviewing each alongside its peers. It will introduce the primary concepts of tool support, highlighting their usage in software development.

1.5 Structure of the Thesis

The thesis will present the current software development life-cycle and discuss its associated pitfalls. Software Development tools that support the representation, implementation and reasoning of software are described with respect to these pitfalls. In particular, the Perfect Developer support tool is presented, documenting the language and the process it supports. A critical analysis of this tool is provided, drawing upon case studies and our experiences with it. This analysis is used to generate recommendations for an improved software development support tool.

In this chapter, the domain and general direction of the thesis is outlined. It serves as an introduction to the material covered herein and sets the tone for the work that will follow. The chapter establishes the goals of our work and the means by which we intend to reach these.

Chapter 2 presents the software development life-cycle and the ways that support can be delivered to developers. The chapter pre-supposes that software is developed in an Object Oriented programming language and presents many pitfalls of the associated programming paradigm. In particular, the chapter describes the need for better representation and implementation support, prompting the ways that reasoning tools could be delivered to assist software development.

Chapter 3 surveys the current technologies that support the reasoning of software to ensure its correctness. These tools are categorised with respect to the
kind of reasoning support they provide. The advantages and disadvantages of each tool are discussed to illustrate their relevance to the software development process advocated in the previous chapter. We conclude by selecting the Perfect Developer software tool as the most ambitious in providing support for the software development life-cycle.

Chapter 4 presents a focused exploration of Perfect, the language of representation and implementation supported by the Perfect Developer support tool. The programming concepts of Perfect are documented while illustrating its syntax. A small supporting example is employed to demonstrate the representation and implementation support provided by Perfect. At the close of the chapter, the reader should be well versed in Perfect and prepared to use the Perfect Developer tool.

Chapter 5 presents Perfect Developer, providing the groundwork for an understanding of role played in the software development life-cycle. The tool components and their support of the software development life-cycle are discussed. In particular, we focus on the support for reasoning with specifications and their refinement within the tool through extension of the example used in Chapter 4.

In Chapter 6, we present two real world case studies which we undertook to explore the use of Perfect Developer. These case studies are both exemplars of software, requiring unique approaches to software development. The case studies serve to illustrate the strengths and weaknesses of the tool with respect to medium scale software development.

Chapter 7 presents our analysis of Perfect Developer with respect to its support for the representation, implementation and reasoning of software, drawing upon the case studies and our experiences with the tool.
The set of recommendations for an Object Oriented support tool are presented in Chapter 8. These recommendations address the issues raised in Chapter 7 employing novel techniques from research to support the representation, implementation and reasoning of software.

In Chapter 9, the thesis concludes by reviewing and evaluating the work with respect to the aims stated in the first Chapter. In particular the practicality of the recommendations is described to prompt future work in their development and deployment as a tool.
2 The Software Development Life-Cycle

In this chapter, we discuss four of the phases in the software development life-cycle that we feel can be improved by software development support tools. These phases are discussed with respect to the difficulties that arise from working within the Object Oriented programming paradigm. Within each phase, we discuss how support for representation and implementation may best be provided. The chapter makes reference to potential avenues of reasoning support that will be examined in greater depth in Chapter 3. Our understanding of “Object Oriented Software Development” is documented, which will justify the need of tool support for this kind of software development.

2.1 Requirements Acquisition

The initial stage of Requirements Acquisition offers many challenges, yet is of the utmost importance. The client who requests the software must define the requirements that should be met. The developer in charge of constructing the software must have a clear understanding of these requirements. A shared language between client and developer must be found to document these requirements. Natural Language is commonly chosen, but this can be lead to incomplete and inconsistent requirements. A Graphical Exploration of requirements may be subsequently performed, but this is usually undertaken by the developer independent of the client. A Technical Description of the software requirements may be described to ensure consistency, completeness and correctness of the software requirements.

2.1.1 Natural Language

Natural Language provides a simple way of stating the clients requirements as it is shared by both client and developer. These kinds of requirements are fraught with many challenges: The requirements may be incomplete owing to
assumptions made on the part of the client; Inconsistencies may arise from the lack of a rigorous definition of natural language; It is possible that a developer may mis-interpret these requirements, resulting in the development of an incorrect software solution. Natural language is clearly not entirely suitable for requirements acquisition, but is the most common starting point for any software development.

2.1.2 Graphical Exploration

Employing graphics to support the representation of requirements is a common development +technique. The Unified Modelling Language(UML)[20] uses Use Case Diagrams[21] to better represent requirements. This representation may be included in later development stages, ensuring that all requirements are addressed by the software. However, they are commonly misunderstood and under-utilized by developers because they force developers to make design decisions earlier than would be ideal. Also, the UML is a software development language not understood in general outside this community. These diagrams would not be presentable to a client to ensure completeness and correctness.

2.1.3 Technical Description

Technical descriptions give developers a more precise encoding of requirements without implementation details. The languages that provide these technical descriptions utilize mathematical notations to capture the essence of requirements such as with functional or algebraic programming languages. Functional Programming languages, such as Objective-ML[22] or Haskell++[23], emphasize the evaluation of expressions, rather than execution of commands. Algebraic Programming languages, such as CafeOBJ[24], promote the specification of software behaviour, rather than the means by which this behaviour is achieved. By describing the requirements as an Algebraic Property, a technical description can
be rigorously checked for correctness and consistency. Both techniques have met with much use in academic circles, but failed to reach industry as a result of the high mathematical expertise required to wield them.

2.2 Software Representation

The representation of software may occur in many ways throughout the development life-cycle. The essence of the software meaning may be represented through some Model Oriented Specification. The Structure of the final software may be represented to promote “divide-and-conquer” solutions. Lightweight Specifications that detail the meaning of individual components may be used to represent lower level parts of the software. Each kind of representation has found its own support in modern software engineering, but most are under-utilized in the software development life-cycle. Worse still is the lack of unification between the techniques.

2.2.1 Model Oriented Specification

Model Oriented Specifications provide a way to abstractly specify the meaning of a piece of software with little or no concern for the efficiency of the software. This is achieved by defining a mathematical model of the system and the operations that act upon that model. The model of the system is treated as functions or relations between the data types of the model. The operations override these functions, changing its state.

Languages such as Object-Z\[25\] and VDM++[26] support this style of specification and are of particular interest here as they also support the Object Oriented programming paradigm. Both languages provide rich mechanisms for software specification and are tied to many tools that provide reasoning support\[27, 28\]. However, neither language is associated with an implementation language and as a result have not yet penetrated industry. The languages
are seen to require a great deal of effort with little short term benefit.

2.2.2 Structural Representation

A structural representation of software describes the individual modules of software, documenting each components position within the software and its interactions with other modules. Structural representation promotes software decomposition and is often a necessity of large software projects. The Unified Modelling Language (UML) [29] was developed to represent the structure of Object Oriented software and has been widely accepted in industry and academia despite its known flaws [30]. A similar language is the Business Object Notation (BON) [31] which promotes development in the Eiffel [32] programming language.

2.2.3 Lightweight Specification

Lightweight Specifications are specifications local to some feature that describe its behaviour. They are most commonly supported by Design by Contract (DbC) [33] specifications consisting of pre-conditions, post-conditions and invariants. A pre-condition describes the assumptions made by a method of a class. A post-condition describes what properties are guaranteed by the execution of a method of a class. An invariant describes that which is true of a class at all observable points of time. DbC specifications are increasingly popular as they are supported by the UML through the Object Constraint Language (OCL) [20].

One of the richest DbC languages currently available is the Java Modelling Language (JML) [34]. The language was developed to support software development in the Java programming language [35]. It consists of a rich assertion language in which to compose the specifications. It includes mechanisms for Aliasing [36], Frame Conditions [34], and Exception Handling [37], all of which are common causes for errors in object oriented software and are discussed in
the next section.

2.3 Software Implementation

There is always the possibility that a software implementation may not meet its specification. This can often occur if the software is developed in an *Object Oriented programming language* as several components of the paradigm are not sound. This section outlines our understanding of the *Object Oriented programming paradigm* discussing several of the challenges that must be considered to ensure the implementation is sound.

2.3.1 Object Oriented Programming Languages

Java[35], C++[38] and Eiffel[32] are among the most common Object Oriented programming languages in use today. Each supports the essentials of the Object Oriented programming paradigm, but it has been shown that many of their features differ considerably[7]. Java provides a platform-independent programming language with a rich graphical interface. C++ is a development of the C programming language that supported Object Oriented development. Eiffel offers a pure Object Oriented programming language with rich features such as multiple inheritance, generic classes and DbC runtime assertions. Each language offers developers a different definition of what it means to be an Object Oriented programming paradigm.

We feel an Object Oriented programming language should provide developers with the level of efficiency they have come to expect from imperative programming languages like C, yet be more portable as with Java. Such a suite of languages is being developed under the Microsoft .Net platform[39], e.g. the C# programming language[40]. This language supports the combination of external components, potentially written in other languages, to construct software.
2.3.2 Object Oriented Programming Paradigm

Each of Object Oriented programming language supports different definitions of what it means to be “Object Oriented”. To continue our discussion, our understanding of the Object Oriented programming paradigm must be presented. The most important definitions are presented below.

**Objects** - Objects are runtime entities that contain some data and message passing mechanisms. Each object is in charge of its data and can only request data from, or transmit data to, other objects through message passing.

**Class** - A class defines the behaviour of a set of objects with related functionality. Class is viewed as synonymous with type in most Object Oriented programming languages, with Eiffel[32] viewing the two terms as equivalent.

**Features** - The behaviour of objects is defined by the features of their class. Features may be either *attributes* or *methods*.

**Attributes** - The attributes of a class represent the data an object may hold.

**Methods** - The methods represent the message passing mechanisms of the objects. Methods may evaluate or change the objects state during their execution.

**Inheritance** - Inheritance[41] is a mechanism that allows re-use of classes by adapting or enriching a class with new behaviour. It should be provided by all Object Oriented languages in some form. The class that is formed from inheritance is called the *child* class, while the class that is inherited from is called the *parent* class. Whenever a class inherits from another class, the child class implicitly contains all the features of the parent class. Methods may be *overridden* by re-implementing the method in a child class different to its parent class. Some features may even be removed from the class via Descendant Hiding[2].
Polymorphism - Polymorphism allows objects to masquerade as a different type than their instantiated type[42]. The polymorphism that allows an object of a class child to have the type of its parents class is commonly found in Object Oriented programming languages.

2.3.3 Object Oriented Programming Challenges

Overriding Methods - As has been stated, inheritance permits Overriding of methods in the child class. This allows descendant classes to be improved or adapted to changes in the software. Consider a method that finds the square root of a number to some level of accuracy. This method may be overridden to a more efficient implementation, but providing less accuracy in the result. An object calling this method would not know the level of accuracy provided by the result at runtime because polymorphism may hide the type of the object implementing the method. This introduces a lack of safety in software that allows free overriding of methods.

Dynamic Binding - A method is dynamically bound at runtime when the type of an object is known.[43] This type cannot be known statically as it may be masquerading as another type. Dynamic Binding causes the need for strict checking of methods to ensure a method called does what it is expected to. In Beugnards paper[44] on dynamic binding signatures, it is shown how unexpected results occur in most Object Oriented programming languages.

Binary Methods - A Binary Method is a method of a class that accepts a parameter of that same class[45]. Such a method, when inherited, ceases to be a binary method (as the child class has the method but parameter accepts object of the parent class). Consider a 2DPoint object and a 3DPoint object that are in an inheritance relationship of parent and child respectively. An equality method may be defined for each, but because of the problem of binary methods,
the two objects may be compared for equality despite this making no logical sense as they exist on different planes.

**Value and Reference Semantics** - Most Object Oriented programming languages employ *Reference Semantics*. This means that program variables do not represent objects, but rather references to runtime objects. With these semantics, there exists the possibility that two distinct variables may reference the same object, an occurrence known as *Aliasing*. Aliasing makes reasoning about software costly because the reasoning cannot be performed independently on modules. Value Semantics avoids this problem by not allowing references, requiring each variable to represent a unique object.

### 2.4 Software Testing

An essential step in the software development life-cycle is testing the correctness of the software. This can be established by executing a set of test cases and analysing the results in comparison with expected results. However, the execution of test case analysis can only find existing bugs, never prove their absence[46]. Testing the software is also required to *validate* that the software meets the requirements placed upon it. This is usually achieved through prototyping and must be carried out early to prevent propagation of errors in the development life-cycle. We suggest that the *Verification* of both the specification and the implementation of the software would remove the need for both test case analysis and validation, provided the specification had been validated earlier.

#### 2.4.1 Test Case Analysis

When an executable piece of software is available, developers must check that the software works correctly and performs as required. This is performed by *testing* the software, that is providing it with sample data and checking its
result against an expected result. Software requires extensive testing before developers can have confidence in it and as the scale of software grows, the scale of testing grows exponentially. The current culture of releasing “updates” that “plug” holes in software arose from the lack of an effective and rigorous software testing process.

The software life-cycle is typically iterated through many times ensuring that the software is developed to match the clients needs. Time and money constraints often cut this process short before an adequate solution to the clients needs is found. As has been highlighted, many of the problems are well known and if caught early enough could improve the quality of the software. It is the contention of this thesis that tools from the formal methods community could provide a solution to many of these problems and that a tool could be generated that supports the software development life cycle in a better way.

2.4.2 Validation

Validation is the process of ensuring software specifications meets its requirements. This can be achieved by Animating[47] the specifications or by Prototyping software. The advantage of animating specifications is that the software may be validated before an implementation is produced. Time can be saved by ensuring that the specification is valid in advance of developing the implementation. Animation has the side product of ensuring the specification is consistent, preventing other kinds of errors from appearing in the implementation.

Prototyping provides the client with an opportunity to experiment with the software. This is an essential step as the software requirements are likely to change once a client has seen a version of the software. It would be advantageous if a prototype is produced early in the development life-cycle as changes
are usually requested. The eXtreme Programming technique, aims to develop
prototypes fast to solve this problem but tends to construct software lacking in
a structure suitable to maintenance[48]. If specifications could be automatically
compiled into executable code, early prototyping could be achieved at an earlier
stage without sacrificing correctness.

2.4.3 Verification

Verification proves that software is free from bugs and correct with respect
to its specifications. It has been disadvantaged by the lack of clarity found in the
semantics of most common implementation languages. The ability to relate an
implementation to a specification has made techniques such as *Extended Static
Checking*, which lie somewhere between type checking and verification, more
reliable and popular. Full verification is rarely possible due to the scale of most
software projects. However, advances in automated reasoning and the increase
in power could enable tools to be constructed that go a long way to reaching
full verification. An exploration of these tools will be taken in the next chapter.

2.5 Conclusions

This chapter has presented our understanding of four phases of the cur-
rent software development life-cycle, introducing the terminology that will be
employed later in the thesis. The challenges faced during the software devel-
opment life-cycle were described, with an emphasis on those that resulted from
the Object Oriented programming paradigm. These challenges will be kept at
the forefront of our investigations into support tools and should be addressed
by the close of the thesis.
3 Tool Support for Development

This chapter describes a selection of tools that provide support for reasoning about software. The tools are categorised and described with respect to the reasoning technique that they employ. The support provided by each tool for representation, implementation and reasoning is discussed along with the advantages and disadvantages in each. The chapter is dominated by tools that assist development in the Object Oriented programming paradigm, although interesting tools from other paradigms are presented to provide a more complete overview. The techniques of Model Verification, Specification Animation, Extended Static Checking and Refinement are all explored with intent to discover a tool that best supports the software development life-cycle with respect to support for representation, implementation and reasoning.

3.1 Model Verification

Model Verification is a completely automated reasoning technology that can prove software never enters dangerous or unreliable states. Models define the states that software may enter and the transitions between these states. A model of the software and some safety property is presented to a model verification tool to ensure the property is valid. Model verification may take the form of Model Checking[12] or Model Finding[49]. Model checkers construct a tree of possible paths of execution and test the safety property on each, ensuring all possible paths are explored. Model finders attempt to refute the safety property by constructing models that may break it and decide if they are valid within the software model. Each form of reasoning can be performed without any assistance from the developer.

The one drawback to model checking technologies is the State Space Explosion Problem[50]. This problem refers to the exponential increase in potential
paths of a model as the number of states increases. Model checking serves a useful function in reasoning support but a limited one. The tools U2B, SLAM and the Alloy Analyzer are presented in this section because they support some form of model verification.

3.1.1 U2B

U2B[51] integrates the UML(Section 2.1.2 and 2.2.2) and B(Section 3.4.1) to construct a tool that compensates for the weaknesses of both. The tool assists developers in constructing B models of software under development by using graphical UML diagrams to initiate the construction. Initial specifications written in UML are augmented with Abstract Machine Notation(AMN, the specification language of B) specifications. The tool translates these specifications to pure B and some B-Tool[52, 53, 54] can verify their correctness. The tool simplifies the development of AMN specifications, helping to solve many of the complaints from industry surrounding its high mathematical expertise requirements[55].

The tool is still in the early stages of development at the University of Southampton with many interesting problems still to be tackled. Object Oriented features like Inheritance and Polymorphism are still to be included in the tool. However, the tool meets many of the needs of industry[55], e.g. providing formal support through an interface that doesn’t require extensive mathematical expertise. It’s main goal was to simplify the exploration and development of B specifications assisting representation greatly. However, its support of the development process stops at this point providing little or no added support for implementation or reasoning then provided by the UML or the B-Method.
3.1.2 SLAM

The SLAM[56] tool verifies the implementation of C device drivers for the Windows Operating System. The tool requires no additional specification of implementations but can automatically prove that a device driver will never make invalid calls to the Windows kernel that result in system failures. The tool relies on complete and correct specification of the Windows kernel to be available to it but provides developers of device drivers a black box that can provide a 100% guarantee on software correctness. The tool has met with much interest at Microsoft, but will always struggle on its assumption that the operating system is completely and correctly specified[57].

While the tool represents a very limited domain, it does illustrate the possibility of constructing tools that greatly assist correct software development without necessarily increasing the development cost. The tool is completely practical and should be employed in all driver development projects. The tool provides no form of representation or implementation support, but does remove the need for specification of components. The tool truly excels in its support of reasoning but operates in too limited a domain to be of further interest in this thesis.

3.1.3 Alloy Analyzer

The Alloy Analyzer[49] is a model finder rather than a model checker. The Analyzer attempts to find a valid model that refutes the property. Models are developed in the Alloy specification language. Alloy is comparable to Z and OCL, though less expressive than the former and better defined than the latter. It is based on first order logic and can model quantifiers, higher arity relations, polymorphism, subtyping, and signatures[49].
The Alloy Analyzer supports partial models to be analyzed making it particularly useful during development. It is fully automated and can therefore be used easily by developers equipped with a knowledge of Alloy. Reasoning about complex reactive software is supported by the Alloy Analyzer best of all. The Alloy language is very simple to learn and has been taught around the world in at least 15 Universities[58]. For its support of representation and reasoning, the Alloy Analyzer can be considered one of the strongest in the field of model verification. However, the tool provides no support for implementation of software with no link between the Alloy specification and any implementation.

3.2 Specification Animation

Specification animation attempts to verify that specifications are correct, complete and consistent. This is achieved by employing Theorem Proving and Symbolic Execution technologies. Theorem Proving is used to ensure the specifications are consistent, while Symbolic Execution is employed to test whether they are complete and correct with respect to the requirements. In recent years, theorem proving technologies have become increasingly available due to the power and scale of hardware. This makes them a viable technique for developers wishing to include them within the software development life-cycle.

The animation of specification plays an extremely important role in the development of correct software, providing an early prototype of the software to be demonstrated before an implementation is developed. Unfortunately, the animation requires great mathematical expertise and comes at a high cost without producing an early implementation. Tools of this nature are often discounted as unnecessary with the incorrect assumption that testing will solve any problems after an implementation is developed. Here we present that LARCH Prover, Z/EVES and the Cafe tools to discuss this technique
3.2.1 LARCH Prover

The LARCH Prover\cite{59} is a manual theorem prover that supports animation by assisting developers to explore specifications written in LARCH\cite{59}. The focus of the Prover is in finding inconsistencies in specifications by discovering refutations. It provides no automation or heuristic proof strategies, based on the principle that the human factor is integral to the exploration of specifications. The tool is suitable for the early stages of specification development where bug finding plays a priority role.

While the LARCH Prover has been used in research, it is of little interest outside this arena. The tool requires much expertise to be used in even the simplest of software projects. It provides no means of automating the support and each proof attempt must begin from scratch using the built-in reduction commands at each step. The tool will assist in performing the mathematics and ensuring their correctness, but no other support in reasoning is provided. LARCH is adequate as a specification language designed for the specification of C programs, but adapted to handle C++. Once again, there is no support for implementing the software.

3.2.2 Z/EVES

Z/EVES\cite{27} assists the construction, type checking and animation of specifications written in the Z notation\cite{60}. It employs the NEVER deduction agent\cite{61} to prove the consistency of the specification. NEVER is capable of advanced automated reasoning techniques, but provides the developer with the ability to manually prove specifications. It is closer to a proof checker than a theorem prover, allowing developers more control to explore the failure of proofs. The tool has had mass popularity in academia and in those industrial projects with safety concerns as a priority.
The tool provides excellent support to the representation and reasoning of software. The suite of tools that assist the development of Z specification is extensive. The deduction agent NEVER is very advanced, being previously employed in the EVES tool\[62\]. However, the tool provides no implementation support. The tool represents a great reasoning tool, but without support for refinement of specifications to implementations, it cannot support the entire software development process.

### 3.2.3 Cafe

Cafe\[63\] is an environment for the development and verification of CafeOBJ\[24\] programs. The CafeOBJ language is a member of the OBJ languages supporting algebraic specification of software systems with some Object Oriented features. Software is constructed within modules containing the sigma axioms that define the behaviour and constraints of the module. Verification of the modules is provided through a rewriting logic and the theorem proving engine TPS.

CafeOBJ can be used as both a representation and implementation language, allowing software to be developed entirely within the Cafe environment. However, it suffers as both representation and implementation language. Algebraic Specification can be too abstract and only truly beneficial for certain types of software such as Abstract Data Types. The implementation of CafeOBJ specifications can be inefficient and would more commonly be used to prototype software under development. The reasoning engine is good, but often provides misleading information when parsing fails.
3.3 Extended Static Checking

Design by Contract has found much popularity for specification of Object Oriented software as it is extremely modular and supports re-use to a high extent. Therefore, it seems only natural that tools should be built to support design by contract specifications and much research has gone into this area. By combining symbolic execution techniques and theorem provers that perform unification, tools that can perform Verified Design by Contract can be constructed. Such tools can verify that methods are only called when their pre-conditions are met, that methods always meet their post-conditions and that classes maintain their invariants. We now present ESC/Java2, The KeY Tool, and Omnibus, as the most well established tools that perform this kind of verification.

3.3.1 ESC/Java2

ESC/Java2[64] is an extended static checker for Java programs. The tool utilizes the Java Modelling Language(JML)[65] to document design by contract specifications of programs. The tool can provide a greater guarantee of safety than type checking alone, but does not provide total verification of software correctness. It is neither sound nor complete[65] but can catch a number of common programming errors and can be considered most useful as a debugger. The tool is being developed at The Nijmegen Institute for Computing and Information Sciences. It follows on from work done at Compaq on ESC/Java[66]. The theorem prover is based on Greg Nelsons Simplify theorem prover[67] but is currently being updated. Currently it is fully automated requiring users to re-specify properties to aid theorem proving.

ESC/Java2 provides its support for representation in the form of JML lightweight contracts. The support for reasoning is good for ensuring these specifications are consistent but cannot handle complex specifications. The tool
does not directly assist implementation, but does help to make development easier by its support of representation and reasoning. However, the fact that the tool is neither sound nor complete removes the strength of any guarantees it delivers. Overall, it is a good tool, but one that cannot be relied on too heavily.

3.3.2 The KeY Tool

The KeY tool[68] consists of a theorem prover for verification of JavaCard applications. JavaCard is a subset of Java which for use in Smartcards (multithreading is not included). The theorem prover is based on the dynamic logic introduced by Harel[69]. The logic extends Hoare logic permitting first order formulas to contain programs. The tool plugs into Borland Together ControlCenter, a UML modelling package. Software is structural developed through UML class diagrams, specifications are added using OCL and finally the JavaCard implementation is included. A theorem prover attempts to automatically verify the specification and implementation, but tricky proofs may be developed interactively.

The KeY tool is one of the most extensive tools available for the software development process. The tool can be employed from the earliest stages of requirements acquisition all the way to verification of implementations. It provides excellent support for representation, implementation and reasoning of software. The limitation that implementations be in the JavaCard subset of the Java language is understandable, and doesn’t limit the tools strengths severely. However, the tool does rely on the external component of Borlands Together ControlCenter which already caused problems when Borland updated from Version 6 to Version 7. The KeY tool must be strongly considered as one of the most interesting and ambitious software development tools available.
3.3.3 Omnibus

Omnibus[70] is a development environment and language that is currently under development at the University of Stirling. The language incorporates Design by Contract, verified Design by Contract and Extended Static Checking. The language is Java-like and aims to incorporate much of the Java GUI library. Currently the tool only supports runtime assertion checking, but work is ongoing in developing extending static checking with the Simplify theorem prover and full verification with the PVS theorem prover.

This tool seems to offer good support of representation, implementation and reasoning but is still in too early a stage of development to be considered in this thesis. At the time of writing, the reasoning support was non existent but under development. The language itself appeared to be very similar to Java and JML to really warrant investigation. However, Omnibus is being developed with a good understanding of the issues that ESC/Java met with and has made many decisions to avoid and solve problems with respect to aliasing and inheritance and polymorphism.

3.4 Refinement

Refinement[1] is a formal process that transforms specifications preserving correctness. This is delivered through a series of refinement steps that translate an abstract specification into a more concrete one. The abstract specification is the simplest representation of the system that defines the interface to the external world. The concrete implementation is the program that realizes this specification. Refinement offers a way to associate specification and implementation, but requires complex mathematics to be proved correct. The suite of tools under discussion in this section are composed of technologies that verify this refinement technique. The B-Tool, ProofPower and Perfect Developer are
presented for their support of refinement.

3.4.1 B

B is possibly the most widely recognised tool for producing large software that is formally verified. The method is composed of the B-method[17], a method which supports iterative refinement of specifications, and a B-tool, which may be Atelier-B[52], B4Free[54] or the B-Toolkit[53]. There currently exist at least three tools that support the B-method, the first two are products of ClearSy while the B-Toolkit is produced by B-Core Limited. Software specification is written in the Abstract Machine Notation(AMN)[17] which has its mathematical foundations in the theory of Generalised Substitutions. Software Implementations are produced in standard C.

B has been used by industry and academia extensively on projects across the world. Most notably, it was used by Siemens in developing the Mtor driver less metro system in Paris[71]. However, its adoption has met with great resistance in industry owing to the need for high expertise in using the tool. AMN is a highly mathematical theory and developers often find it too cumbersome to use in day to day projects, relying on it only when correctness is essential. The tool that was researched for this thesis was the B4Free tool[54], a theorem prover with the most of the power of Atelier-B that is free to academics for research purposes. The tool, which runs on Linux, uses the Click’n’Prove user interface to develop and refine specifications.

3.4.2 ProofPower

ProofPower [72] is a tool that supports the refinement of Z specifications into Ada programs. Animation of specification and verification of the refinement is supported by a theorem prover based on HOL [73]. However, verification of the implementation is not supported by ProofPower.
3.4.3 Perfect Developer

Perfect Developer\cite{19} is a tool that supports verified design by contract and the refinement process through the object oriented programming language \textit{Perfect}. The tool has been launched by Escher Technologies\cite{74} and has been met with great interest in research and industry, being the focus of a number of projects\cite{75, 76, 77}. The Perfect language is one that support both specification and implementation notation, giving developers one language to develop in from specification to final implementation.

Perfect Developer seems to offer us the best of all worlds in its support of representation, implementation and reasoning. Perfect offers a specification and implementation language that is compact enough to be learned quickly, but expressive enough to specify software abstractly and then implement software efficiently. The tool provides an automated theorem prover to support verification of software without unnecessary input from the developer. It even supports the refinement of specifications into implementations and can be integrated with Java, C++ and Ada 95.

3.5 Conclusion

The review of support tools and technologies has provided us with a rich knowledge of the many ways software development may be supported. A review of the support tools currently available and their support of representation, reasoning and implementation is presented in the following tables. Table 1 presents the mechanisms of support and Table 2 presents the quality of this support.

We found that the Model Verification tools provided an excellent first stage in developing software, requiring the least user assistance for Reasoning. However, they are limited by not support the full software development life-cycle
<table>
<thead>
<tr>
<th>Tool</th>
<th>Representation</th>
<th>Implementation</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2B</td>
<td>UML+AMN</td>
<td>N/A</td>
<td>Auto Model Checking</td>
</tr>
<tr>
<td>SLAM</td>
<td>N/A</td>
<td>C</td>
<td>Auto Model Checking</td>
</tr>
<tr>
<td>Alloy Analyzer</td>
<td>Alloy</td>
<td>N/A</td>
<td>Auto Model Finding</td>
</tr>
<tr>
<td>LARCH</td>
<td>LARCH</td>
<td>C++</td>
<td>Man Theorem Proving</td>
</tr>
<tr>
<td>Z/EVES</td>
<td>Z</td>
<td>N/A</td>
<td>Auto Theorem Proving</td>
</tr>
<tr>
<td>Cafe</td>
<td>CafeOBJ</td>
<td>CafeOBJ</td>
<td>Auto Symbolic Execution</td>
</tr>
<tr>
<td>ESC/JAVA2</td>
<td>JML</td>
<td>Java</td>
<td>Auto Symbolic Execution</td>
</tr>
<tr>
<td>KeY Tool</td>
<td>UML+OCL</td>
<td>JavaCard</td>
<td>Auto Theorem Proving</td>
</tr>
<tr>
<td>Omnibus</td>
<td>Omnibus</td>
<td>Omnibus</td>
<td>Auto Symbolic Execution</td>
</tr>
<tr>
<td>B</td>
<td>AMN+GS</td>
<td>C</td>
<td>Man Theorem Proving</td>
</tr>
<tr>
<td>ProofPower</td>
<td>Z</td>
<td>Ada</td>
<td>Man Theorem Proving</td>
</tr>
<tr>
<td>Perfect Developer</td>
<td>Perfect</td>
<td>Perfect</td>
<td>Auto Theorem Proving</td>
</tr>
</tbody>
</table>

Table 1: Summary of Support Mechanisms

and do not scale well. Specification Animation tools provide excellent support of specifications, but as with Model Verification do not continue to provide support in the later stages of software development. The tools that supported a form of Extended Static Checking of software were excellent, with ESC/Java2 and the KeY tool being among the best for support the full software development life-cycle. However, Refinement was a much more powerful technique and the Perfect Developer tool seemed to offer a more well-rounded and easy to use software development tool. We feel the goals of Perfect Developer reflect our ideal software development better than the other tools we examined.
<table>
<thead>
<tr>
<th></th>
<th>Representation</th>
<th>Implementation</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2B</td>
<td>Good</td>
<td>N/A</td>
<td>Good</td>
</tr>
<tr>
<td>SLAM</td>
<td>N/A</td>
<td>N/A</td>
<td>Excellent</td>
</tr>
<tr>
<td>Alloy Analyzer</td>
<td>Good</td>
<td>N/A</td>
<td>Excellent</td>
</tr>
<tr>
<td>LARCH</td>
<td>Average</td>
<td>Average</td>
<td>Poor</td>
</tr>
<tr>
<td>Z/EVES</td>
<td>Excellent</td>
<td>N/A</td>
<td>Excellent</td>
</tr>
<tr>
<td>Cafe</td>
<td>Poor</td>
<td>Poor</td>
<td>Excellent</td>
</tr>
<tr>
<td>ESC/JAVA2</td>
<td>Excellent</td>
<td>Poor</td>
<td>Good</td>
</tr>
<tr>
<td>KeY Tool</td>
<td>Excellent</td>
<td>Good</td>
<td>Excellent</td>
</tr>
<tr>
<td>Omnibus</td>
<td>Good</td>
<td>Good</td>
<td>N/A</td>
</tr>
<tr>
<td>B</td>
<td>Excellent</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>ProofPower</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Perfect Developer</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Good</td>
</tr>
</tbody>
</table>

Table 2: Summary of Support Quality
4 The Perfect Language

This chapter presents Perfect, the language that is supported by the Perfect Developer tool. The language is an OO language that encompasses both specification and implementation details. It is not the intention of this chapter to act as a reference manual for the language but to act as a tutorial, providing a broad understanding of Perfect as an OO language. The language is illustrated with a running example highlighting the peculiarities of the language. The treatment of Perfect’s theory of programming is explored showing both syntax and semantics of the language. The method of specification and the process of refinement supported by the language are documented. Finally, novel language features that appear in Perfect are overviewed. For more information or to support reading of this chapter, the reader is directed to the online Reference Manual[78].

The running example that supports this chapter is a harbour management system, comprising of a harbour that contains a number of ports for docking ships. The software will be very simple and for illustrative purposes only. At any given time a port may have a ship docked in it or it may be vacant. As ships arrive, they make a request to be docked and are placed on a queue (see figure 1). The software aims to be as simple as possible while illustrating as many features of the language as possible.

4.1 Typing

Perfect offers four mechanisms to type software: Class; Enumeration Type; Constrained Type; and Generic Type. All four are introduced by the keyword class. A Class defines the attributes and behaviour of a set of objects as found in most object oriented languages. An Enumeration Type is a set of ordered values. A Constrained Type defines a set of allowed objects by adding
constraints to a previously defined type. No behaviour can be added to the type. A *Generic Type* is a class that accepts types as parameters, promoting re-use of code. These are the main concepts of type supported by Perfect.

### 4.1.1 Class

The standard means of introducing a type in Perfect is through the introduction of a *class*. The class mechanism is similar to most object oriented languages as has been discussed, but in Perfect it is segmented differently. Unlike most languages that use keywords to denote the visibility of features of a class, here features with common visibility are grouped together, located within a *section*, a body of code demarked by the keywords *abstract*, *internal*, *confined* and *interface*. There are three levels of visibility: Private features are defined in the *abstract* section; Features accessible to the class and any of its descendants are defined in the *confined* section; Public features are defined in the *interface* section. A class may also contain an *internal* section if a data refinement has occurred.

For example, the structure of a Harbour in our example occurs in the following manner:
class Harbour ^=
abstract
  // Specification Attributes are written here
internal
  // Implementation Attributes are written here
confined
  // Private Methods are written here
interface
  // Public Methods are written here
end;

The running example also requires a Ship class and a Port class that are structured similarly.

4.1.2 Enumeration Type

An Enumeration Type creates a new class where each element of the enumeration represents a ordered value, whose ordering is derived from their location in the enumeration. The only methods that apply to enumeration values are comparison operators that return the rank of the value with respect to other values of that enumeration type. The ranks of elements in an enumeration may be below, above or same. Functions are also defined on the class itself to return the highest or lowest element of the class using these keywords.

To illustrate, let us return to our running example and consider each ship as having a size that belongs to a ship size grouping. This set is defined as:

class ShipSize ^=
  enum
    Dinghy,
    Yacht,
    Barge,
    Ferry
  end;
4.1.3 Constrained Type

A *Constrained Type* is used to define a type representing a subset of possible instantiations of a pre-defined class. The notation will not allow additional features of the class to be introduced within this construct, but it is useful as a specification tool. In our example, it will be of use to keep track of those ships whose size is strictly smaller than a Ferry. Using a Constrained type, the class is defined as:

```plaintext
class SmallShips ^= those s:Ship :- s.size < Ferry@ShipSize;
```

It is assumed the class *Ship* has a method called *size* that returns the size of a ship object.

4.1.4 Generic Types

To promote re-use, Perfect provides the mechanism of Generic Types, a concept very similar to Template classes as found in C++[38]. This allows a class to have multiple type parameters that are instantiated when a variable of this class is declared. Several restrictions on the type parameters may also be encoded: it may require the type has definitions of equality or comparison; it may require the type contains some method; it may require the type be within an inheritance hierarchy. Each of these limitations can aid in providing some meaning to the class and prevent abuse of the mechanism.

In our working example, it will be required to keep track of the ships that arrive at the Harbour and are queued. To this end, a *Queue* class could be developed for handling incoming ships. Instead of developing a Queue class specifically for ships, a generic *Queue of X* class could be developed and then re-used whenever a queue is required. The class would have the form:

```plaintext
class Queue of X ^=
```
abstract
var queue:seq of X;
end;

This class would be instantiated in the Harbour system as:

class Harbour ^=
abstract
var incoming: Queue of Ship != Queue of Ship{};
end;

4.2 Class Attributes

As has been stated, Perfect provides two sets of attributes within a class, one for specification and one for implementation. Both sets use the same syntax introduced by the keyword var. In each var expression, an attribute identifier is provided followed by the attributes type. To illustrate this, we develop the Ship class with its specification attributes:

class Ship ^=
abstract
var myName:string;
var size:ShipSize;
end;

All ships to have been developed to have a name and a size. As a shorthand, we can run several var expressions together, replacing the intermediary semicolons with commas. The Port class could be defined as:

class Port ^=
abstract
var id:nat, // Port id
docked:Bay; // What ship, if any, is docked.
end;

Perfect also provides a mechanism for constraining the values of variables within a class. It is known as an invariant, borrowing terminology from the
Design by Contract[33] paradigm. The Harbour class will require there at least 1 Port object in its collection. This is captured by the invariant stating:

class Harbour ^=
abstract
var ports:set of Port, // all ports available
    incoming:Queue of Ship; // incoming queue of ships

invariant #ports >0; // a harbour has ports

This illustrates how Perfect populates a class with attributes. These examples all relate to the specification attributes of a class. The implementation attributes follow an identical syntax but are placed following the internal keyword as opposed to the abstract keyword. A class must have abstract attributes before internal attributes may be defined. All class attributes must be declared together within their associated section.

4.3 Class Methods

The behaviour of an object is defined through the methods of the class of that object. In Perfect there are three varieties of methods: Constructors; Functions; and Schema. Constructors are the methods that create a new instance of a class and must instantiate all the attributes of the class and ensure the invariant holds. Functions return the result of some evaluation based on parameters passed to the function or the current objects attributes. Schema are state changing methods that return no result but may change the state of the current objects attributes or the state of parameters passed to the schema. If a function or a schema do not rely upon class attributes they may be declared as non-member.

To illustrate the three varieties of methods, we will develop the Port class further.
4.3.1 Constructor

In Perfect, the constructor has a special syntax using the keyword build and parameters may be passed inside curly braces { }. A constructor that accepts values for all parameters will be written and placed in the confined section of the Port class to be accessible to the class and its descendants.

```plaintext
confined
build{portId:nat,isDocked:Bay}
  post id!=portId,
   docked!=isDocked;
```

The constructor states as its post-condition that the value of id will become equal to the value of portId and the value of docked will become equal to the value of isDocked. The two statements are equivalent to assignment statements. As the assignments are separated by a comma, the order of execution is treated as unimportant and if possible, they may occur in parallel.

A public constructor can be developed to take advantage of this private constructor.

```plaintext
interface
build{i:nat}
  ^= Port{i,Bay{}};
```

This constructor is defined as equivalent to calling the previous constructor. Perfect offers a shorthand to constructors that only need to instantiate attributes of the class from the parameters. By placing the attribute identifiers in the constructor and preceding each with a ! symbol, the constructor requires no additional post-condition. This is illustrated by the constructor to the Ship class:

```plaintext
interface
build{!myName:string,!size:ShipSize};
```
This constructor instantiates the objects attributes using the parameters of the same name.

4.3.2 Function

In Perfect, those methods that are side-affect free are preceded by the keyword function. The functions of a class return results based on evaluations of the state of the object and any parameters passed to the function. A basic function can return the value of some attribute without performing any calculations. These “getter” functions have the form:

```perfect
function getShipName:string
  ^= myName;
```

This cumbersome syntax can be shortened in Perfect by stating the attribute name directly after the keyword function, such as in returning the size of a Ship object:

```perfect
function size;
```

Some functions may require additional parameters to be passed to the object. In these cases, the parameters follow the function name and are placed within brackets, providing a parameter name and type. For example, there may be an inquiry on a harbour object to see if a particular ship is docked at the harbour. This would be written as:

```perfect
function isDocked(s:Ship):bool
  ^= (let shipsDocked ^= for b::ports yield b.dockedHere;
      s in shipsDocked);
```

Here the function creates a temporary set of all the ships docked in the harbour, using a let statement, and then tests if the parameter object is in the set.
4.3.3 Schema

In Perfect, schemas are used to change the state of the system either through an object’s attributes or through some of the parameters passed to the schema. The keyword `schema` is used to distinguish these from other kinds of methods. When a schema changes the state of the object it is called upon, the schema name is preceded by the `!` symbol. For example, it is expected that the Harbour object will change to accommodate arrival of a ship. This would be declared as:

```perfect
schema !arrive(s:Ship) post ([haveVacantPorts]: // if vacant
    !land(s,vacantPort), // land
    []: // else
    incoming!=incoming.append(s) // add to queue
);
```

where the `!` appears before the method name to state that the current object will be changed. The keyword `post` is used to declare the post-condition of a schema. The post-condition defines the changes on the system that occur when the schema is called. In this case, what occurs is the ship is docked at a `vacantPort` if the harbour contains any or else joins the incoming queue and waits for a port to become vacant.

The example also illustrated one of the selection mechanisms of Perfect. The `[p]` can be read as a conditional guard on a statement, where `p` is the choice predicate. The guards are mutually exclusive and at least one guard must contain a predicate. If there is no predicate, the guarded statement occurs when all other predicates in the same conditional statement fail.

4.4 Assertions

An assertion is a boolean expression that when evaluated should always evaluate to true. The expression in Perfect is introduced by the keyword `assert` and
can be embedded within a code fragment at the specification or implementation level. Assertions in Perfect are supported by a rich language to enable most statements to be made clearly and without significant effort. The language contains standard quantifiers and a selection of useful higher order functions that will now be presented.

The universal and existential quantifiers are supported by the language. They are introduced by the keywords \texttt{forall} and \texttt{exists} respectively. The quantifiers may be applied on any class of data with the exception of reference types (values declared on a heap) or, more commonly, on a data collection type. The quantification must be provided with a predicate that is to be applied to the data domain. The result of quantification is a boolean value. For example, we may wish to declare that all ships have a unique id, which could be declared as:

\begin{verbatim}
assert forall x:Ship :- forall y:Ship :- x\neq y ==> x.id \neq y.id;
\end{verbatim}

Perfect also contains some of the most common higher order functions from the function programming paradigm. These include the techniques of folding, mapping and filtering but, in Perfect, are termed \textit{over}, \textit{transforms} and \textit{choice} respectively. However, with the exception of choice, the higher order functions can only be applied to the built in collection data types. Their form can be found in the language reference manual[78].

4.5 Specification

Beyond the language of programming that has been presented so far, Perfect contains a collection of specification elements that do not produce code, but allow better documentation of software and its meaning. The lightest of these techniques is \textit{Design by Contract}, wherein the conditions of usage of a software
fragment are encapsulated in a contract that specifies behaviour. Some functions
can be declared as *Satisfy Statements* that captures the result of the function
in a predicate that may be non-deterministic. The most high level specification
technique provided by Perfect is that of *Algebraic Properties*. An algebraic
property can state requirements of the software promoting validation within
the code development. Each of these techniques will now be presented and
illustrated by the Harbour example for clarification.

### 4.5.1 Design by Contract

Design by contract is the mechanism through which Perfect supports
lightweight specification. It provides this mechanism in the form of pre-
conditions, post-assertions and class invariants. The pre-conditions and post-
assertions define the contract that a function or schema adheres to. The class
invariant restricts the range of values the class may be instantiated to.

The keywords that Perfect adopts for pre-conditions is *pre* while post-
assertions are *assert* statements that follow the method definition. Both ex-
pressions may involve the values of the class attributes (at the specification level
only) or the function parameters. The post-assertion may reference the value
of these variables at both the time of calling the method and at the methods
termination. To distinguish between the variables, a priming of the variable
name occurs to denote the final (or current) value of the variable.

To illustrate this, consider our Harbour class and a method that allows a
ship to depart from the harbour. The method is defined as a schema, changing
the state of the Harbour object on which it is called. It is assumed that any
ship departing the harbour must already have been docked there and that once
it departs it is no longer docked there or arriving at the harbour. This defines
the contract in English. The contract obligation defined in Perfect would be:

```
schema !depart(s:Ship)
  pre isDocked(s)
    // An implementation will go here
  assert ~isDocked(s) & s ~in incoming;
```

When defining the invariants of a class, the `invariant` keyword is used. The invariant is also an assertion that may reference any attributes or functions that are declared prior to the declaration of the invariant. As Perfect is not truly modular, invariants may be broken by code fragments declared before the invariant. Therefore the placement of invariants near the beginning of a class file is essential. To illustrate a simple invariant, consider a restriction on our class that prevents ships from being both queued and docked. This is obviously something we would like to prevent and an invariant is placed on the class. To states that there exists no ship that is docked that is also in the incoming queue, we would write:

```
invariant ~(exists s::Ship :- isDocked(s) & s in incoming);
```

The invariant of the class can become complex, relying upon several helper functions to define some safety property in the system. For example, it would be of benefit if the harbour system never allowed a ship to queue while there was a port available. To include this as part of an invariant will require the following definitions:

```
function vacantPorts:set of Port // the ports available
  ^= those p::ports :- p.vacant;

function haveVacantPorts:bool // are there vacant ports?
  ^= #vacantPorts >0;

function hasQueued:bool // are there ships queued?
```

42
\[
\text{^= ~incoming.empty;}
\]

// if we have vacant ports, we don’t have ships queuing
invariant haveVacantPorts ==> ~hasQueued;

As has been stated the assertion language of Perfect is rich and can capture, in one way or another, a developers intentions and specification. The language is ideally suited to Design by Contract providing quantifiers and permitting helper functions.

### 4.5.2 Satisfy Statements

One of the most unique aspects of Perfect, is the ability to define a function using a predicate using a *Satisfy Statement*. Replacing the \(\neq\) symbol with the keyword `satisfy` when defining the function, the language permits the encoding of the function such that the result satisfies some predicate. The predicate must contain the keyword `result` referring to the value the function must evaluate to, but this value can be non-deterministic in the context of the specification.

In our example, we have already referred to a function called `vacantPort` that returns some vacant port in the harbour. At the time of specification, there was no need to specify how to find this port, just that it was vacant. This is coded as:

```plaintext
function vacantPort:Port
    satisfy result.isVacant;
```

It relies on a function `isVacant` in class `Port` that returns a boolean value declaring if the Port object is vacant.

### 4.5.3 Algebraic Properties

Perfect also allows for some elements of algebraic specification to be encoded alongside the object oriented specifications. These *Algebraic Properties* define
constraints on the class in terms of its behaviour. They define the state that results from applying some sequence of methods to an object. They can be used for capturing requirements of software in the early stages. The keyword **property** is used to define an algebraic property and is usually followed by an **assert** statement that defines the property. Sometimes, it is beneficial to include parameters when declaring a property as they can represent a set of possible values of their type, using a pre-conditions.

To illustrate the use of **property**, consider the harbour example. It should always be the case that when a ship arrives and there are vacant ports, the ship will be docked instantly. This cannot be stated easily within a contract, but using the property mechanism is declared as:

```plaintext
property(s:Ship)
  pre hasVacantPorts
  assert (self after it!arrive(s)).isDocked(s);
```

The **self** keyword refers to the current object. The **it** keyword refers to the object referenced before the keyword **after**, in this case the **self** object. The property applies the **arrive** schema to the **it** object (i.e a copy of the self object). This resulting new object then has the method **isDocked** called upon it. By stating this sequence as a property, the semantics of the class is defined clearly and concisely.

### 4.6 Inheritance and Polymorphism

Like most object oriented languages, Perfect supports inheritance and polymorphism to a large extent. Both concepts are closely related in the paradigm and are often the source of much debate, confusion and errors. Perfect overcomes these areas of difficulty by encoding a fault proof system of inheritance and polymorphism that will be presented below.
4.6.1 Inheritance

All classes in Perfect inherit from the anything class which provides a toString method and is implicitly inherited. To explicitly inherit from a parent class, the child class is defined as normal with the insertion of the keyword inherits followed by the parent class in the class name declaration. When this declaration is made, all attributes and features of the parent class are included in the new class but only those features declared in the confined and interface sections are accessible to the child class. New attributes and methods may be added as with a new class but the old methods may be overridden or even removed.

To override a method of a parent class within the child class, the specification of the method must be taken into consideration. If the method has any pre-conditions, these pre-conditions must be retained or weakened by the new method. Similarly if the method has any post-assertions this must be retained or strengthened by the new method. Perfect will not allow the method to be overridden unless the keyword redefine is placed before the function. To hide a function, the function is declared as absurd and the functions pre-condition is set to false. By doing this, the function can never be called. Absurd functions may be unhidden further down the inheritance hierarchy.

To illustrate inheritance consider adding a FastShip class that inherits from the Ship class. Lets suppose that all ships have an associated maximum speed and that all fast ships have a maximum speed above a certain limit. This would be encoded as:

```plaintext
class FastShip inherits Ship ^=
  interface
    function maxSpeed:nat
      ^= max
```
assert result > 50;

Assume there is another class called \texttt{BrokenShip} representing ships that are damaged and can no longer move. It would be pointless to ask what the maximum speed of a broken ship is, so this would be declared as an \texttt{absurd} method.

\begin{verbatim}
class BrokenShip inherits Ship :=
  interface
    absurd maxSpeed:nat
    pre false;
\end{verbatim}

Here we have made it impossible for any calls to \texttt{maxSpeed} to be made on all \texttt{BrokenShip} objects.

4.6.2 Polymorphism

In most object oriented languages polymorphism through inheritance is supported by default but not in Perfect. Rather than accepting an instance of a descendant class whenever a variable of some ancestor class is declared, Perfect will only accept instances of the ancestor class. This is provided to guarantee against errors when methods are hidden or specifications adapted in descendant classes. But Perfect does allow for this type of polymorphism through the use of the \texttt{from} keyword. Whenever a variable is declared as \texttt{from Ship}, the variable may contain any instance of a Ship class or any descendant of the Ship class.

Another form of polymorphism that Perfect supports is that of \textit{United Types}. A united type is a type that is constructed from the union of two or more other disjoint types. The operator of union is two vertical bars $||$ and is commonly used to allow objects have a \texttt{null} value. For example, if we wanted to allow a variable hold either a Ship or a void object, it could be declared as:

\begin{verbatim}
var holder: void||Ship;
\end{verbatim}
4.7 Refinement

The most novel concept in the Perfect language is its treatment of Refinement. As has been discussed in Section 3.4, refinement is a formal process that permits the translation of specifications to implementations. It is supported within the Perfect language as a single step of specification to implementation within a class. In this step, the developer may: refine a method specification to an implementation; refine a data structure through addition or transformation; or refine the specification itself adapting it to accept more possibilities or guaranteeing a stronger system. Each form of refinement has been written about in depth in [79]. For the purposes of this work, refinement of data structures and refinement of implementations are presented for use in Chapte 6.

4.7.1 Data Refinement

Data Refinement is perhaps the most intriguing element of Perfect. It provides the user with a mechanism to enrich a specification with implementation details that, while unnecessary for the specification, are highly valuable to efficiency of the software. There are two forms of this refinement: Algorithm Introduction and Transformation. Algorithm Introduction simply augments a specification with additional attributes whose purpose is to store values that are repeatedly derived in the software. Transformation alters the data specification completely, introducing a new data structure for the one specified in a class. Whenever data refinement occurs, algorithm refinement must occur on all methods of the class, regardless of whether the change impacts the method.

To illustrate transformation, consider the collection of Ports in the Harbour example. The Ports have been randomly collected within a set object for simplicity, but as accesses usually will either involve those ports that are open, (i.e.
no ships docked) or those ports that are closed, (i.e. a ship is docked), we can refine the set by splitting the set into two sets. The new data structure will be as follows:

```plaintext
class Harbour ^= 
abstract // Specification of model
  var ports:set of Port; // all ports available
internal // Data Refinement

  var openPorts:set of Port, // vacant ports
  closedPorts:set of Port; // ports in use

function ports // retrieve function
  ^= openPorts++closedPorts;

// No port is vacant and in use
invariant openPorts**closedPorts=set of Port{};

// open ports are vacant & closed ports are not
invariant forall p::openPorts :~ p.vacant;
invariant forall p::closedPorts :~ p.vacant;
```

Two internal data variables are declared that will represent the `openPorts` and the `closedPorts`. A `retrieve function` is then declared that defines how the abstract variables can be constructed from the internal variables. This function is essential to performing transformation and must be in this direction owing to Perfects treatment of refinement. Finally some invariants are declared to give a semantics of the internal variables.

### 4.7.2 Algorithm Refinement

Algorithm Refinement as implemented in Perfect allows a code fragment to be replaced with an implementation of it. Refinements may be made on the methods of a class or on any statements within a method. The refinement is introduced by the keyword `via`. If code contains a `satisfy` statement, this must
be refined before it may be compiled and executed. Algorithm Refinements may also take place as a result of a data refinement occurring. They are identical in syntax to algorithm refinements alone, but any algorithm refinement must be re-written after a data refinement has occurred.

In our example, all the class methods must be refined. To illustrate, consider two of the methods: The first a `satisfy` function that returns a vacant Port, found through some mechanism; The second a method that checks whether a ship is docked in the harbour.

```plaintext
function vacantPort:Port
    satisfy result.vacant;
    via value openPorts.min
end;

// is s docked in our harbour?
function isDocked(s:Ship):bool
    ^= (let shipsDocked ^= for b::ports yield b.dockedHere;
        s in shipsDocked
    )
    via let shipsDocked ^= for b::closedPorts yield b.dockedHere;
        value s in shipsDocked;
end;
```

In the first method, the original specification offers no algorithm for finding a vacant port, just that the `result` would be vacant. Following the refinement, an algorithm for finding a vacant port is provided. In the second method, the original specification suggests an algorithm that checked all ports in the harbour, collecting a set of ships docked. This search would involve checking all those ports that have no ships docked at them. In the refined algorithm, the search is cut, removing the need to search those ports that have no ships docked at them.
4.8 Conclusions

This chapter presented the Perfect language and has guided the reader through an example of the language in use. The language can be considered an object oriented language in its support for object oriented features. It supports high and low level specification of software that can assist in requirements development phases and software documentation. It contains several novel language features that are combined in a new way to set it apart such as generics, higher order functions, and refinement. It can be used to assist developers in thinking about software in much the same way as Eiffel or Object-Z but unlike those languages it is immediately supported by a tool that provides verification and validation features. With knowledge of the Perfect language, a closer inspection of the Perfect Developer tool can be undertaken.
5 Perfect Developer

This chapter presents the Perfect Developer tool[19] and its support of the software development process described in Chapter 2. The tool is comprised of four components: the Project Manager; the Perfect Language; the Verifier; and the Compiler. The Perfect Programming Language was described in Chapter 4, hence we focus our discussions here on the remaining three components of Perfect Developer. The support which these three components give to the software development process is illustrated by returning to the Harbour Example as introduced in Chapter 4.

5.1 Project Manager

The Project Manager Component of the Perfect Developer tool is an application that can be run on most modern PC’s with a Pentium compatible processor and at least 256MB of memory. The application will run under Windows XP, Windows 2000, Windows NT4 or Linux. The Project Manager consists of a graphical user interface in charge of the source code maintenance and the interactions with the other components of the tool. The Project Manager does not contain a UML graphics package or a Text Editor, but can be integrated with third party software that provides this functionality.

Our investigations used Perfect Developer version 2.10 installed on a Windows 2000 running on an Intel Pentium 3. The software installation was simple to perform from the Perfect Developer Installation Disc which can be acquired from Escher Technologies for a fee depending on your usage[80]. The latest version is Perfect Developer 3.1, which includes some new language features and an improved Verifier component. Changes from Version 2.1 can be found on the Escher Technologies website[80].
5.1.1 Graphical User Interface

The Graphical User Interface (GUI) for Perfect Developer is a single window that provides control of the entire development process in Perfect Developer (see Figure 2). The most common actions available to a developer are given buttons on the toolbar (see Table 3). Project settings and build configurations may be altered through the pop-up dialog boxes.

Navigation with the Perfect Developer GUI is simple on small projects, but will not be ideal as software scales up. The Files Window (19) is the focus of the GUI but will become cluttered on a project of many source files because it uses absolute addressing of files. The Results Window (20) can also become cluttered during checking, verification and compilation of software projects with many spurious comments that are of no concern to the developer. The level of error reporting can be set to hide correct verifications and warnings, but not
other control mechanism exists.

### 5.1.2 UML Importer

As Perfect is object oriented, integration with the UML[21] notation would be of great benefit. Rather than re-inventing the wheel by developing their own UML application, Perfect Developer allows developers to import UML diagrams from a *.xmi file. To illustrate, using a third party editor like Argo UML[81], the UML diagram in Figure 3 was constructed, describing the Harbour management software.

The diagram is exported to the *.xml format (see Appendix A for code). This file can then be selected and imported into the current Perfect Developer project. By importing the project, all class names mentioned in the UML diagram are automatically generated and added to the project. All methods

Table 3: Controlling Perfect Developer

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>New Project</td>
</tr>
<tr>
<td>2</td>
<td>Open Project</td>
</tr>
<tr>
<td>3</td>
<td>Save Project</td>
</tr>
<tr>
<td>4</td>
<td>Configuration Manager</td>
</tr>
<tr>
<td>5</td>
<td>Settings</td>
</tr>
<tr>
<td>6</td>
<td>Build</td>
</tr>
<tr>
<td>7</td>
<td>Re-build</td>
</tr>
<tr>
<td>8</td>
<td>Check</td>
</tr>
<tr>
<td>9</td>
<td>Verify</td>
</tr>
<tr>
<td>10</td>
<td>Cross Reference</td>
</tr>
<tr>
<td>11</td>
<td>Stop</td>
</tr>
<tr>
<td>12</td>
<td>Clean</td>
</tr>
<tr>
<td>13</td>
<td>User Guide</td>
</tr>
<tr>
<td>14</td>
<td>Language Reference</td>
</tr>
<tr>
<td>15</td>
<td>Exit</td>
</tr>
<tr>
<td>16</td>
<td>Add</td>
</tr>
<tr>
<td>17</td>
<td>Create</td>
</tr>
<tr>
<td>18</td>
<td>Remove</td>
</tr>
<tr>
<td>19</td>
<td>Files Window</td>
</tr>
<tr>
<td>20</td>
<td>Results Window</td>
</tr>
</tbody>
</table>
and their signatures are also added.

One problem with this process arises from difference between Perfect semantics and UML semantics. As we stated, Perfect does not support polymorphism through inheritance by default but the UML does, resulting in all parameters having the from annotation. Unfortunately the tool does not currently support OCL[82] integration even though the languages share the design by contract specification mechanism. The tool currently doesn’t offer a mechanism to export Perfect Developer projects into *.xml files, removing the ability to keep documentation synchronised.

5.1.3 Text Editor

Like the UML translator, Perfect Developer does not enforce a single text editor giving developers the freedom to use their own favourite. The tool does allow many text editors to be integrated, allowing quick file access and syntax error highlighting if the editor supports this feature. Perfect Developer comes equipped with several text editor customization packages, such as TextPad4[83],
Multi-Edit 9[84] and Crimson Editor[85].

5.1.4 Syntax Checker

Syntax Checking in Perfect Developer is performed by pushing the Check Button (8) and errors are reported in the Results Window (20). Perfect Developer does not usually provide good results when it checks the syntax. Results very often refer to the line subsequent to the error, costing development time when mistakes occur.

As a side note, the language itself is quite difficult to debug. Statements do not follow a regular pattern as found in most languages. The ; symbol may follow a post-condition, unless there is a post-assertion, in which case it follows that. After a refinement, the ; symbol again moves to follow the refined algorithm. Very commonly, errors reported by the syntax checker will occur owing to an unmoved semi-colon but the error will be reported about the feature that follows this mistake.

5.2 The Verifier

The Verifier is the backbone of Perfect Developer providing Horizontal and Vertical Verification of specifications and refinements respectively. Horizontal Verification ensures that specifications are consistent and error-free. Vertical Verification ensures that an implementation is a correct refinement of the specification. Both kinds of verification are performed when the developer clicks on the “Verify” button(9). Usually, horizontal verification is performed in advance of refinement to eliminate the refinement of incorrect specifications. However, this separation of verifications cannot be made by the Verifier.

The Verifier parses the source file collecting proof obligations and then attempts to prove these with by execution of its theorem proving engine. The
entire process is automated and can only be halted by pressing the “Stop” button(11). Parameters may be set to limit time and memory consumption on each proof attempt. This prevents the existence of infinite proof attempts by the Verifier. At the time of writing, very little detail is available on the Verifier. However, we will present some verification attempts from the Verifier in order to demonstrate how it operates on several kinds of verification.

5.2.1 Contract Verification

The Verification of Contracts is one of the most common tasks that the Verifier faces. There are kinds of proof obligation that arise with contracts: Establishing Pre-Conditions; Ensuring Post-Conditions; and Invariants. The first is gathered whenever a function with a pre-condition is called. The second and third are gathered whenever a function is declared provided the function has post-conditions or its class has an invariant. Each obligation is proved through symbolic execution of the code followed by a unification attempt.

Returning to our Harbour example, consider the symbolic execution of the depart schema:

```plaintext
schema !depart(s:Ship)
pre isDocked(s)
post (  
  let dockedPort^= myPort(s);  
  let emptyPort^=dockedPort.Clear;  
  ports!=ports.remove(dockedPort) then //no longer docked  
  ports!=ports.append(emptyPort) then //now empty  
  ([hasQueued]: !landQ(emptyPort), // land queued ships or..  
   []:pass // ..do nothing
  )
)
assert ~isDocked(s) & ~incoming.has(s);
```

During the execution of this schema, the pre-conditions of myPort and landQ are collected as proof obligations to be dispatched. The post-assertion is also
collected. Using the invariant:

\[
\text{invariant } \neg(\exists s :: \text{Ship} : - \\
\quad (\forall p1, p2 : \text{ports} : - (p1 = p2 \lor \\
\quad (s1 = p1.\text{dockedHere} \land s = p2.\text{dockedHere}))));
\]

which states that no ship is docked at two distinct ports, the Verifier can prove that \( s \) is not docked after \text{dockedPort} is removed from \text{ports}. Similarly, using the invariant:

\[
\text{invariant } \neg(\exists s :: \text{Ship} : - \text{isDocked}(s) \land \text{incoming}.\text{has}(s));
\]

which states that no ship is docked and is incoming, it can prove the \text{incoming} queue does not contain \( s \).

The proof obligations gathered through the symbolic execution of code fragment still need to be dispatched. The first obligation, proving the pre-condition of \text{myPort} is easily dispatched as it is simply a check that \( \text{isDocked}(s) \) is true, a fact implicit from the pre-condition of the method.

5.2.2 Inheritance Verification

Perfect allows over-riding of methods when inheritance occurs, but this can cause problems if the over-ridden method is defined by a contract. Owing to polymorphism by inheritance, a call which expects the over-ridden method could execute the new method. For this reason, when over-riding a method, the pre-condition can only be as strong as the pre-condition of the inherited method. Similarly, the post-condition can only be as weak as post-condition of the inherited method.

Consider the \text{FastShip} class described in Section 4.6.1, where we defined a \text{maxSpeed} where the result had to be above 50. Let us suppose a new class of ship was called for, \text{SuperFastShip} which inherited from \text{FastShip} with the
added assurance that the maximum speed is above 100. It would be encoded as:

class SuperFastShip inherits FastShip
  interface
    function maxSpeed:nat
      ^= max
    assert result > 100;

In this case, the verification would be successful, because it can be proved that:

forall s::SuperFastShip :- s.maxSpeed > 50
  (post-condition of FastShip::maxSpeed)

5.2.3 Property Verification

Properties of a class must also be verified during this phase. This verification assists in proving that the software meets the technical requirements of the client. The properties can state the behaviour of the class or even represent test cases. Symbolic Execution of a partially instantiated object is performed and unification with property is attempted by the Verifier.

Consider the property that ships arriving at a vacant port are immediately docked, defined by:

property(s:Ship)
  pre hasVacantPorts
  assert (self after it!arrive(s)).isDocked(s);

This can be verified by Perfect Developer as the specification of the arrive schema guarantees a docking if the Harbour object has vacant ports.

5.2.4 Refinement Verification

Verification that method implementations meet their specification is the only form of Refinement Verification found in Perfect. If this algorithm refinement is
as a result of a data refinement, the Internal Data Structure is used to perform this verification. If a refinement cannot be verified, it may signal an incorrect data refinement, an incomplete specification or an incorrect algorithm refinement. When a failure to verify a refinement occurs, the information provided by the Verifier is limited requiring much time to be spent searching for a reason.

Consider the refinement of the Harbour that was presented in section 4.7. When initially presented to the Verifier, we had neglected to include the specification that:

```plaintext
// open ports are vacant & closed ports are not
invariant forall p::openPorts :- p.vacant;
invariant forall p::closedPorts :- ~p.vacant;
```

This had the net effect of causing the Verifier to fail in its attempt to verify the project.

5.3 The Compiler

The Compiler is used to compile Perfect source code into Java, C++ or Ada95 source code. It can also generate the executable files in certain circumstances. The Compiler assists fast prototyping of software by allowing most specifications to be compiled automatically before refinement. All implementations in Perfect can be compiled after the refinement stage. Compilation does not require a verification of the software to be made, but no guarantee of correctness can be made without performing that step.

Compilation is performed through execution of the “Build” or “Re-Build” functions (6,7). A build generates source files for all the classes in the project. A Re-build deletes all previously compiled source files in the project and generates new ones. Software may be compiled to Java, C++ or Ada95 depending
on users preferences. Additional pre-build or post-build instructions may be included in the compilation process to generate executable versions of the software immediately.

5.3.1 Prototyping

Prototyping is one of Perfect Developers greatest strengths as most specifications can be prototyped instantly. The only exception to this occurs in the case of Satisfy statements (Section 4.5.2) which have to be refined before compilation may continue. Commonly, the prototype can be tested by using a command line interface that is provided with Perfect Developer. This interface uses a single method, the main schema, with the signature:

```plaintext
schema main(context!: limited Environment, args: seq of string,
             ret!: out int)
pre #args > 0
post(var h:Harbour!=Harbour{1}, // a harbour of size 1
     s1:Ship!=Ship"1", // some ships
     s2:Ship!=Ship"2";

   // Arrival of ships to the harbour
   h!arrive(s1) then
   context!print("S1 arrives\n") then
   context!print(h.toString++"\n") then

   h!arrive(s2) then
   context!print("S2 arrives\n") then
   context!print(h.toString++"\n") then

   // Departure of ships from the harbour
   // Includes landing as ships leave in sequence

   h!depart(s1) then
   context!print("S1 departs\n") then
   context!print(h.toString++"\n") then

   h!depart(s2) then
   context!print("S2 departs\n") then
```

60
context!print(h.toString++"\n") then

// End of test

context!print("Harbour empty")
)then
ret! = 0;

This method accepts an Environment parameter defined by context that performs the input and output operations of the class. It can read from and write to the display, perform file handling operations and retrieve information about the system the program runs under. The args represents a sequence of string valued parameters passed to the program at the point of execution. The ret variable may be used for error control, but is generally ignored unless required.

Prototypes built in this fashion offer very limited behaviour, but can illustrate much of the features and functionality of the software under construction. Test case analysis may be performed at this stage and pre-condition, post-condition and invariant errors reported to the developer helping to isolate the cause of errors in the software.

5.3.2 Implementations

The compilation of implementations by Perfect Developer can be performed in the push of the Build button. The only difference is if the developer wishes to develop a graphical user interface for the application. Perfect Developer does not support the development of these user interfaces and therefore such an undertaking lies outside the domain of the tool. However, it must be noted that the development of such an interface relies on the construction of a Wrapper Class to interact with the Perfect Developer backend. This will be discussed in further detail in Section 6.1.5 as it is not integral to the presentation of Perfect
5.4 Conclusions

It has been demonstrated to the reader that Perfect Developer may be of great use to the software development process. The benefits the tool are multiple: Satisfaction that the software meets the requirements; Assurance that the software is error free; Providing early prototypes that can be tested either manually or automatically. The Project Manager is basic but performs its task well. The Verifier can verify most obligations, but poor documentation of its theorem proving capabilities make failures hard to understand and correct. The Compiler produces code easily, but does require additional work to get right. It must be noted that many tools can amaze on simple examples, but fail to scale well in real world software applications. In the chapter which follows, we consider how well Perfect Developer scales up to medium and large scale software projects.
6 Case Studies with Perfect Developer

In this chapter, we describe the development of two medium scale case studies constructed with the assistance of Perfect Developer: A Library Database and a Resource Manager. We developed both of these case studies to contribute to the evaluation of Perfect Developer. They are designed to illustrate two exemplars of software, database software and reactive software, respectively. The developments are presented to illustrate the support provided by Perfect Developer with respect to Representation, Implementation and Reasoning. In each area, a presentation of the topic is accompanied by a discussion of the support provided by Perfect Developer. We begin each case study by presenting the requirements of the software.

6.1 Case Study 1: Library Database

Kemmerer’s library database problem has been an illustrative tool for specification languages and development methods since 1985[86]. Wing categorized 12 approaches to the problem in her paper[87]. The problem was the development of a university library database. The database had two kinds of users, borrowers and staff and had the following transactions available on the database:

- Add/remove a copy of a book to/from the library (staff only)
- Return/Borrow a copy to/from the library (staff only)
- Get the titles of books by some set of authors/subjects
- Find out a list of books currently borrowed by any user (staff only)/by yourself (user only).
- Find out which user last borrowed a copy (staff member only)
The database also had the following requirements:

- All items must be either checked out or available

- No item may be both checked out and available

In the original specification, it was assumed by the requirements specifier that the Library will have some staff and some borrowers. We extended the specification to include operations to add or remove users to the software. In a real system there would be some restrictions on these operations, but for the sake of brevity these restrictions are ignored.

6.1.1 UML

The structure of the Library Database is represented by the UML class diagram in Figure 4. The top level of the software is the LibraryDB class. This class distributes queries made upon the software to the required modules, the LibraryStock module for stock queries and the UserBase module for user queries. The LibraryStock module manages a LibraryCatalog that contains all the information of books known by the database and the current stock levels, represented by a collection of Borrowings. The UserBase manages the BorrowerBase and StaffBase keeping records of borrowers and staff respectively.

The skeleton code found in Appendix E was automatically created by Perfect Developers UML Importer (See Section 5.1.2) importing an .xmi file of the class diagram presented in Figure 4 into a new Project created by the Project Manager. The skeleton code inserts ? symbols where specification or implementation details need to be added. It also creates several spurious source code files representing both the nat and string classes. Many manual modifications to the generated skeleton code were required before any development could have
Figure 4: UML Model of the Library Database
begun. All variable types in this skeleton code were defined as from X to provide polymorphism. This typing is not necessary to our software and could potentially hinder verification in later phases of development. All attributes have been declared as selector functions, permitting over-riding of attributes in the software which is also not required and could potentially disrupt the specification.

We found the extensive editing of the skeleton code made using the importer redundant. The cost of reviewing and correcting the code could easily be avoided by manually constructing the specification from the beginning which would probably be quicker in the long run. The importer, while a good idea, does not work well with Perfect Developer unless more information can be provided in the UML design which would prevent the kind of mistakes that we met. Another major problem is the lack of support to export software to an .xmi file. If the software is changed during the development, the original UML model will also require manual alterations.

6.1.2 Specification

The complete specification of the Library Database can be found in Appendix F, while we present a selection of the most interesting specifications here. These specifications demonstrate the support for representation that is provided by Perfect Developer.

The requirement that “all items be either checked out or available” is ensured by the structure that we imposed on the software. The set of items that are checked out are defined as a set of Borrowing objects. Therefore, the specification of checkedOut is:

\[
\text{function checkedOut:set of LibraryItem}^\text{= for b::currentlyBorrowed yield b.getItem;}
\]
This has the benefit of defining the available items as those that the library has, less the items that are checked out:

\[
\text{function available: set of LibraryItem} \\
\quad = \text{catalog.allKnownItems} \setminus \text{checkedOut};
\]

This simplifies the specification greatly ensuring the second requirement: that no item is both checked out and available.

To ensure that an item not known by the LibraryDB is not borrowed, we include the specification:

\[
\text{invariant } \forall a::\text{currentlyBorrowed} : \neg \text{catalog.has(a.getItem)};
\]

We also ensure that the borrowing and returning of an item has no net effect on the LibraryDB, with the algebraic property:

\[
\text{property(i:Item, b:Borrower, d:Date)} \\
\quad \text{pre } \neg \text{itemAvailable(i)} \\
\quad \text{assert (self after it!borrowLibraryItem(i, b, d)} \\
\quad \quad \text{then it!returnLibraryItem(i)) = self;}
\]

In developing the specification, we always found at least one way to specify what was required by the software. Perfect is rich enough to capture the requirements at a variety of levels, from abstract properties to the specification of method contracts. Struggles with definition were rare, but occasionally appeared when nesting quantifiers. Overall, the support for representing the Library Database software could not be faulted.

6.1.3 File Handling

Permanent storage of any database is a necessary component to allow the software to terminate without losing all the information. Therefore, an implementation of the Library Database should encorporate this permanent storage
issue. However, the Perfect File Handling mechanism left much to be desired for providing limited features through an uncomfortable interface. File Handling in Perfect is controlled through global access to an Environment object available through the GUI (See Appendix H.7). This object provides the ability to create, read and write to files directly.

The Environment class describes a File abstract data type, but this data type is unused in all of the file handling mechanisms. Instead, Perfect Developer provides developers with file input/output via a FileRef object, (essentially a pointer to a block of memory). With this FileRef, the developer may read from a block of data interpreting the data as a raw byte or as a string. The data written to memory is treated as a sequence of bytes. Files have no structure in Perfect, and the language offers no means to provide it.

As a result of the weakness of the file handling mechanism in Perfect Developer, we were forced to abandon permanent storage of the database in our software. Attempts were made to construct our own File abstract data type, but the underlying mechanisms gave us very little power to make guarantees and ensure correctness of the software under construction.

6.1.4 Refinement

Two data refinements via transformation of the library were carried out in the LibraryStock class and the other in the LibraryCatalog class. Both are printed as part of Appendix G, but for brevity we will only present the latter here. Within the LibraryCatalog class, the catalog variable is refined into a Hashed Bucket implementation to support efficient retrieval of items. A Hashed Bucket, where the key is the authors name, provides efficient searching of the catalog by author name, but no less efficient on searches by subject. This
refinement is defined by:

```plaintext
internal
  var
    hashedLibraryItems: HashedBucket of LibraryItem;

  function allLibraryItems
    ^= hashedLibraryItems.ran;
```

This required the development of a generic `HashedBucket of X` class, the specification of which can be found in Appendix G.1. The development of the `HashedBucket` class required little effort because of the powerful specification language elements of Perfect. This produced an elegant and powerful generic class that could be integrated into our Library with ease. However, we note the need to overwrite the original specification to include the refinement as a negative aspect. The refinement code is scattered throughout the source code file making it difficult to parse as specification or implementation alone.

### 6.1.5 Wrapper Class

The Wrapper Class for this implementation was constructed in Java and is presented in full in Appendix H.7. This class creates a graphical user interface(GUI) that moderates the interactions between user and the Perfect implementation. We are required to read and understand the compiled Perfect code to properly develop this GUI.

Unfortunately, the compiled code is extremely difficult to parse as many of the functions and attributes have been mangled. For example the `isNumber` function of the `LibraryDB` class was defined as:

```plaintext
function isNumber(val:string):bool
  ^= forall i::0..<#val :- val[i].isDigit & #val>0;
```

but has been compiled to:
The function signature has changed to include a new variable `t0val` that is not referenced in the code, while the `string` type has become an `_eSeq` type. The `_eSeq` class is the Java class that provides the Perfect `seq` class functionality. Any `string` objects in Perfect will be considered `_eSeq` objects in Java as `string` is defined as `seq of nat`. The need for the `char` variable is to denote the type of the generic parameter of the sequence class. This is not documented in the Perfect Developer documentation, but is required to construct executable software.

We note that this could potentially cause an error in software if the developer interacts with the `_eSeq` object incorrectly. There exists no type checking of the elements in an `_eSeq` object. All elements are cast up to `_eAny` objects, Perfects
version of the Object type. This provides opportunities for nat objects to be inserted into strings by accessing the Java code, (See Appendix H). These difficulties hindered our implementation of software with Perfect Developer, but eventually an implementation was produced.

6.1.6 Animation

The Verifier struggled to animate the specification of the Library Management software. A few errors due to an incomplete specification were discovered and corrected, but the majority were left unresolved by the Verifier. These obligations arose out of trying to ensure the verification of the following invariant:

\[
\text{invariant } \forall \text{item1, item2} : \text{allLibraryItems} \rightarrow \text{item1} = \text{item2} \rightarrow \text{item1.getId} \neq \text{item2.getId};
\]

The major problem which the Verifier has is when an item object is borrowed or returned; the object is deleted and a new object with the same id is inserted in its place. This unusual method of changing state was necessary as Perfect Developer uses Value Semantics. The problem could not be removed from the verification by any means. Ultimately, Perfect Developer verified the software to 83%, failing to prove 35 out of 196 proof obligations. All of the failed obligations could be proven manually.

6.1.7 Refinement Verification

After Refinement occurred, the verification suffered enormously with only 63% of proof obligations being verified correct by Perfect Developer. We discovered that this failure was due to the weakness of the theorem prover at verifying refinements unless an exact replica of the specification was provided. For example, the following refinement could only be verified by altering the specification of the add schema in the HashedBucket class:
The specification of `add` had to be amended to mirror the specification of `append` in the Perfect Developer Reference Manual[78]. With this change in place, this specification could be verified.

However, other specifications could not be changed as easily. The complement to the above schema, the `removLibraryItem` schema, could not be verified even when an identical specification was provided. Through correspondence with Escher Technologies, it was discovered the specification provided in the Reference Manual was incomplete, with additional specification properties not included. Much work went into exploring Perfect Developers support of Refinement and the theorem prover was found to be very weak when refinement between two distinct abstract data types occurred. Further details of these refinement weaknesses may be found in our publication “The Perfect Developer tool for Software Construction through Refinement”[79].

6.1.8 Case Study 1 Results

We found Perfect Developer supported the development of the Library Manager but the technology behind the tool was lacking. With respect to representation and implementation support, Perfect was easy to use during development. However, constructing an executable piece of software was difficult. The need for wrapper classes makes life difficult for the developer. The lack of adequate file handling mechanisms limits the usefulness of Perfect for industrial software. The tools integration with UML is poor, requiring much revision by developers.
Most disappointing was the support for reasoning in Perfect Developer. The Verifier claims to be the strongest component of Perfect Developer, yet it is the most lacking in documentation and user-friendliness. The information provided when proofs failed was difficult to read, often relying upon sub-proofs which failed that contain variables not mentioned in the goal. These problems were evident in both verification of specification and implementation, but were most problematic after refinement when no supporting evidence is given for the reason failures occur.

6.2 Case Study 2: Resource Management Software

A Resource Manager is an operating system component that is in charge of allocation and de-allocation of resources to processes. Its job is essential in the operation of multi-threaded systems where many processes could be running simultaneously each making ongoing requests for resources to complete their jobs. Ideally, a resource manager should prevent deadlock or livelock, permitting all processes to complete in an efficient and timely manner.

The ability to formally verify complex resource managers is an enormous and it was the intention of this case study to experiment with Perfect Developers verification technology. Equally important to the discussion is how the tool deals with reactive software problems.

6.2.1 UML

The structure of the Resource Manager is presented in Figure 5. The System class takes on the role of the Operating System with respect to resource management. It is aware of all the Resources and their state in the system. It spawns processes, accepts their requests for resources and grants the resources on the judgment of the manager. The manager is the Resource Manager, and contains
the algorithms that determine whether or not a resource should be granted to a process. Control of the resources is controlled by the use of a Semaphore object.

As with the Library Manager, the UML class diagram was imported to Perfect Developer and edited as required. The full specification can be found in Appendix I and our discussions of the case study which follows will highlight the important elements.

6.2.2 Temporal Specification

Unfortunately, specification of the Resource Manager is almost impossible as Perfect contains no language for Temporal Specification[88]. It would be beneficial to be able to represent that eventually a process will acquire all it’s resources or that eventually all process get executed, but neither is supported. Attempts were made to include these kind of specifications by defining the notion of progress. Progress defines how close a process is to termination by measuring the number of resources it requires to execute and how soon it is believed these resources will be acquired. By defining progress on each process, a System specification that defines eventually execution could be encoded such that:

\[
\text{property}(i:\text{nat})
\]
assert (self after it!step(i)).progress < self.progress;

6.2.3 Safety Properties

The two essential properties of a resource manager are ensuring that deadlock and starvation never occur in the system. It was expected that these two properties could be represented in the software. Attempts were made that utilized the notion of progress presented in the previous section. Deadlock was defined as:

function hasDeadlock:bool
  ^=self after it!step.progress >= self.progress;

However, this proved to be of no use in verification was replaced by:

// if there are no resources, there is no deadlock otherwise
// deadlock occurs if the set of resources contains a cycle
function hasDeadlock:bool
  ^=([resources.empty]:
      false,
      []: exists r::resources :-
          hasCycle(set of from ResourceItem{},r,r.next.rep(1))
    );

function hasCycle(viewed:set of from ResourceItem,
                   at:from ResourceItem,
                   toBeViewed:bag of from ResourceItem):bool

pre viewed <<= resources,
    at in resources,
    toBeViewed.ran <<= resources
decrease #resources - #viewed
 ^= ([toBeViewed.empty]:
     false,
     [at in viewed]:
     true,
     []: (let newAt ^= toBeViewed.min;
           hasCycle(viewed.append(at),newAt,
                     toBeViewed.remove(newAt)++newAt.next.rep(1)))
The specification of starvation is impossible but could be approximated by including a `timeToStarvation` variable associated with each process. The System could then define an invariant, such that:

```
invariant forall p:ProcessItem :- p.timeToStarvation > 0;
```

Unfortunately, this approach is highly application dependent as the `timeToStarvation` value would differ from System to System and would require immense experimentation to discover. Ensuring this invariant would require the Resource Manager to check all processes each step and would be highly inefficient to guarantee. It was felt that this problem was beyond the capability of Perfect Developer to specify.

### 6.2.4 Parallel Execution

Perfect does not support multi-threaded applications yet this software is inherently multi-threaded. We therefore include an approximation of multi-threading by developing a `step` method that behaves like a system clock and performs some action determined by a number provided to the method. By assuming this number is randomized, a model of parallel execution can be thought to exist. The `step` schema is structured so that:

```plaintext
schema !step(choice:nat)
  pre ~systemEnd  // there is work to do.
  post (var p:ProcessItem,r:from ResourceItem;
     [choice=0]:
       If a process needs a resource,
       allow it to declare this need.
     ),
     [choice=1]:
       If a process is currently waiting on
```
a resource available, allocate it.
)
[]:
If a resource is currently held by
some process, release it.
)
);

If choice = 0, a random hungry process, called p, is selected from the set
of executing processes with needs. A random resource needed by this process,
called r, is selected and removed from the set of resources. The manager at-
ttempts to acquire resource r for the process p. The objects are then returned
to the system and execution continues. If the system has no hungry processes,
this step is skipped.

[choice=0]: // declare want of resource
([hasHungry]:
p!=pickHungry then
resources!=resources.remove(p) then
toBeExecuted!=toBeExecuted.remove(p) then
r!=p.getNeed then
resources!=resources.remove(r) then
manager.acquire(p!,r!) then
resources!=resources.append(p) then
toBeExecuted!=toBeExecuted.append(p) then
resources!=resources.append(r),
[]:
pass // System has allocated all it needs
// we don’t need to acquire any more
),

The other choices available in the step schema can be viewed in Appendix
I.7.

6.2.5 Value Semantics

As has been discussed in section 2.3.3, Perfect chose to employ value seman-
tics in the language. This choice has the effect of complicating implementation
of software. As can be seen in the previous section, code bloating occurs when variables are required to change state. During execution of the step schema, the values of the resource and process objects may change. These objects are cloned from elements of the resources and toBeExecuted sets, not references to the objects found there. For this reason, we must remove them from the sets, change them, and then insert them back into the sets. This not only is a cumbersome policy of data overriding but an inefficient one, requiring searching of sets to be made.

6.2.6 Non-Termination

It is likely that if this software to be constructed in the real world, it would be valid to state that it does not need to terminate. The Resource Manager could potentially run forever while there are processes being created that have needs. This notion of potential infinite execution cannot be modelled in Perfect Developer as it requires total correctness of verification. Total Correctness of a component demands the component guarantees its post-condition at termination and that it terminates at some point. It is enforced in Perfect Developer by defining a variant in all recursive functions and loops.

6.2.7 Animation

The verification of the Resource Manager case study was extremely difficult and a complete verification was found to be impossible. This was due to a number of factors including lack of elegant specifications and weaknesses of the Verifier to prove implications.

The specification language of Perfect Developer was not designed to specify temporal quantities, but an attempt was made to approximate it during this case study. These specifications are inelegant and presumably cause the Verifier
great difficulty in any attempt to manipulate them.

We discovered that Perfect Developer could not perform verification by implication by analysis of this case study. The tool opts for a brute force strategy that attempts all possible combinations in trying to dispatch a proof obligation. This approach is one of the worst for verifying resource managers as they permit huge numbers of potential situations to arise in even the smallest example sets. Various properties on small systems with a few processes and resources were included for verification, but met with failures to verify by Perfect Developer. This case study proved to show the weakness of the reasoning support of Perfect Developer.

6.2.8 Case Study 2 Results

The representation and implementation of a resource manager is a little beyond the capabilities of Perfect Developer. Perfect contains none of the elements of temporal specification or multi-threading making this task almost impossible. An approximation of the problem was developed, but it would be unfair to criticize Perfect Developer for not providing Reasoning support for this case study. After initial investigations into this case study, it was deemed unsuitable for further study. Perfect Developer provides no support for reactive systems and must be regarded as a tool for the development of single threaded software only.

6.3 Conclusions

In just two case studies, we have uncovered a number of limitations with Perfect Developer. These limitations are mainly a result of the poor documentation and control associated with the Verifier and its support of reasoning about software. Perfects support of representation of software was excellent,
being expressive enough to capture nearly all of what we wanted, though this expressivity does come at a cost of simplicity of language. The support for implementation is also limited with problems being met in most respects. The lack of decent file and exception handling mechanisms makes little sense. The implications of value semantics makes efficiency a huge concern for the developer. Overall, Perfect Developer appears to be good for small scale examples, but does not work equally well on larger software projects. What has been learned from these case studies will be expanded upon in Chapter 7 where we will provide a critical analysis of Perfect Developer.
7 Analysis of Perfect Developer

In this chapter we present our evaluation of the Perfect Developer support tool with respect to its support for Representation, Implementation and Reasoning. We selected Perfect Developer because the philosophy behind the tool best met our ideal of software development tools. Perfect presents an easy to learn language for specification and implementation of software. Perfect Developer supports the refinement of specifications into implementations. It provides a mechanism for fast prototyping of software to assist in validation while retaining good clean specification of software. The built-in Verifier can dispatch a high percentage of proof obligations with ease, simplifying and shortening the time spent during a testing phase. The tool could be integrated with Java and C++ projects increasing its appeal to industry. All these reasons made us extremely interested in this tool.

Through experimenting and analysing the tool in great depth, we found the tool did not quite live up to its own expectations. Perfect Developer contains such a great breadth of ideas, but unfortunately all of these ideas are limited in the tool. Perfect is lacking in many of the most powerful features in other specification languages to enable it represent so many styles of specification languages. The use of value semantics makes for correct code, but is too restricting for developers forcing developers to write inelegant code. The Verification technology is undocumented and lacks any user interaction. These results will be expanded on further in this chapter providing references to smaller examples or the case studies when relevant.
7.1 Representation Support

Perfect Developer supports the representation of software in part through *UML class diagrams*, but primarily through the specification elements of Perfect. The language supports a restricted range of *Higher Order Functions* that benefit specifications. *Refinement* provides a powerful mechanism to connect specifications to implementations but is lacking in power. The *Design by Contract* mechanism is well supported, but employs confusing terminology reducing the clarity of specifications.

7.1.1 UML support

Perfect Developer permits developers to initiate software development from UML class diagrams. This is achieved through an *import* feature of the tool. Importing a UML class diagram has the affect of creating a source file for every class named in the class diagram, each containing skeleton code for the attributes and methods listed in the class diagram (See Appendix XXX). This is useful to ensure the association between the structure design and the specification is retained. However, the UML importer is extremely limited at the time of writing. It supports only the most basic aspects of the UML and lacks the ability to reflect changes made to the implementation in the UML.

The poor clarity of skeleton code generated by the UML importer is evident from the example given in Appendix D. The types of all variables are declared as *from X*, where X is the UML type. This overcomes the difference in the treatment of polymorphism in UML and Perfect. Generally, *from X* is not the required type, but there is no way to set the UML to record X as the type. Source files of the primitive types are often unnecessarily created if they are referred to in the code and must be removed manually from the project.
The main disadvantage to the importer is that it provides no support to OCL specifications in the class diagrams, even though this support could be easily enabled with Perfect’s support of DbC.

7.1.2 Higher Order Functions

We have already discussed the fact that Perfect contains some built in higher order functions taken from the function programming domain, but the real failing of Perfect is that this is all it provides. In functional programming, software engineers are given the freedom to construct their own higher order functions that can be applied to their own data structures. By giving this freedom to developers, their software can be increasingly generic and re-use is strongly encouraged. With Perfect, the developers toolkit is restricted to the common higher order functions. Even these are restricted by only operating on the domain of the built in data types.

Functional languages have long known of the benefits of providing developers with the tools to construct their own higher order functions. Projects such as Pizza[89] and GJ[90] have worked general purpose higher order functions into the Java language. By including this feature, Java programs can be re-used effectively and the modular notion of classes and data structures is freed to a great extent. Perfect has instead opted to give a few higher order functions, which while being an advantage over most currently used OO languages, is not nearly enough. The language of higher order functions in Perfect is restricted to the domain of the specification.

We attempted to develop a mechanism that would allow a richer specification of higher order functions in Perfect Developer(See Appendix J). The language of generics proved lacking to fully incorporate all the features of functional
programming that we would like.

7.1.3 Refinement

Refinement is the great idea of Perfect, placing the language apart as a simple, robust language that supports formal processes. The feature is prized and works well, but in the Library Database case study (Section 6.1), we found it too is restricted. Source code that undergoes refinement suffered extensively from bloating. When refinement occurred, it is intermixed with the specification. If mining source code, developers will suffer trying to separate their specifications from implementations. Further limiting refinement is the requirement that all refinements are big step refinements. A developer must leap from specification to implementation in one step, regardless of the difficulties that this may generate.

The most closely related language to Perfect with respect to refinement is the B-tool’s AMN[17]. The AMN supports retrieve relations to define a refinement step. Multiple refinement steps may be made, each transforming the specification into a more concrete and implementable specification. Finally, the specification is refined into an implementation. This solves two of our grievances with Perfect. The third is easily solved by placing each specification/implementation in separate files, something also supported by the AMN. It does appear that in trying to re-invent refinement for object oriented software, Perfect has chosen not to adhere to a useful and powerful language technique already available within the B-tool.

The model of refinement as an integral language element is novel and Perfect must be praised for it’s introduction but these limitations, especially ones that can be easily overcome, begs questions about how much foresight was put into
the Perfect language at inception. The concepts can all be found in the language, but their use is overly complex.

7.1.4 Contracts

Perfect does provide developers with a rich assertion language permitting complex design by contract specifications to be encoded. This language goes far in advance of Eiffel[32], the first OO programming language built to include design by contract, and is on a par with JML[34]. However this language is encumbered with confusing syntax and stylistic choices that make it difficult to parse by humans. The prime notation for distinguishing before and after values of variables can easily be confused making understanding the meaning of a value difficult. This was evident in

JML is one of the most advanced Design by Contract specification languages taking lightweight specification as its main goal. The language uses meaningful and well-understood keywords to clearly point out the start and end of a contract. Not only this, but the contract is isolated within the code, separating specification and implementation in a manner that improves parsing of specifications. This differs from Perfect who uses post-assertions merely as assert statements tagged on at the end of an implementation. Another useful feature is the keyword \texttt{old} to denote old values of variables. While this causes some overhead in typing, the clarity of semantics is immense. Rather than opting for a minimal solution, Perfect could easily have included simple and readable notation to improve human parsing of specifications.

7.1.5 Results

Perfect supports a range of specification styles but has no core style of its own. This does allow many different kinds of developers to approach the tool,
but it makes learning to specify in Perfect difficult. The language lacks the completeness of a single theory, never giving the developer all the support higher order functions or design by contract could provide. It opts to merge all these competing ideas together so that there is always a way of specifying what one wants but requiring much effort to be carried out to find this way. The language lacks many of the niceties of existing languages of similar nature as it attempts to provide a little flavour of everything. Perfect supports representation in too many ways, which in turn makes the support difficult to understand at points.

7.2 Implementation Support

The concepts that Perfect supports have made it less than the ideal choice for implementing software. The language contains Terminology that can confuse developers attempting to read and maintain Perfect Code. There is no support of Exception or file handling mechanisms, two essentials of programming languages. The language does not compile into machine code directly requiring a difficult Translation into a high level language. While this should not upset development, it adds to the developers difficulty in writing Wrapper Classes that provide graphical user interface, something not supported at all in Perfect. This form of support greatly reduces the benefits of employing Perfect to ensure correctness of code through static checking. The greatest limitation of Perfect as an implementation language is the choice of Value Semantics which gives rise to object cloning of the highest order in most software. We now discuss each of these drawbacks and how they hinder implementation in Perfect.

7.2.1 Terminology

The term schema has its origins in Z and is used to describe the state of the system. This is a suitable term for the model oriented specification style of
Z, but in object oriented languages it is misleading. The schema in Perfect is a method that either changes the state of the object to which it refers or the state of one of the parameters of the method.

Perfect Developer has an alternate terminology for the traditional concept of *post-condition*. Instead of defining a post-condition to be the guarantee of a contract, Perfect re-terms this a post-assertion. To make matters worse, a post-condition is given a new meaning for use in defining the change of state after execution of a schema.

The treatment of inheritance is also confusing as Perfect enforces *Contravariance of Methods*[91] in the inheritance hierarchy but also permits covariance through functional pre-conditions (See Appendices J and K). Covariance is also included in the form of descendant hiding, permitted by Perfect Developer. Unlike other languages that choose a single theoretical grounding, Perfect Developer attempts to offer all.

### 7.2.2 File Handling

Most programming languages provide a simple and intuitive file handling mechanism. File Handling consist of providing an abstract data type of the file structure, providing an interface to the operating system, and handling errors when reading or writing to memory. These are tasks that Perfect is ideally suited for but the default mechanism is primitive. All messages associated with file handling are passed through an *Environment* object. This is not a global object and must be passed as a parameter to any object wishing to perform file handling operations. However, in our exploration with the Library Database Case Study(Section 6.1) we found Perfect Developer to be very lacking in its support of File Handling.
The Environment object contains a few primitive schema that control file handling. Each schema has associated with it an out variable that provides some form of result. These result types are not uniform across the range of schema in the system and may be united with other types. Post-conditions are written in terms of these result types leading to complex post-assertions. The File data structure is defined as a seq of byte. Reading and writing to a file will require significant casting. Even if the developer reads the data from a file correctly, there are not guarantees that the data will be used correctly.

7.2.3 Exception Handling

Exception handling is a programming language mechanism designed to handle runtime errors that are unforeseeable. The exception handling mechanism increases robustness of software systems with respect to events outside the systems knowledge. Perfect contains no exception handling language. Developers may use the design by contract strategy to generate software exceptions by including tests on the conditions of the contract. These checks will be performed at runtime to guarantee the contract is fulfilled. If a failure is encountered, the program should be immediately halted and the error printed to the screen. This mechanism sometimes fail to terminate the program, putting the system in an inconsistent state.

7.2.4 Translation

Perfect cannot be compiled straight to executable machine or byte code but into other high level programming languages like Java and C++. The compiled code is usually unreadable and can contain different method signatures, different attribute names and even the creation of new attributes that serve no significant purpose in the program. Ideally, no developer should have to work with the
compiled code but the efficiency of such translations come into question. This efficiency cannot be tested effectively as much of it relies upon pre-compiled code provided by the Perfect libraries.

### 7.2.5 Wrapper Classes

A major inconvenience to generating complete software systems with Perfect Developer is the lack of a graphics library. Perfect contains no GUI generation tools and only has an ad hoc command line system that can be used. In order to create any form of graphical interface, the developer is required to know the language the code is being compiled to and create the interface with that. This involves reading and understanding the compiled code and interacting directly with it. To simplify the task, it is suggested developers construct wrapper classes that operate as a barrier for passing information in and out of the system.

### 7.2.6 Value Semantics

One of the most concerning inefficiencies found in code compiled from Perfect is the choice of value semantics as default. Using value semantics causes inefficiencies in both space and time complexity which can grow exponentially as the software size increases. To support the Perfect assertion language, whenever an objects value changes, a cloned object must be made in case the original value is referred to elsewhere. The clone must be deep cloned to ensure no two references point to the same object. As objects become larger, consisting of nested sub-objects, this cloning could become highly costly in space, especially for an object that is often discarded after the operations lifetime. The other big cost comes when comparing two objects for equality which requires a check for equality on all the sub-objects. This operation is often performed when dealing with sets, bags or maps which may be quite large collections.
7.2.7 Results

The only support that Perfect Developer provides the Implementation of software is given by its static checking of methods and variable typing. This support, while excellent, can be provided by any of a number of tools. The tool fails to provide many of the facilities that are available to other programming languages. There is no graphics library which forces developers to read the compiled source code and developer wrapper classes. There is no file or exception handling mechanisms other than the most basic of interfaces. The implications that Value Semantics has on the execution of software is great, requiring massive cloning in certain cases. Perfect Developer must be noted as providing poor support for the implementation of efficient executable software.

7.3 Reasoning Support

The centre of Perfect Developer is the reasoning engine provided by the Verifier. It sets Perfect Developer apart as a useful tool for developers without much mathematical knowledge but wanting formally verified software. While Verification is able to prove a large percentage of common proof obligations, the technology itself often seems basic and lacking. Using the Verifier is simple and straightforward, but consumes much processor time a lot of which can be wasted time when global obligations that have been proved are proved again and again owing to local changes. When errors are reported, the challenge of finding them and correcting them is made increasingly difficult by the choice of an automated technology. The Verifier does what it does well, but when it fails, this failures offer huge challenges and we discuss them below.
7.3.1 Verification Technology

7.3.2 The Verification Process

One of the weakest points of Perfect Developer is the interaction between developer and the Verifier. While the idea of an automated theorem prover appears to be sound, offering developers with limited mathematical knowledge the ability to verify software, one has to question who these developers are. If a developer does not understand the mathematical underpinnings of the theorem prover, what does it mean to them for software to be correct? Automating the operations of the theorem prover only limits what developers with good mathematical knowledge can do. Other tools such as KeY\cite{68} and B\cite{52} offer semi-interactive theorem proving. This offers the developer the option to have proof obligations dispatched immediately if possible, but if it cannot find a proof on its own, the developers assistance is sought. These tools also offer the ability to construct proof tactics or strategies that can be added to the theorem provers knowledge base to be used in the future.

Another disadvantage to Perfect Developer is the requirement that all proof obligations within a source file be re-verified each time the verifier is employed. If there are a large number of proof obligations, and one is causing problems, every local change to the source code will require a complete re-verification. While this is necessary to prevent global changes from being missed, techniques such as proof-carrying code\cite{92} would allow the proof strategy to be re-used making this search efficient.

7.3.3 Error Reporting

As an automated theorem prover provides the backbone to the Verifier, the requirement for it to produced meaningful and understandable output is
essential. Instead, as has been illustrated, the Verifier can only produce useful, annotated reports when a proof is successful. When the Verifier fails to find any correct path it usually states the failure and introduces the last step it tried to verify. This last step can contain new variables whose meaning is unclear from the context. There exists techniques for finding the source of these errors, but this often requires altering the source code and re-verifying it. Such an effort is discouraging to developers as little or no progress is made between iterations.

7.4 Conclusions

We have provided an analysis of Perfect Developer with respect to its support of Software Representation, Implementation and Reasoning. It has been shown to be a good mechanism for representing software, but one that is lacking in clarity and sophistication. It provides too many possible mechanisms of representation, but lacks the power and sophistication of any. It’s support of implementing software is terrible, lacking many of the features that are found in the most common programming languages. It supports reasoning extremely well when it can prove software correct, but extremely poorly when it cannot. The Verifier is undocumented and not interactive. This leaves developers who wish to experiment with the tool high and dry. Perfect Developer represents a good first step in creating a software development tool, but it has a long way to go before being perfect.
8 Recommendations for Tool Support

In this chapter we present our recommendations for a tool that could assist software development based on our experiences with the current tool support technologies. We present our recommendations under the three forms of support used throughout this thesis: Support of Representation; Implementation; and Reasoning. The techniques are based on current cutting edge research and have been employed on smaller scales. We believe that by combining these techniques together within a single yet compact software tool, the software development process could be improved and software quality increased.

8.1 Representation Support

We recommend an improved support for the representation of software by a tool that supports a greater scope of representation. This tool should provide some mechanism of Requirements Exploration to better document the development ensuring no requirement is lost. It should also permit specification of software on a larger granularity supporting Kinds[] and Design Patterns[] to be used during the development of specifications. Software changes could be better represented by including elements of the Aspect Oriented Programming paradigm. These ways will be further outlined, providing a guide to how a tool could better support representation of software.
8.1.1 Requirements Exploration

8.1.2 Kinds

8.1.3 Design Patterns

8.1.4 Aspect Oriented Programming

8.2 Implementation Support

We believe that languages like Perfect need to be enriched in several areas to better support the implementation of software. Increasingly, software relies upon External Components and some support of this needs to be catered for by a tool. The implementation language must have expressive mechanisms to describe Graphics Libraries and Database Interactions to support modern software practices. A useful File and Exception Handling mechanism must exist that is both efficient and correctly specified. Software could be constructed that is large, powerful and efficient by supporting these implementation details.

8.2.1 File and Exception Handling

8.2.2 Database Interactions

8.2.3 Graphics Libraries

8.2.4 External Components

8.3 Reasoning Support

It has been clear that Perfect Developer provided excellent reasoning support when proofs could be generated but poor support when they could not be. We believe that a tool that employed a well documented and standard language for describing these proofs would be of great benefit. This language would not hide the details of theorem proving, but rather promote an easier form of Interactive Theorem Proving in the cases that it was advantageous to do so. Software correctness could be ensure by using Proof Carrying Code giving
rise to a greater ability to re-use external components. Proof Carrying Code could also assist in the recognition of proof patterns in software, improving the automation capabilities of the tool through the use of \textit{Tactlets}.

\section*{8.3.1 Accepted Standard Logic}

\section*{8.3.2 Interactive Theorem Proving}

\section*{8.3.3 Proof Carrying Code}

\section*{8.3.4 Tactlets}

\section*{8.4 Conclusion}

In this section we have presented our recommendations for a tool that better supports software development. This tool would provide stronger support for representation of software, encompassing the phase of requirements acquisition and remaining a factor throughout the development life-cycle. Support of implementation features would rely on better languages for handling external commodities like graphics libraries, databases and third-party components. The tool should be built with a strong reasoning mechanism that uses the developer as a key element of the reasoning mechanism.
9 Conclusions

An overview of the object oriented software development process has been presented, illustrating the strengths and weaknesses of such a tact. Computer tools that support this process were reviewed and their role within the development process analysed. In particular depth we looked at Perfect Developer, a tool that while limited, contains many of the ideas at the heart of an ideal support tool. This overview of the technologies and tools at the cutting edge served as a stepping stone to generate recommendations for an ideal tool that better meets the ideal.
List of References


[75] G. Salzer, “Perfect developer @ tu wien.” http://www.logic.at/perfect/.


[89] M. Odersky and P. Wadler, “Pizza into Java: Translating theory into prac-

[90] G. Bracha, “Generics in the java programming language.”

ACM Transactions on Programming Languages and Systems, vol. 17,

[92] G. Necula, “Proof-Carrying Code: design, implementation and applica-
tions,” in Proc. of international conference on Principles and practice of
A Harbour XMI File
Here is the Harbour.xmi file generated by ARGO UML.

<?xml version="1.0" encoding="UTF-8"?>
<XMI xmi.version="1.0">
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>Novosoft UML Library</XMI.exporter>
      <XMI.exporterVersion>0.4.19</XMI.exporterVersion>
    </XMI.documentation>
  </XMI.header>
  <XMI.metamodel xmi.name="UML" xmi.version="1.3"/>
</XMI.content>

<Model_Management.Model xmi.id="xmi.1"
  xmi.uuid="-107--99--10-30-6b473a1e:1035517eeb7:-8000">
  <Foundation.Core.ModelElement.name>untitledModel</Foundation.Core.ModelElement.name>
  <Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
  <Foundation.Core.GeneralizableElement.isRoot xmi.value="false"/>
  <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false"/>
  <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false"/>
  <Foundation.Core.Namespace.ownedElement>
    <Foundation.Core.Class xmi.id="xmi.2"
      xmi.uuid="-107--99--10-30-6b473a1e:1035517eeb7:-7ff9">
      <Foundation.Core.ModelElement.name>Port</Foundation.Core.ModelElement.name>
      <Foundation.Core.ModelElement.visibility xmi.value="public"/>
      <Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
      <Foundation.Core.GeneralizableElement.isRoot xmi.value="false"/>
      <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false"/>
      <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false"/>
      <Foundation.Core.Class.isActive xmi.value="false"/>
      <Foundation.Core.ModelElement.namespace>
        <Foundation.Core.Namespace xmi.idref="xmi.1"/>
      </Foundation.Core.ModelElement.namespace>
    </Foundation.Core.Class>
    <Foundation.Core.Classifier.feature>
      <Foundation.Core.Attribute xmi.id="xmi.3"
        xmi.uuid="-107--99--10-30-6b473a1e:1035517eeb7:-7fa5">
        <Foundation.Core.ModelElement.name>id</Foundation.Core.ModelElement.name>
        <Foundation.Core.ModelElement.visibility xmi.value="public"/>
        <Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
        <Foundation.Core.Feature.ownerScope xmi.value="instance"/>
        <Foundation.Core.StructuralFeature.multiplicity>
          <Foundation.Data_Types.Multiplicity_range xmi.id="xmi.4">
            <Foundation.Data_Types.MultiplicityRange xmi.id="xmi.5">
              <Foundation.Data_Types.MultiplicityRange.lower>1</Foundation.Data_Types.MultiplicityRange.lower>
            </Foundation.Data_Types.MultiplicityRange_range>
          </Foundation.Data_Types.MultiplicityRangeRange>
        </Foundation.Core.StructuralFeature.multiplicity>
      </Foundation.Core.Attribute>
    </Foundation.Core.Classifier.feature>
  </Foundation.Core.Namespace.ownedElement>
</Model_Management.Model.xmi>
<Foundation.Data_Types.MultiplicityRange.lower>1</Foundation.Data_Types.MultiplicityRange.lower>
</Foundation.Data_Types.MultiplicityRange>
</Foundation.Core.AssociationEnd.multiplicity>
<Foundation.Core.AssociationEnd.changeability xmi.value="changeable"/>
<Foundation.Core.AssociationEnd.association>
<Foundation.Core.Association xmi.idref="xmi.45"/>
</Foundation.Core.AssociationEnd.association>
<Foundation.Core.AssociationEnd.type>
<Foundation.Core.Classifier xmi.idref="xmi.2"/>
</Foundation.Core.AssociationEnd.type>
</Foundation.Core.AssociationEnd>
</Foundation.Core.Association.connection>
</Foundation.Core.Association>
<Foundation.Core.AssociationEnd xmi.id="xmi.50" xmi.uuid="-107--99--10-30-6b473ae1:1035517eeb7--7fad">
<Foundation.Core.ModelElement.name>size</Foundation.Core.ModelElement.name>
<Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
<Foundation.Core.GeneralizableElement.isRoot xmi.value="false"/>
<Foundation.Core.GeneralizableElement.isLeaf xmi.value="false"/>
<Foundation.Core.GeneralizableElement.isAbstract xmi.value="false"/>
<Foundation.Core.ModelElement.namespace>
<Foundation.Core.Namespace xmi.idref="xmi.1"/>
</Foundation.Core.ModelElement.namespace>
<Foundation.Core.AssociationEnd.connection>
<Foundation.Core.AssociationEnd xmi.id="xmi.51" xmi.uuid="-107--99--10-30-6b473ae1:1035517eeb7--7fac">
<Foundation.Core.ModelElement.visibility xmi.value="public"/>
<Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
<Foundation.Core.AssociationEnd.isNavigable xmi.value="true"/>
<Foundation.Core.AssociationEnd.ordering xmi.value="unordered"/>
<Foundation.Core.AssociationEnd.aggregation xmi.value="composite"/>
<Foundation.Core.AssociationEnd.targetScope xmi.value="instance"/>
<Foundation.Core.AssociationEnd.multiplicity>
<Foundation.Data_Types.Multiplicity xmi.idref="xmi.4"/>
</Foundation.Core.AssociationEnd.multiplicity>
<Foundation.Core.AssociationEnd.changeability xmi.value="changeable"/>
<Foundation.Core.AssociationEnd.association>
<Foundation.Core.Association xmi.idref="xmi.50"/>
</Foundation.Core.AssociationEnd.association>
<Foundation.Core.AssociationEnd.type>
<Foundation.Core.Classifier xmi.idref="xmi.25"/>
</Foundation.Core.AssociationEnd.type>
</Foundation.Core.AssociationEnd>
<Foundation.Core.AssociationEnd xmi.id="xmi.52" xmi.uuid="-107--99--10-30-6b473ae1:1035517eeb7--7fab">
<Foundation.Core.ModelElement.visibility xmi.value="public"/>
<Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
<Foundation.Core.AssociationEnd.isNavigable xmi.value="true"/>
<Foundation.Core.AssociationEnd.ordering xmi.value="unordered"/>
<Foundation.Core.AssociationEnd.aggregation xmi.value="none"/>
<Foundation.Core.AssociationEnd.targetScope xmi.value="instance"/>
<Foundation.Core.AssociationEnd.multiplicity>
<Foundation.Data_Types.Multiplicity xmi.idref="xmi.4"/>
</Foundation.Core.AssociationEnd.multiplicity>
<Foundation.Core.AssociationEnd.changeability xmi.value="changeable"/>
<Foundation.Core.AssociationEnd.association>
<Foundation.Core.Association xmi.idref="xmi.50"/>
</Foundation.Core.AssociationEnd.association>
<Foundation.Core.AssociationEnd.type>
<Foundation.Core.Classifier xmi.idref="xmi.14"/>
B Harbour Skeleton Code

Here is the skeleton source code of the Harbour example generated by the UML importer.

B.1 Harbour.pd
class Harbour =
abstract
var myShip: set of ref from Ship;
var myPort: those x: set of ref from Port := # x >= 1;
interface
schema !freePort post ?;
schema !arrive (s: from Ship) post ?;
schema !depart (s: from Ship) post ?;
selector myShip;
selector myPort;
end;

B.2 Port.pd
class Port =
abstract
var id: int;
var myShip: ref from Ship || void;
var myHarbour: from Harbour;
interface
selector id;
function free: from bool ^= ?;
function dockedHere: from Ship ^= ?;
function Dock (Ship: void): from Port ^= ?;
function Clear: from Port ^= ?;
selector myShip;
selector myHarbour;
end;

B.3 Ship.pd
class Ship =
abstract
var id: from string;
var myPort: from Port;
var myHarbour: from Harbour;
var myShipSize: from ShipSize;
interface
selector id;
selector myPort;
selector myHarbour;
selector myShipSize;
end;

B.4 ShipSize.pd
class ShipSize =
abstract
var myShip: from Ship;
interface
selector myShip;
end;

B.5 string.pd
class string =
interface
end;
C Harbour Specification Code

Here is the specification of the Harbour example.

C.1 FastShip.pd

class FastShip inherits Ship ==

interface

function maxSpeed:nat == max
assert result > 50;

C.2 Harbour.pd

class Harbour ==

abstract

var ports:set of Port, // all ports available
    incoming:Queue of Ship; // incoming queue of ships

invariant #ports > 0; // a harbour has ports

function freePorts:set of Port // the ports available
  == those p::ports :- p.free;
function haveFreePorts:bool // are there free ports?
  == #freePorts > 0;
function hasQueued:bool // are there ships queued?
  == ~incoming.empty;

// if we have free ports, we don’t have ships queueing
invariant haveFreePorts == > ~hasQueued;

confined // Private methods

// a ship lands at the given port
schema !land(s:Ship,b:Port)
  pre b in freePorts // the port is free to begin
  post ports!=ports.remove(b) then // no longer free
      ports!=ports.append(b.Dock(s)); // ship docked

// a queued ship lands at the freed port
schema !landQ(p:Port)
  pre haveFreePorts // we have a free port
  post (let s^=incoming.head; // first ship on the queue
      incoming!=incoming.tail then // remove him
    !land(s,p) // land the ship
  );

interface // Class methods

build{n:nat} // Constructor
  pre n>0 // there will be positive number of ports
  post ports!=(for i::1..n yield Port{i}).ran, // make ports
    incoming!=seq of Ship{}; // no ships queueing

// output
redefine function toString:string
  == "Incoming : " ++ incoming.toString ++ " & ports :" ++ ports.toString;

// from Abstract section
function freePorts;
function haveFreePorts;
function hasQueued;

// Get the least used port (lowest id)
function freePort:Port
  == freePorts.min;
// is s docked in our harbour?
function isDocked(s:Ship):bool
  ^= (let shipsDocked= for b::ports yield b.dockedHere;
    s in shipsDocked
  );

// where is s docked in our harbour?
function myPort(s:Ship):Port
  pre isDocked(s)
  ^= that b::ports :- b.dockedHere=s;

// a ship arrives, land the ship or queue her
schema !arrive(s:Ship)
  post ([haveFreePorts]:!land(s,freePort), // if free ports, land...
    []:incoming!=incoming.append(s) // ...else add to queue
  );

// a ship leaves, free Port and check queue for new ships
schema !depart(s:Ship)
  pre isDocked(s)
  post (let dockedPort=myPort(s);
    let emptyPort=dockedPort.Clear;
    ports!=ports.remove(dockedPort) then //no longer docked
    ports!=ports.append(emptyPort) then //now empty
    ([hasQueued]:!landQ(emptyPort), // land queued ships or..
      []:pass // ..do nothing
    )
  );

property(s:Ship)
pre hasVacantPorts
assert (self after it!arrive(s)).isDocked(s);
end;

C.3 Main.pd
schema main(context!: limited Environment, args: seq of string,
  ret!: out int)
  pre #args > 0
  post (var h:Harbour!=Harbour{1}, // a harbour of size 1
    s1:Ship!=Ship{"1"}, // some ships
    s2:Ship!=Ship{"2"},
    s3:Ship!=Ship{"3"};
    h!arrive(s1) then
      context!print("S1 arrives\n") then
      context!print(h.toString++"\n") then

    h!arrive(s2) then
      context!print("S2 arrives\n") then
      context!print(h.toString++"\n") then

    h!arrive(s3) then
      context!print("S3 arrives\n") then
      context!print(h.toString++"\n") then

    // Departure of ships from the harbour
    // Includes landing as ships leave in sequence

    h!depart(s1) then
      context!print("S1 departs\n") then
      context!print(h.toString++"\n") then
h!depart(s2) then
  context!print("S2 departs\n") then
  context!print(h.toString++"\n") then

h!depart(s3) then
  context!print("S3 departs\n") then
  context!print(h.toString++"\n") then

// End of test

context!print("Harbour empty")
)then
ret! = 0;

// End

C.4 Port.pd

class Port =~
abstract // Specification of model
var id:nat, // Port id
docked:Ship||void, // a docked Ship or nothing
used:nat; // used so many times
confined // Private methods

// constructor which initializes all attributes
// !!!not accessible globally!!!
build(!id:nat,!docked:Ship||void,!used:nat);

interface // Class methods

// global constructor. All ports are created free and never used
build(i:nat)
  ^= Port{i,null,0};

// ordering of Ports
total operator ~~(arg)
  ^= ([(used=arg.used]: id~~arg.id, // if used same amount of times, ids
       []): used~~arg.used // otherwise how many times used
);

// output
redefine function toString:string
  ^= ([free]:"Port" ++ id.toString ++ " is docking " ++ (docked is Ship).toString,
      []): "Id:" ++ id.toString ++ " is free"
);

// get method
function id;

// is the Port free?
function free:bool
  ^= docked = null;

// What is docked here?
function dockedHere:Ship||void
  ^= docked;

// Empty this Port, (new object for Value semantics)
function Clear:Port
  ^=Port{id,null,used+1};

// Dock at this Port, (new object for Value Semantics)
function Dock(s:Ship):Port
  ^=Port{id,s,used};
end;
C.5  Queue.pd
class Queue of X :=
abstract
  var queue:seq of X;
end;

C.6  Ship.pd
class Ship :=
abstract
  var myName:string;
  var size:ShipSize;
interface            // Class methods
// constructor
build(!id:string);
// get Method
function id;
// output
  redefine function toString:string
    ^= "Ship - " ++ id;
end;

C.7  SmallShips.pd
class SmallShips := those s:Ship := s.size <= Barge@ShipSize;

C.8  ShipSize.pd
class ShipSize :=
enum
  Dinghy,
  Yacht,
  Barge,
  Ferry
end;

C.9  SuperFastShip.pd
class SuperFastShip inherits FastShip :=
interface
  redefine function maxSpeed:nat
    ^= max
  assert result > 100;
D Harbour Refinement Code

Here is the refinement of the Harbour example.

D.1 Harbour.pd

```plaintext
class Harbour ^=
  abstract // Specification of model
  var ports:set of Port, // all ports available
      incoming:seq of Ship; // incoming queue of ships
  invariant #ports >0; // a harbour has ports

  function freePorts:set of Port // the ports available
      ^= those p::ports :- p.free
        via value openPorts // refined
        end;

  function haveFreePorts:bool // are there free ports?
      ^= #freePorts >0
        via value #openPorts>0 // refined
        end;

  function hasQueued:bool // are there ships queued?
      ^= ~incoming.empty
        via value ~incoming.empty // refined
        end;

    // if we have free ports, we don’t have ships queueing
    invariant haveFreePorts ==> ~hasQueued;

  internal // Data Refinement
    var openPorts:set of Port, // free ports
        closedPorts:set of Port; // ports in use
    function ports // retrieve function
        ^= openPorts++closedPorts;

    // No port is free and in use
    invariant openPorts++closedPorts=set of Port();

    // open ports are free & closed ports are not
    invariant forall p::openPorts :- p.free;
    invariant forall p::closedPorts :- ~p.free;

closed // Private methods

    // a ship lands at the given port
    schema !land(s:Ship,b:Port)
      pre b in freePorts // the port is free to begin
      post ports!=ports.remove(b) then // no longer free
        ports!=ports.append(b.Dock(s)) // ship docked
        via
          openPorts!=openPorts.remove(b), // refined
          closedPorts!=closedPorts.append(b.Dock(s))
        end;

    // a queued ship lands at the freed port
    schema !landQ(p:Port)
      pre haveFreePorts // we have a free port
      post (let s^=incoming.head; // first ship on the queue
            incoming!=incoming.tail then // remove him
              !landQ(s,p) // land the ship
            )
        via (let s^=incoming.head; // refined
            incoming!=incoming.tail then
              !landQ(s,p)
            )
      end;

  interface // Class methods
```

119
operator = (arg);  // Refined =

build{ n:nat }  // Constructor
pre n>0  // there will be positive number of ports
post ports!=(for i::1..n yield Port{i}).ran,  // make ports
      incoming!=seq of Ship{}  // no ships queueing
via  // refined
      openPorts!=(for i::1..n yield Port{i}).ran,  // all ports free
      closedPorts!=set of Port{},  // no ports closed
      incoming!=seq of Ship{}  // no
end;

// output
 redefine function toString: string
   ^= "Incoming : " ++ incoming.toString ++ " & ports : " ++ ports.toString
via value "Incoming : " ++ incoming.toString ++ " & ports :
   ++ (openPorts++closedPorts).toString
end;

// from Abstract section
 function freePorts;
 function haveFreePorts;
 function hasQueued;

// Get the least used port (lowest id)
 function freePort: Port
   ^= freePorts.min
   via value openPorts.min
end;

// is s docked in our harbour?
 function isDocked(s: Ship): bool
   ^= (let shipsDocked ^= for b::ports yield b.dockedHere;
      s in shipsDocked)
   via let shipsDocked ^= for b::closedPorts yield b.dockedHere;
   value s in shipsDocked;
end;

// where is s docked in our harbour?
 function myPort(s: Ship): Port
 pre isDocked(s)
   ^= that b::ports :- b.dockedHere=s
   via value that b::closedPorts :- b.dockedHere=s // refined
end;

// a ship arrives, land the ship or queue her
 schema ! arrive(s: Ship)
 post ([haveFreePorts]!land(s, freePort),  // if free ports, land...
      []: incoming!= incoming.append(s)  // ...else add to queue
   )
 via ([haveFreePorts]!land(s, freePort),  // refined
      []: incoming!= incoming.append(s)
   )
end;

// a ship leaves, free Port and check queue for new ships
 schema ! depart(s: Ship)
 pre isDocked(s)
 post (let dockedPort ^= myPort(s);
      let emptyPort ^= dockedPort.Clear;
      ports!=ports.remove(dockedPort) then  //no longer docked
      ports!=ports.append(emptyPort) then  //now empty
      ([hasQueued]: ! landQ(emptyPort),  // land queued ships or..
      []: pass  //...do nothing
via let dockedPort = myPort(s); // refined
let emptyPort = dockedPort.Clear;
closedPorts = closedPorts.remove(dockedPort);
openPorts = openPorts.append(emptyPort);
if [hasQueued]: !landQ(emptyPort);
[]: pass;
fi
end;
end;
E Library Skeleton Code

Here is the skeleton source code for the Library Database generated by the UML importer.

E.1 Author.pd
class Author ^=
abstract
var name_: from string;
var myBookDescription: from BookDescription;
interface
selector name_; 
function getName: from string ^= ?;
selector myBookDescription;
end;

E.2 BookDescription.pd
class BookDescription ^=
abstract
var isbn: from nat;
var title: from string;
var myAuthor: from Author;
var mySubject: from Subject;
interface
selector isbn; 
selector title; 
function getIsbn: from nat ^= ?;
selector myAuthor;
selector mySubject;
end;

E.3 BookInfo.pd
class BookInfo ^=
abstract
var myLibraryCatalog: from LibraryCatalog;
var myBorrowing: from Borrowing;
var myLibraryDB: from LibraryDB;
interface
function findBookByAuthor: from LibraryBookDescription ^= ?;
function findBookBySubject: from LibraryBookDescription ^= ?;
schema !borrowItem (copy: from LibraryItem, bor: from Borrower, date: from string) post ?;
schema !returnItem (copy: from LibraryItem) post ?;
selector myLibraryCatalog;
selector myBorrowing;
selector myLibraryDB;
end;

E.4 BorrowerBase.pd
class BorrowerBase ^=
abstract
var myUserBase: from UserBase;
var myBorrower: from Borrower;
interface
schema !addBorrower post ?;
selector myUserBase;
selector myBorrower;
end;

E.5 Borrower.pd
class Borrower ^= inherits Person
abstract
var borrowId: from nat;
var limit: from nat;
var myBorrowing: from Borrowing;
var myBorrowerBase: from BorrowerBase;
interface
selector borrowerId;
selector limit;
schema !increaseLimit (inc: from nat) post ?;
selector myBorrowing;
selector myBorrowerBase;
end;

E.6 Borrowing.pd
class Borrowing ^=
abstract
var dateBorrowed: from string;
var myBookInfo: from BookInfo;
var myLibraryItem: from LibraryItem;
var myBorrower: from Borrower;
interface
selector dateBorrowed;
selector myBookInfo;
selector myLibraryItem;
selector myBorrower;
end;

E.7 LibraryBook.pd
class LibraryBook ^=
interface
end;

E.8 LibraryBookDescription.pd
class LibraryBookDescription ^= inherits BookDescription
abstract
var libraryId: from nat;
var cost: from nat;
var section: from string;
var myLibraryItem: from LibraryItem;
interface
selector libraryId;
selector cost;
selector section;
function getId: from nat ^= ?;
selector myLibraryItem;
end;

E.9 LibraryCatalog.pd
class LibraryCatalog ^=
abstract
var myBookInfo: from BookInfo;
var myLibraryItem: from LibraryItem;
interface
schema !buyBook post ?;
selector myBookInfo;
selector myLibraryItem;
end;

E.10 LibraryDB.pd
class LibraryDB ^=
abstract
var myBookInfo: from BookInfo;
var myUserBase: from UserBase;
interface
selector myBookInfo;
selector myUserBase;
end;
E.11 LibraryItem.pd
class LibraryItem =
abstract
var copyId: from nat;
var dateAcquired: from string;
var myBorrowing: from Borrowing;
var myLibraryBookDescription: from LibraryBookDescription;
var myLibraryCatalog: from LibraryCatalog;
interface
selector copyId;
selector dateAcquired;
function getId: from nat ^= ?;
selector myBorrowing;
selector myLibraryBookDescription;
selector myLibraryCatalog;
end;

E.12 Person.pd
class Person =
abstract
var firstName: from string;
var lastName: from string;
var dateOfBirth: from string;
interface
selector firstName;
selector lastName;
selector dateOfBirth;
function getName: from string ^= ?;
end;

E.13 Staff.pd
class Staff = inherits Person
abstract
var staffId: from nat;
var salary: from nat;
var myStaffBase: set of ref from StaffBase;
interface
selector staffId;
selector salary;
schema !increaseSalary (inc: from nat) post ?;
selector myStaffBase;
end;

E.14 StaffBase.pd
class StaffBase =
abstract
var myStaff: from Staff;
var myUserBase: from UserBase;
interface
schema !addStaff post ?;
selector myStaff;
selector myUserBase;
end;

E.15 Subject.pd
class Subject =
abstract
var name_: from string;
var myBookDescription: from BookDescription;
interface
selector name_;
function getName: from string ^= ?;
selector myBookDescription;
end;
E.16  UserBase.pd

class UserBase ^= 
abstract
var myBorrowerBase: from BorrowerBase;
var myStaffBase: from StaffBase;
var myLibraryDB: from LibraryDB;
interface
schema !addBorrower post ?;
schema !addStaff post ?;
schema !removeBorrower (b: from Borrower) post ?;
schema !removeStaff (s: from Staff) post ?;
selector myBorrowerBase;
selector myStaffBase;
selector myLibraryDB;
end;

E.17  nat.pd

class nat ^= 
interface
end;

E.18  string.pd

class string ^= 
interface
end;
F Library Specification
Here is the specification of the Library Database.

F.1 Author.pd
class Author ^= abstract
    var myName:string;
interface
    build{!myName:string};
    function getName:string ^= myName;
    redefine function toString:string ^= getName;
end;

F.2 BookDescription.pd
class BookDescription ^= abstract
    var isbn:nat;
    var title:string;
    var authors:set of Author;
    var subjects:set of Subject;
interface
    build{!isbn:nat,!title:string,!authors:set of Author,!subjects:set of Subject};
    build{!isbn:nat,!title:string}
        post authors!=set of Author{},
            subjects!=set of Subject{};
    function getIsbn:nat ^= isbn;
    redefine function toString:string ^= "ISBN: "++isbn.toString++", "++title;
    function myName:string ^= toString;
    function getAuthors:set of Author ^= authors;
    function getSubjects:set of Subject ^= subjects;
end;

F.3 BorrowerBase.pd
class BorrowerBase ^= abstract
    var allBorrowers:set of Borrower;
    invariant forall b1,b2::allBorrowers :- b1!=b2 ==> b1.getId ~= b2.getId;
interface
    build{}
        post allBorrowers!=set of Borrower();
    function uniqueId(b:Borrower):bool ^= b.getId "in usedIds;
    function uniqueId(bId:nat):bool ^= bId "in usedIds;
    function allTheBorrowers:set of Borrower ^= allBorrowers;
function usedIds:set of nat
  "= for b::allBorrowers yield b.getId;

schema !addBorrower(bor:Borrower)
  pre self.uniqueId(bor)
  post allBorrowers!=allBorrowers.append(bor)
  assert "self'.uniqueId(bor);
end;

F.4 Borrower.pd
class Borrower =
  inherits Person
abstract
  var borrowerId:nat;
  var limit:nat;
interface
  build{first:string, last:string, dob:string, !borrowerId:nat, !limit:nat}
  inherits Person{first, last, dob};
build{first:string, last:string, dob:string, !borrowerId:nat}
  inherits Person{first, last, dob}
  post limit!=5;

function getId:nat
  "= borrowerId;

schema !increaseLimit (inc:nat)
  post limit!=limit+inc;

  redefine function toString:string
    "= borrowerId.toString++":":"++getName;
end;

F.5 Borrowing.pd
class Borrowing =
abstract
  var dateBorrowed:string;
  var item: LibraryItem;
  var heldBy: Borrower;
interface
  build{!item:LibraryItem, !heldBy:Borrower,!dateBorrowed:string};

function getItem:LibraryItem
  "= item;

function borrowedBy:Borrower
  "= heldBy;
end;

F.6 LibraryBookDescription.pd
class LibraryBookDescription =
  inherits BookDescription
abstract
  var libraryId:nat;
  var cost:nat;
  var section:string;
interface
  build{libraryId:nat, !cost:nat, !section:string, isbn:nat, title:string, auth:set of Author, subj:set of Subject}
  inherits BookDescription(isbn, title, auth, subj);

build{libraryId:nat, !cost:nat, !section:string, isbn:nat, title:string}
  inherits BookDescription(isbn, title);
class LibraryCatalog

abstract

var allLibraryItems: set of LibraryItem;

function alltheBooks:set of LibraryBookDescription
  "= for item::allLibraryItems yield item.getBook;

invariant forall item1,item2::allLibraryItems
  item1~item2 => item1.getId~item2.getId;

invariant #alltheBooks <= #allLibraryItems;

invariant forall book::alltheBooks :- #(those copy::allLibraryItems
  := copy.getBook=book)<=1000;

interface

build{}
  post allLibraryItems!=set of LibraryItem{};

function newItemId(id:nat):bool
  "= forall items::allKnownItems :- id~items.getId;

function newBook(b:LibraryBookDescription):bool
  "= b in allKnownBooks;

function allKnownItems:set of LibraryItem
  "= allLibraryItems;

function allKnownBooks:set of LibraryBookDescription
  "= alltheBooks;

function allKnownItemIds:set of nat
  "= for item::allKnownItems yield item.getId;

function allKnownBookIds:set of nat
  "= for book::allKnownBooks yield book.getId;

function hasItem(copy:LibraryItem):bool
  "= copy in allKnownItems;

function uniqueItemId(id:nat):bool
  "= id ~in allKnownItemIds;

schema !addLibraryItem(copy:LibraryItem)
  pre "hasItem(copy),
  uniqueItemId(copy.getId)
  post allLibraryItems!=allLibraryItems.append(copy);

schema !removeLibraryItem(copy:LibraryItem)
  pre hasItem(copy)
  post allLibraryItems!= allLibraryItems.remove(copy);

end;

F.8 LibraryDB.pd

class LibraryDB


abstract
var books:LibraryStock;
var users:UserBase;
var today:string;
interface
  build{context:Environment}
  post today!=context.getCurrentDateTime.toString,
        books!=LibraryStock{},
        users!=UserBase{};

// Functions for building information about books from a set of strings
// ensure the string is a valid number
function isNumber(val:string):bool
  "= forall i::0..<#val :- val[i].isDigit & #val>0;

function getNumber(val:string):nat
  pre isNumber(val)
  "= nat(val);

// get the first string before a comma from a string
function firstComma(s:string):string
  "= ( let n= s.findIndex(',';
      [n<0]: s, []: s.take(n+1)
    );

// make a set of all the authors of a book
function makeAuthorSet(s:string):set of Author
decrease #s
  "= ( let stripped = s;
      [stripped.empty]: set of Author{},
      []: 
        (let a = firstComma(stripped);
         makeAuthorSet(stripped.drop(#a)).append(Author{a})
        )
    );

// as above except for subjects
function makeSubjectSet(s:string):set of Subject
decrease #s
  "= ( let stripped = s;
      [stripped.empty]: set of Subject{},
      []: 
        (let su = firstComma(stripped);
         makeSubjectSet(stripped.drop(#su)).append(Subject{su})
        )
    );

// function to make the new book from several strings
function makeLibraryBook(libID,cost,section,isbn,title,authors,subjects:string):
  LibraryBookDescription
  pre isNumber(libID), isNumber(cost), isNumber(isbn)
  "=(let authset=makeAuthorSet(authors);
     let subject=makeSubjectSet(subjects);
     LibraryBookDescription{getNumber(libID),
     getNumber(cost),
     section,
     getNumber(isbn),
     title,
     authset,
     subjsct}
    );

function makeStaffMember(first,last,doB,id,salary:string):Staff
  pre isNumber(id),isNumber(salary)
  "= Staff(first,last,doB,natNumber(id),natNumber(salary));
function makeBorrowerMember(first, last, doB, id, limit: string): Borrower
pre isNumber(id), isNumber(limit)
  " = Borrower(first, last, doB, getNumber(id), getNumber(limit));

function makeLibraryItem(id, libID, cost, section, isbn, title, authors, subjects: string): LibraryItem
pre isNumber(id), isNumber(libID), isNumber(cost), isNumber(isbn)
  " = (let libBook^ = makeLibraryBook(libID, cost, section, isbn, title, authors, subjects); LibraryItem(getNumber(id), today, libBook})
);

function makeLibraryItem(id: string, libBook: LibraryBookDescription): LibraryItem
pre isNumber(id)
  " = LibraryItem(getNumber(id), today, libBook);

schema !addStaff(first, last, doB, id, salary: string, rslt!: LibraryResultCode)
post ([-isNumber(id)|-isNumber(salary)]: rslt!=incorrectInput@LibraryResultCode, []: (let s^ = makeStaffMember(first, last, doB, id, salary); [users.uniqueId(s)]: users!addStaff(s), rslt!=success@LibraryResultCode, []: rslt!=knownUser@LibraryResultCode)
);

schema !addBorrower(first, last, doB, id, limit: string, rslt!: LibraryResultCode)
post ([-isNumber(id)|-isNumber(limit)]: rslt!=incorrectInput@LibraryResultCode, []: (let b^ = makeBorrowerMember(first, last, doB, id, limit); [users.uniqueId(b)]: users!addBorrower(b), rslt!=success@LibraryResultCode, []: rslt!=knownUser@LibraryResultCode)
);

schema !nextDay(next: string)
post today!=next;

schema !addLibraryItem(id, libID, cost, section, isbn, title, authors, subjects: string, rslt!: LibraryResultCode)
post ([-isNumber(id)|-isNumber(libID)|-isNumber(cost)|-isNumber(isbn)]: rslt!=incorrectInput@LibraryResultCode, []: (let newCopy^ = makeLibraryItem(id, libID, cost, section, isbn, title, authors, subjects); [books.hasItem(newCopy)]: rslt!=duplicateCopy@LibraryResultCode, []: books!addLibraryItem(newCopy), rslt!=success@LibraryResultCode)
);

schema !removeLibraryItem(id: string, rslt!: LibraryResultCode)
post ([-isNumber(id)]: rslt!=incorrectInput@LibraryResultCode, []: (let copyId^ = getNumber(id); [-books.hasId(copyId)]: rslt!=notOwned@LibraryResultCode, []: (let newCopy^ = books.findItem(copyId); [-books.itemAvailable(newCopy)]: rslt!=notAvailable@LibraryResultCode, []: books!removeLibraryItem(newCopy), rslt!=success@LibraryResultCode)
schema !borrowLibraryItem(itemId:string, borrowerId:string, rslt!:LibraryResultCode)
post {
    [~isNumber(itemId)|~isNumber(borrowerId)]: rslt!=incorrectInput@LibraryResultCode,
    []: (let copyId=getNumber(itemId); let borrId=getNumber(borrowerId);
        [~books.hasId(copyId)]: rslt!=notOwned@LibraryResultCode,
        [~users.hasId(borrId)]: rslt!=unregistered@LibraryResultCode,
        []: (let copy=books.findItem(copyId); let borrower=users.findUser(borrId);
            [~books.itemAvailable(copy)]: rslt!=notAvailable@LibraryResultCode,
            [~books.withinLimits(borrower)]: rslt!=maxLimits@LibraryResultCode,
            []: books!borrowLibraryItem(copy, borrower, today),
            rslt!=success@LibraryResultCode)
)
);

schema !returnLibraryItem(itemId:string,rslt!:LibraryResultCode)
post ( [~isNumber(itemId)]:rslt!=incorrectInput@LibraryResultCode,
    []: (let copyId=getNumber(itemId);
        [~books.hasId(copyId)]: rslt!=notOwned@LibraryResultCode,
        []: (let copy=books.findItem(copyId);
            [books.itemAvailable(copy)]: rslt!=available@LibraryResultCode,
            []: books!returnLibraryItem(copy),
            rslt!=success@LibraryResultCode)
)
);

schema !findItemsByAuthor(author:string,booksList!:string,rslt!:LibraryResultCode)
post ( let authset=makeAuthorSet(author);
    [authset.empty]: booksList="No Author",
    rslt!=incorrectInput@LibraryResultCode,
    []: (let bookset= books.findBookByAuthor(authset);
        booksList=bookset.toString,
        rslt!=success@LibraryResultCode)
);

schema !findItemsBySubject(subject:string,booksList!:string,rslt!:LibraryResultCode)
post ( let subjset=makeSubjectSet(subject);
    [subjset.empty]: booksList="No Subject",
    rslt!=incorrectInput@LibraryResultCode,
    []: (let bookset= books.findBookBySubject(subjset);
        booksList=bookset.toString,
        rslt!=success@LibraryResultCode)
)
end;

F.9 LibraryItem.pd

class LibraryItem ^=
    abstract
    var copyId:(nat in 0..999);
    var dateAcquired:string;
    var myLibraryBookDescription: LibraryBookDescription;

interface
    build{!copyId:nat,!dateAcquired:string,
        !myLibraryBookDescription:LibraryBookDescription}
    pre copyId<999;

    function getId:nat = (myLibraryBookDescription.getId*1000) + copyId;

131
function getBook:LibraryBookDescription
  ^= myLibraryBookDescription;
end;

F.10 LibraryResultCode.pd
class LibraryResultCode ^=
  enum
    success,  // successful operation
    unauthorized,  // the requestor is not a member of staff
    unregistered,  // the borrower is not a member of library
    notOwned,  // the book is not owned by the library
    notAvailable,  // the book is currently checked out
    available,  // the book is available
    unknownAuthor,  // the author is not in the database
    unknownSubject,  // the subject is not in the database
    maxLimits,  // the borrower is over their borrowing limit
    duplicateCopy,  // the copy being added already exists
    duplicateBook,  // the book being added already exists
    noCopy,  // the copy isn’t registered
    knownUser,  // the user exists
    neverBeenReturned,  // the copy has never been returned (i.e. no previous borrower)
    incorrectInput  // the input to the wrapper class is incorrect
  end;

F.11 LibraryStock.pd
class LibraryStock ^=
  abstract
    var catalog: LibraryCatalog;
    var currentlyBorrowed: set of Borrowing;

  interface
    build{}
    post catalog!=LibraryCatalog(),
       currentlyBorrowed!=set of Borrowing();

    function checkedOut:set of LibraryItem
      ^= for b::currentlyBorrowed yield b.getItem;

    function available:set of LibraryItem
      ^= catalog.allKnownItems -- checkedOut;

    function itemAvailable(c:LibraryItem):bool
      ^= c in available;

    function booksHeld:set of LibraryBookDescription
      ^= catalog.allKnownBooks;

    function itemCurrentlyBorrowed(copy:LibraryItem):bool
      ^= copy in checkedOut;

    function copyBorrowing(copy:LibraryItem):Borrowing
      pre itemCurrentlyBorrowed(copy)
      ^= that b::currentlyBorrowed :- b.getItem = copy;

    function hasItem(copy:LibraryItem):bool
      ^= catalog.hasItem(copy);

    function hasId(id:nat):bool
      ^= id in catalog.allKnownItemIds;

    function withinLimits(bor:Borrower):bool
function findItem(id:nat):LibraryItem
  pre hasId(id)
  ^= that copy::catalog.allKnownItems :- copy.getId = id;

function findBookByAuthor(aut:set of Author):set of LibraryBookDescription
  ^= those b::booksHeld :- aut <<= b.getAuthors;

function findBookBySubject(sub:set of Subject):set of LibraryBookDescription
  ^= those b::booksHeld :- sub <<= b.getSubjects;

function findCopiesBorrowedByBorrower(b:Borrower):set of LibraryItem
  ^= those item::checkedOut :- copyBorrowing(item).borrowedBy = b;

schema !borrowLibraryItem (copy:LibraryItem, bor:Borrower, date:string)
  pre itemAvailable(copy)
  post currentlyBorrowed!=currentlyBorrowed.append(Borrowing{copy,bor,date});

schema !returnLibraryItem (copy:LibraryItem)
  pre itemCurrentlyBorrowed(copy)
  post currentlyBorrowed!=currentlyBorrowed.remove(copyBorrowing(copy));

schema !addLibraryItem(copy:LibraryItem)
  pre ~hasItem(copy)
  post catalog!addLibraryItem(copy);

schema !removeLibraryItem(copy:LibraryItem)
  pre hasItem(copy),
    itemAvailable(copy)
  post catalog!removeLibraryItem(copy);

end;

F.12 Person.pd

class Person ^=
  abstract
  var firstName:string;
  var lastName:string;
  var dateOfBirth:string;
  interface
    build{firstName:string,lastName:string,dateOfBirth:string};

  function getName:string
    ^= firstName++" "++lastName;

  redefine function toString:string
    ^= getName;
  end;

F.13 Staff.pd

class Staff ^=
  inherits Person
  abstract
  var staffId:nat;
  var salary:nat;
  interface
    build{first:string,last:string,dOB:string,staffId:nat,salary:nat}
      inherits Person{first,last,dOB};

  redefine function toString:string
    ^= staffId.toString++":"++getName;
  end;
function getId:nat
   ^= staffId;

schema !increaseSalary (inc:nat)
   post salary!=salary+inc;

end;

F.14 StaffBase.pd
class StaffBase ^=
abstract
   var allStaff:set of Staff;
   invariant forall s1,s2::allStaff :- s1\=s2 \Longrightarrow s1.getId \= s2.getId
   interface
      build{}
      post allStaff!=set of Staff{};
    
      function uniqueId(s:Staff):bool
         ^= s.getId ~in usedIds;
    
      function usedIds:set of nat
         ^= for s::allStaff yield s.getId;
    
    schema !addStaff(sta:Staff)
       pre uniqueId(sta)
       post allStaff!=allStaff.append(sta);
end;

F.15 Subject.pd
class Subject ^=
abstract
   var myName:string;
   interface
      build{!myName:string};
      
      function getName:string
         ^= myName;
      
      redefine function toString:string
         ^= getName;
end;

F.16 UserBase.pd
class UserBase ^=
abstract
   var allBorrowers: BorrowerBase;
   var allStaff: StaffBase;
   
   interface
      build{}
      post allBorrowers!=BorrowerBase{},
         allStaff!=StaffBase{};
      
      function usedIds:set of nat
         ^= allBorrowers.usedIds ++ allStaff.usedIds;
      
      function uniqueId(u:Borrower):bool
         ^= ~hasId(u.getId);
      
      function uniqueId(u:Staff):bool
function hasId(uId:nat):bool
  ^= uId in usedIds;

function findUser(id:nat):Borrower
  pre hasId(id)
  ^= that user::allBorrowers.allTheBorrowers :- user.getId = id;

schema !addBorrower(bor:Borrower)
  pre uniqueId(bor)
  post allBorrowers!addBorrower(bor);

schema !addStaff(sta:Staff)
  pre uniqueId(sta)
  post allStaff!addStaff(sta);
end;
G  Library Refinement

Here are the Refined Perfect source code files of the Library Database Software.

G.1  HashedBucket.pd

class HashedBucket of X require X has operator ={arg};
  function hash:nat end=

abstract
  var
    hb:map of (nat ->set of X),
    // the table is represented as a map of natural numbers to sets
    // this prevents the creation of empty redundant entries
    // necessary as perfect has dynamic allocation of sequences
    // (Not arrays)
    hashSize:nat;  // keep track of the max size of the hash table

    invariant hashSize>0;
    // all the keys are in the range of 0 to size
    invariant forall x::hb.dom :- 0<=x<=hashSize;
    // the hashtable is never bigger than the size
    invariant #hb <= hashSize;
    // all the elements in the hb have valid keys
    // (says nothing about elements not in hb)
    // needed? to prove being a member -> the key is in the hb for removal
    axiom assert forall a:X :- a in self ==> hash(a) in hb.dom;

  confined
    // methods accessible to the object only

    // add an element that has a new hash key
    schema !addNewHash(pos:nat,a:X)
      pre 0<=pos<=hashSize,  // key in range
      pos ~in hb.dom,      // key not in table
      #hb < hashSize  // we have space in the hashtable
      post (let newMap^=set of X{a};  // new element
            hb!=hb.append(pair of (nat,set of X){pos,newMap})
          )
    assert forall elem::self.ran :- elem in self',
          a in self';

    // add an element whose key is in the system
    schema !addOldHash(pos:nat,a:X)
      pre 0<=pos<=hashSize,  // key in range
      pos in hb.dom  // key in table
      post (let newMap^=hb[pos].append(a);  // old element + new element
            hb!=hb.remove(pos) then  // remove the old
            hb!=hb.append(pair of (nat,set of X){pos,newMap})
            // add everything back
          )
    assert forall elem::self.ran :- elem in self',
          a in self';

  interface

    // construct an empty hashedbucket
    build{!hashSize:nat}
      pre hashSize>0
      post hb!=map of (nat ->set of X){};

    //**Some standard operations for a data structure**/**

    // get all the elements of the hashedbucket
    function ran:set of X
      ^= flatten(hb.ran);
// get all the keys used in the hashedbucket
function keysUsed:set of nat
  ^= hb.dom;

// is the hashedbucket empty
function empty:bool
  ^= hb.empty;

// how many elements are in the hashedbucket
operator #:nat
  ^= #ran;

// what is the maximum Hash value
function maxHashSize:nat
  ^= hashSize;

// is the element in the hashtable
operator (a:X)in:bool
  ^= ([hash(a) in hb.dom]: // is the key valid?
    a in hb[hash(a)], // is the element in that bucket
    []: false
  );

// undefined hash function. Ensures the result is in the range though
// We might want the classes that use the system to define their own hash
// function. Problems will occur in 1) building the code; 2) Primitive types
function hash(a:X):nat
  satisfy 0<=result<=hashSize
  via
    value (a.hash)% hashSize
end;

// add an element to the hashedbucket
schema !add(a:X)
  post (let pos^= hash(a);
    [pos in hb.dom]: // we have that key
      !addOldHash(pos,a),
    []: //=hb < hashSize]: // we have space for new hash
      !addNewHash(pos,a)
    )
  assert forall elem::self.ran :- elem in self',
                 a in self';

// remove an element from the hashedbucket provided it is already there
schema !remove(a:X)
  pre a in self
  post (let pos^= hash(a);
    let newMap^=hb[pos].remove(a);  // new bucket, without element
    hb!=hb.remove(pos) then // remove old bucket
      hb!=hb.append(pair of(nat,set of X){pos,newMap}) // all new bucket
    )
  assert forall elem::self'.ran :- elem in self',
                 a ~in self';
end;

G.2 PriorityQueue.pd

class PriorityQueue of X require X has operator =(arg);
  function priority:nat end ^=

  abstract
    var
      myQueue:Heap of X;
interface
build{
  post myQueue!="Heap of X";
}

schema !insert(a:X)
  post myQueue!="insert(a)";

function getElement:X
  pre ~myQueue.empty
  ^= myQueue.largest;

schema !remove
  pre ~myQueue.empty
  post myQueue!="remove(getElement)";

function ran:set of X
  ^= myQueue.ran;
end;

//*******************************************************
//* File: C:\Projects\Perfect\Heap\Heap.pd
//* Author: Gareth Carter
//* Created: 12:34:53 on Monday February 23rd 2004 UTC
//*******************************************************
class Heap of X require X has operator ==(arg);
  function priority:nat end ^=
abstract
  var
    l:seq of X;
  invariant isHeap(l);
confined
nonmember function heapify(xs:seq of X,low,high:nat):seq of X
  pre ~xs.empty,
    0<low<=high<=#xs
  decrease high-low
  ^= (let large^=2*low; assert large>1;
      [large<=high]:
        ([large<high
        & xs[<large].priority<xs[large].priority]
        & xs[<low].priority < xs[large].priority]:
          heapify(exchange(xs,low,large+1),large+1,high),
        [xs[<low].priority < xs[large].priority]:
          heapify(exchange(xs,low,large),large,high),
        []: xs
      ),
      []: xs
    );

function sift_up(xs:seq of X,pos:nat):seq of X
  pre 0<pos<=#xs
  decrease pos
  ^= (let parentPos=" pos/2;
      [pos = 1]: // at root
        xs,
      [xs[<pos].priority<xs[<parentPos].priority]:
        // my priority less than parents
        xs,
      []: // my priority greater than parents
        sift_up(exchange(xs,parentPos,pos),parentPos)
function sift_down(xs: seq of X): seq of X
  pre ~xs.empty
  ^= ([#xs=1]:
   seq of X{},
   []:
   heapify(exchange(xs,1,#xs).front,1,#xs))

interface
  build{}
  post l!=seq of X{};
  // Start of HEAP DEFINITION //
  nonmember function isHeap(a: seq of X): bool
    ^= forall p::1..(#a/2) :- greaterPriority(p,a);
  nonmember function greaterPriority(p: nat, a: seq of X): bool
    pre 0<p<=#a
    ^= (let child^=2*p;
       [child<#a]: // has two children
         a[<p].priority>=a[<child].priority
         & a[<p].priority>=a[<child+1].priority,
       [child=#a]: // has one child
         a[<p].priority>=a[<child].priority,
       []: // has no children
         true)
    );
  // End of HEAP DEFINITION //
  nonmember function exchange(xs: seq of X,i:nat,j:nat): seq of X
    pre 0<i<j<=#xs
    ^= xs.take(<i).append(xs[j])++
      xs.take(<j).drop(i).append(xs[i])++
      xs.take(#xs).drop(j)
    assert result.ranb = xs.ranb;
  nonmember schema swap(xs!: seq of X,i:nat,j:nat)
    pre 0<i<j<=#xs
    post (let temp^= xs[<i];
      xs[<i]=xs[<j] then
      xs[<j]=temp)
    assert xs.ranb = xs'.ranb;
  schema !insert(a:X)
    post l=sift_up(l.append(a),#l+1)
    assert forall elem::self.ran :- elem in self',
      a in self';
  schema !remove(a:X)
    pre ~empty,
      largest=a
    post l=sift_down(l)
    assert forall elem::self'.ran :- elem in self,
      a "in self';
  function largest:X
    pre ~empty
    ^= l.head;
  function empty: bool
    ^= l.empty;
G.3 LibraryCatalog.pd

```plaintext
class LibraryCatalog {
  var allLibraryItems: set of LibraryItem;

  ghost function alltheBooks: set of LibraryBookDescription
    "= for item::allLibraryItems yield item.getBook;

  invariant forall item1, item2::allLibraryItems
    := item1 = item2 => item1.getId = item2.getId;
  invariant #alltheBooks <= #allLibraryItems;
  invariant forall book::alltheBooks : 
    #(those copy::allLibraryItems
      := copy.getBook=book) <= 1000;

  internal

  var allLibItems : HashedBucket of LibraryItem;

  function allLibraryItems
    "= allLibItems.ran;

  interface

  operator =(arg);

  build{}
  post allLibraryItems!=set of LibraryItem{}
  via allLibItems!=HashedBucket of LibraryItem{1000}
  end;

  function newItemId(id:nat):bool
    "= forall items::allKnownItems :- id=items.getId;

  function newBook(b:LibraryBookDescription):bool
    "= b in allKnownBooks;

  function allKnownItems: set of LibraryItem
    "= allLibraryItems
    via
      value allLibItems.ran
      end;

  function allKnownBooks: set of LibraryBookDescription
    "= alltheBooks
    via
      value for item::allKnownItems yield item.getBook;
      end;

  function allKnownItemIds: set of nat
    "= for item::allKnownItems yield item.getId;

  function allKnownBookIds: set of nat
```
- for book::allKnownBooks yield book.getId;

function hasItem(copy:LibraryItem):bool
- = copy in allKnownItems
  via
  value (copy in allLibItems)
end;

function uniqueItemId(id:nat):bool
- = id ~in allKnownItemIds;

schema !addLibraryItem(copy:LibraryItem)
pre ~hasItem(copy),
uniqueItemId(copy.getId)
post allLibraryItems!allLibraryItems.append(copy)
via
allLibItems!add(copy)
end;

schema !removeLibraryItem(copy:LibraryItem)
pre hasItem(copy)
post allLibraryItems!= allLibraryItems.remove(copy)
via allLibItems!remove(copy)
end;

G.4 LibraryItem.pd
class LibraryItem ^=
abstract
  var copyId:(nat in 0..999);
  var dateAcquired:string;
  var myLibraryBookDescription: LibraryBookDescription;

interface
build{!copyId:nat,!dateAcquired:string,!myLibraryBookDescription:LibraryBookDescription}
pre copyId<=999;

function getId:nat ^= (myLibraryBookDescription.getId*1000) + copyId;

function getBook:LibraryBookDescription
- = myLibraryBookDescription;

function hash:nat
- = getId;
end;

G.5 LibraryStock.pd
class LibraryStock ^=
abstract
  var catalog: LibraryCatalog;
  var currentlyBorrowed: set of Borrowing;

internal
  var borrowed:PriorityQueue of Borrowing;

interface
operator =(arg);
build{}
  post catalog!=LibraryCatalog{},
     currentlyBorrowed!=set of Borrowing{}
  via catalog!=LibraryCatalog{},
     borrowed!=PriorityQueue of Borrowing{}
end;

function checkedOut:set of LibraryItem
"= for b::currentlyBorrowed yield b.getItem
via value(for b::borrowed.ran yield b.getItem)
end;

function available:set of LibraryItem
"= catalog.allKnownItems -- checkedOut;

function itemAvailable(c::LibraryItem):bool
"= c in available;

function booksHeld:set of LibraryBookDescription
"= catalog.allKnownBooks;

function itemCurrentlyBorrowed(copy:LibraryItem):bool
"= copy in checkedOut;

function copyBorrowing(copy:LibraryItem):Borrowing
pre itemCurrentlyBorrowed(copy)
"= that b::currentlyBorrowed :- b.getItem = copy
via value(that b::borrowed.ran :- b.getItem = copy)
end;

function hasItem(copy:LibraryItem):bool
"= catalog.hasItem(copy);

function hasId(id:nat):bool
"= id in catalog.allKnownItemIds;

function withinLimits(bor:Borrower):bool
"= true;

function findItem(id:nat):LibraryItem
pre hasId(id)
"= that copy::catalog.allKnownItems :- copy.getId = id;

function findBookByAuthor(aut:set of Author):set of LibraryBookDescription
"= those b::booksHeld :- aut <<= b.getAuthors;

function findBookBySubject(sub:set of Subject):set of LibraryBookDescription
"= those b::booksHeld :- sub <<= b.getSubjects;

function findCopiesBorrowedByBorrower(b:Borrower):set of LibraryItem
"= those item::checkedOut :- copyBorrowing(item).borrowedBy = b;

schema !borrowLibraryItem (copy:LibraryItem, bor:Borrower, date:string)
pre itemAvailable(copy)
post currentlyBorrowed!=currentlyBorrowed.append(Borrowing{copy,bor,date})
via borrowed!insert(Borrowing{copy,bor,date})
end;

schema !returnLibraryItem (copy:LibraryItem)
pre itemCurrentlyBorrowed(copy)
post currentlyBorrowed!=currentlyBorrowed.remove(copyBorrowing(copy))
via borrowed!remove
end;

schema !addLibraryItem(copy:LibraryItem)
pre `hasItem(copy)`
post catalog!addLibraryItem(copy);

schema !removeLibraryItem(copy:LibraryItem)
pre hasItem(copy),
    itemAvailable(copy)
post catalog!removeLibraryItem(copy);

end;
H  Library Java Implementation

Here is the Java source code of the Library Database generated from the Perfect Refinement in Appendix G.

H.1  Author.java

package oolibrary;

import Ertsys.*;

// Packages imported
import oolibrary.*;

public class Author extends _eAny {
    protected _eSeq myName;
    public Author (_eSeq _vmyName, char _t0_vmyName)
    {
        super ();
        myName = _vmyName;
    }

    public _eSeq getName ()
    {
        return myName;
    }

    public _eSeq _rtoString ()
    {
        return getName ();
    }

    public boolean _lEqual (Author _vArg_11_9)
    {
        if (this == _vArg_11_9) return true;
        return _vArg_11_9.myName._lEqual (myName);
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == Author.class &&
        _lEqual ((Author) _lArg));
    }
}

// End of file.

H.2  BookDescription.java

package oolibrary;

import Ertsys.*;

// Packages imported
import oolibrary.*;

public class BookDescription extends _eAny {
    protected int isbn;
    protected _eSeq title;

protected _eSet authors;
protected _eSet subjects;

public BookDescription (int _visbn, _eSeq _vtitle, char _t0_vtitle, _eSet _vauthors,
                        Author _t0_vauthors, _eSet _vsubjects, Subject _t0_vsubjects)
{
    super ();
    isbn = _visbn;
    title = _vtitle;
    authors = _vauthors;
    subjects = _vsubjects;
}

public BookDescription (int _visbn, _eSeq _vtitle, char _t0_vtitle)
{
    super ();
    isbn = _visbn;
    title = _vtitle;
    authors = new _eSet ();
    subjects = new _eSet ();
}

public int getIsbn ()
{
    return isbn;
}

public _eSeq _rtoString ()
{
    return _eSystem._lString ("ISBN: ")._oPlus (_eSystem._ltoString (isbn), (_eTemplate_0) null)._oPlus (_eSystem._lString (", "), (_eTemplate_0) null)._oPlus (title, (_eTemplate_0) null);
}

public _eSeq myName ()
{
    return _rtoString ();
}

public _eSet getAuthors ()
{
    return authors;
}

public _eSet getSubjects ()
{
    return subjects;
}

public boolean _lEqual (BookDescription _vArg_11_9)
{
    if (this == _vArg_11_9) return true;
}

public boolean equals (_eAny _lArg)
{
    return _lArg == this || (_lArg != null && _lArg.getClass () == BookDescription.class && _lEqual ((BookDescription) _lArg));
}

// End of file.
H.3 BorrowerBase.java

package oolibrary;

import Ertsys.*;

// Packages imported
import oolibrary.*;

public class BorrowerBase extends _eAny {
    final void _lc_BorrowerBase (String _lArg)
    {
        if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
            maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                boolean _vQuantifierResult_12_15;
                {
                    _vQuantifierResult_12_15 = true;
                    int _vCaptureCount_b1_12_29 = allBorrowers._oHash ();
                    int _vLoopCounter_12_22 = 0;
                    for (;;)
                    {
                        if (((_vLoopCounter_12_22 == _vCaptureCount_b1_12_29) || (!
                            _vQuantifierResult_12_15))) break;
                        boolean _vQuantifierResult_12_25;
                        {
                            _vQuantifierResult_12_25 = true;
                            int _vCaptureCount_b2_12_29 = allBorrowers._oHash ();
                            int _vLoopCounter_12_25 = 0;
                            for (;;)
                            {
                                if (((_vLoopCounter_12_25 == _vCaptureCount_b2_12_29) || (!
                                    _vQuantifierResult_12_25))) break;
                                _vQuantifierResult_12_25 = true;
                                int _vCaptureCount_b3_12_29 = allBorrowers._oHash ();
                                int _vLoopCounter_12_25 = 0;
                                for (;;)
                                {
                                    if (((_vLoopCounter_12_25 == _vCaptureCount_b3_12_29) || (!
                                        _vQuantifierResult_12_25))
                                        break;
                                    _vQuantifierResult_12_25 = true;
                                    _vCaptureCount_b2_12_29 = allBorrowers._oHash ();
                                    int _vLoopCounter_12_25 = 0;
                                    for (;;)
                                    {
                                        if (((_vLoopCounter_12_25 == _vCaptureCount_b2_12_29) || (!
                                            _vQuantifierResult_12_25)));
                                    }
                                    else
                                    {
                                        _vLoopCounter_12_25 = _eSystem._oSucc (_vLoopCounter_12_25);
                                    }
                                }
                                _vQuantifierResult_12_25 = _vQuantifierResult_12_25;
                                if (((_vQuantifierResult_12_25))
                                    {
                                }
                                else
                                {
                                    _vLoopCounter_12_22 = _eSystem._oSucc (_vLoopCounter_12_22);
                                }
                            }
                        }
                    }
                }
            }
        } else throw new _xClassInvariantItem (;

}
protected _eSet allBorrowers;

public BorrowerBase ()
{
    super ();
    allBorrowers = new _eSet ();
    _lc_BorrowerBase ("BorrowerBase.pd:15,14");
}

public boolean uniqueId (Borrower b)
{
    return (!usedIds ()._ovIn (((_eAny) new _eWrapper_int (b.getId ()))));
}

public boolean uniqueId (int bId)
{
    return (!usedIds ()._ovIn (((_eAny) new _eWrapper_int (bId))));
}

public _eSet allTheBorrowers ()
{
    return allBorrowers;
}

public _eSet usedIds ()
{
    _eSet _vForYield_27_12 = new _eSet ();
    int _vCaptureCount_b_27_19 = allBorrowers._oHash ();
    int _vLoopCounter_27_16 = 0;
    for (;;)
    {
        if (_vLoopCounter_27_16 == _vCaptureCount_b_27_19) break;
        _vForYield_27_12 = _vForYield_27_12.append (((_eAny) new _eWrapper_int (((Borrower) allBorrowers._oIndex (_vLoopCounter_27_16)).getId ())));
        _vLoopCounter_27_16 = _eSystem._oSucc (_vLoopCounter_27_16);
    }
    return _vForYield_27_12;
}

public void addBorrower (Borrower bor)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(uniqueId (bor))) throw new _xPre ("BorrowerBase.pd:30,18");
        }
    }
catch (_xCannotEvaluate _lException)
{
}
_eSystem.currentCheckNesting --;
}
allBorrowers = allBorrowers.append ((_eAny) bor);
_lClassInvariantCheck ("BorrowerBase.pd:31,14");

public boolean _lEqual (BorrowerBase _vArg_11_9)
{
    if (this == _vArg_11_9) return true;
    return _vArg_11_9.allBorrowers._lEqual (allBorrowers);
}

public boolean equals (_eAny _lArg)
{
    return _lArg == this || (_lArg != null && _lArg.getClass () == BorrowerBase.class
    && _lEqual ((BorrowerBase) _lArg));
}

// End of file.

H.4 Borrower.java

package oolibrary;

import Ertsys.*;

// Packages imported
import oolibrary.*;

public class Borrower extends Person
{
    protected int borrowerId;
    protected int limit;
    public Borrower (_eSeq first, char _t0first, _eSeq last, char _t0last, _eSeq dOB, char
    _t0dOB, int _vborrowerId, int _vlimit)
    {
        super (first, (char) 0, last, (char) 0, dOB, (char) 0);
        borrowerId = _vborrowerId;
        limit = _vlimit;
    }
    public Borrower (_eSeq first, char _t0first, _eSeq last, char _t0last, _eSeq dOB, char
    _t0dOB, int _vborrowerId)
    {
        super (first, (char) 0, last, (char) 0, dOB, (char) 0);
        borrowerId = _vborrowerId;
        limit = 5;
    }
    public int getId ()
    {
        return borrowerId;
    }
    public void increaseLimit (int inc)
    {
        limit = (limit + inc);
    }
}
public _eSeq _rtoString ()
{
    return _eSystem._ltoString (borrowerId)._oPlusPlus (_eSystem._lString (":") , (_eTemplate_0) null)._oPlusPlus (getName (), (_eTemplate_0) null);
}

public boolean _lEqual (Borrower _vArg_12_9)
{
    if (this == _vArg_12_9) return true;
    if (!super._lEqual (_vArg_12_9)) return false;
    return ((_vArg_12_9.borrowerId == borrowerId) && (_vArg_12_9.limit == limit));
}

public boolean equals (_eAny _lArg)
{
    return _lArg == this || (_lArg != null && _lArg.getClass () == Borrower.class &&
    _lEqual ((Borrower) _lArg));
}

// End of file.

H.5 Borrowing.java
package oolibrary;

import Ertsys.*;
// Packages imported
import oolibrary.*;

public class Borrowing extends _eAny
{
    protected _eSeq dateBorrowed;
    protected LibraryItem item;
    protected Borrower heldBy;
    public Borrowing (LibraryItem _vitem, Borrower _vheldBy, _eSeq _vdateBorrowed, char _t0_vdateBorrowed)
    {
        super ();
        item = _vitem;
        heldBy = _vheldBy;
        dateBorrowed = _vdateBorrowed;
    }

    public LibraryItem getItem ()
    {
        return item;
    }

    public Borrower borrowedBy ()
    {
        return heldBy;
    }

    public int priority ()
    {
        return 1;
    }
}
public boolean _lEqual (Borrowing _vArg_11_9)
{
    if (this == _vArg_11_9) return true;
    return ((_vArg_11_9.item._lEqual (item) && _vArg_11_9.heldBy._lEqual (heldBy)) &&
            _vArg_11_9.dateBorrowed._lEqual (dateBorrowed));
}

public boolean equals (_eAny _lArg)
{
    return _lArg == this || (_lArg != null && _lArg.getClass () == Borrowing.class &&
                      _lEqual ((Borrowing) _lArg));
}

// End of file.

H.6 HashedBucket.java
package oolibrary;

import Ertsys.*;

// Packages imported
import oolibrary.*;

class _n1_HashedBucket extends _eAny
{
    final void _lc_HashedBucket (String _lArg)
    {
        if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
    maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!(0 < hashSize)) throw new _xClassInvariantItem (
                    "HashedBucket.pd:23,27", _lArg);
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
        if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
    maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                boolean _vQuantifierResult_25_19;
                {
                    _eSet _vCaptureBound_x_25_32 = hb.dom ();
                    _vQuantifierResult_25_19 = true;
                    int _vCaptureCount_x_25_32 = _vCaptureBound_x_25_32._oHash ();
                    int _vLoopCounter_25_26 = 0;
                    for (;;)
                    {
                        if (((_vLoopCounter_25_26 == _vCaptureCount_x_25_32) || (!
                            _vQuantifierResult_25_19)) break;
                        _vQuantifierResult_25_19 = ((((_eWrapper_int)
                            _vCaptureBound_x_25_32._oIndex (_vLoopCounter_25_26)).value <=
                    )
                }
            }
        }
hashSize) && (0 <= ((_eWrapper_int) _vCaptureBound_x_25_32. _oIndex (_vLoopCounter_25_26)).value));
if (!(_vQuantifierResult_25_19))
{
}
else
{
  _vLoopCounter_25_26 = _eSystem._oSucc (_vLoopCounter_25_26);
}
}
if (!(_vQuantifierResult_25_19)) throw new _xClassInvariantItem ("HashedBucket.pd:25,19", _lArg);
catch (_xCannotEvaluate _lException)
{
}
_eSystem.currentCheckNesting --;
}
if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem. maxCheckNesting)
{
  _eSystem.currentCheckNesting ++;
  try
  {
    if (!((_hB._oHash () <= hashSize))) throw new _xClassInvariantItem ("HashedBucket.pd:27,23", _lArg);
  }
  catch (_xCannotEvaluate _lException)
  {
  }
  _eSystem.currentCheckNesting --;
}
void _lClassInvariantCheck (String _lArg)
{
  _lc_HashedBucket (_lArg);
}
protected _eMap hb;
protected int hashSize;
protected void addNewHash (int pos, _eAny a)
{
  if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
  {
    _eSystem.currentCheckNesting ++;
    try
    {
      if (!((((0 <= pos) && (pos <= hashSize)))) throw new _xPre ("HashedBucket.pd:39,14");
    }
    catch (_xCannotEvaluate _lException)
    {
    }
    _eSystem.currentCheckNesting --;
  }
  if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
  {
    _eSystem.currentCheckNesting ++;
    try
    {
      if (!((hb.dom ())._ovIn (((_eAny) new _eWrapper_int (pos))))) throw new _xClassInvariantItem ("HashedBucket.pd:25,19", _lArg);
    }
    catch (_xCannotEvaluate _lException)
    {
    }
    _eSystem.currentCheckNesting --;
  }
}
protected void addOldHash (int pos, _eAny a) {
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting) {
        _eSystem.currentCheckNesting ++;
        try {
            if (!(hb.dom()._ovIn (((_eAny) new _eWrapper_int (pos)))) throw new _xPre ("HashedBucket.pd:51,17");
        } catch (_xCannotEvaluate _lException) {
        }
        _eSystem.currentCheckNesting --;
    }
    _eSet _vLet_newMap_52_19 = ((_eSet) hb._oIndex (((_eAny) new _eWrapper_int (pos))).append (a);
    hb = hb.remove (((_eAny) new _eWrapper_int (pos)));
    hb = hb.append (new _ePair (((_eAny) new _eWrapper_int (pos)), ((_eAny) _vLet_newMap_52_19)), (_eTemplate_0) null, (_eTemplate_1) null);
    _lClassInvariantCheck ("HashedBucket.pd:52,14");
}
public _n1_HashedBucket (int _vhashSize)
{
  super ();
  if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting )
  {
    _eSystem.currentCheckNesting ++;
    try
    {
      if (!((0 < _vhashSize))) throw new _xPre ("HashedBucket.pd:64,21");
    } catch (_xCannotEvaluate _lException) {
    }
    _eSystem.currentCheckNesting --;
  }
  hashSize = _vhashSize;
  hb = new _eMap ();
  _lc_HashedBucket ("HashedBucket.pd:65,14");
}

public _eSet ran ()
{
  return Ertsys.RtsGlobals.flatten (hb.ran (), _eSet) null, (_eTemplate_0) null);
}

public _eSet keysUsed ()
{
  return hb.dom ();
}

public boolean empty ()
{
  return hb.empty ();
}

public int _oHash ()
{
  return ran ()._oHash ();
}

public int maxHashSize ()
{
  return hashSize;
}

public boolean _ovIn (_eAny a)
{
  return (hb.dom ()_.ovIn (((_eAny) new _eWrapper_int (hash (a))))
    ? (((_eSet) hb_.oIndex (((_eAny) new _eWrapper_int (hash (a)))))._ovIn (a) :
      false);
}

public int hash (_eAny a)
{
  return _eSystem._oMod (a.hash (), hashSize);
}

public void add (_eAny a)
{
  int _vLet_pos_108_19 = hash (a);
  if (hb.dom ()_.ovIn (((_eAny) new _eWrapper_int (_vLet_pos_108_19))))
  {
    addOldHash (_vLet_pos_108_19, a);
  }
else
{
    addNewHash (_vLet_pos_108_19, a);
}
_lClassInvariantCheck ("HashedBucket.pd:108,14");
}

public void remove (_eAny a)
{
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
    _eSystem.currentCheckNesting ++;
    try
    {
        if (!(this._ovIn (a))) throw new _xPre ("HashedBucket.pd:119,15");
    }
    catch (_xCannotEvaluate _lException)
    {
    }
    _eSystem.currentCheckNesting --;
}
int _vLet_pos_120_19 = hash (a);
_eSet _vLet_newMap_121_19 = ((_eSet) hb._oIndex (((_eAny) new _eWrapper_int (_vLet_pos_120_19))).remove (a));
hb = hb.remove (((_eAny) new _eWrapper_int (_vLet_pos_120_19))); hb = hb.append (new _ePair (((_eAny) new _eWrapper_int (_vLet_pos_120_19)), ((_eAny) _vLet_newMap_121_19)), (_eTemplate_0) null, (_eTemplate_1) null);
_lClassInvariantCheck ("HashedBucket.pd:120,14");
}

public boolean _lEqual (_n1_HashedBucket _vArg_16_9)
{
if (this == _vArg_16_9) return true;
return (_vArg_16_9.hb._lEqual (hb) && (_vArg_16_9.hashSize == hashSize));
}

public boolean equals (_eAny _lArg)
{
    return _lArg == this || (_lArg != null && _lArg.getClass () == _n1_HashedBucket.class && _lEqual ((_n1_HashedBucket) _lArg));
}


// End of file.

H.7 LibraryAccess.java
// Simple Java Swing application to front-end a Perfect Developer program
package oolibrary;
import java.io.IOException;
import java.io.File;
import java.awt.
import java.awt.event.*;
import javax.swing.
import Ertsys.*;
// This is the top-level class
public class LibraryAccess implements ActionListener
{
JFrame userFrame, catalogFrame, stockFrame;
JPanel userPanel, catalogPanel, stockPanel;

// Staff Database Frame
// Person information - Borrowers and Staff
JTextField firstName, lastName, dateOfBirth, userId, salary, limit;
JButton addStaff, addBorrower;

// LibraryCatalog Frame
// Book Description Information
JTextField libId, cost, section, isbn, bookTitle, author, subject;

// Library Item Information
JTextField copyId;
JButton addCopy, removeCopy;

// Library Stock Frame
// Id input parameters
JTextField staffIdQ, borrowerIdQ, copyIdQ, bookIdQ, authorQ, subjectQ;
JButton borrowCopy, returnCopy;

JButton findAuthor, findSubject;
JButton whatBorrowedBy, whoBorrowed;

// output for all windows
JLabel uStatus, cStatus, sStatus;

LibraryDB backend;

// Constructor
LibraryAccess(Environment context) {
    // Create the back end
    backend = new LibraryDB(context);

    // Create the frame and container.
    userFrame = new JFrame("User Database Control");
    userPanel = new JPanel();
    userPanel.setLayout(new GridLayout(10, 2));
    catalogFrame = new JFrame("Catalog Control");
    catalogPanel = new JPanel();
    catalogPanel.setLayout(new GridLayout(10, 2));
    stockFrame = new JFrame("Stock Control");
    stockPanel = new JPanel();
    stockPanel.setLayout(new GridLayout(10, 2));

    // Add the widgets.
    addWidgets();

    // Add the panel to the frame.
    userFrame.getContentPane().add(userPanel);
    catalogFrame.getContentPane().add(catalogPanel);
    stockFrame.getContentPane().add(stockPanel);

    // Exit when the window is closed.
    userFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    catalogFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    stockFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // Show the app.
    userFrame.pack();
    userFrame.setVisible(true);
    catalogFrame.pack();
    catalogFrame.setVisible(true);
    stockFrame.pack();
}
stockFrame.setVisible(true);

// Create and add the widgets for app.
private void addWidgets()
{
  // Create widgets.
  firstName = new JTextField(20);
  lastName = new JTextField(20);
  dateOfBirth = new JTextField(20);
  userId = new JTextField(20);
  salary = new JTextField(20);
  limit = new JTextField(20);
  addStaff = new JButton("Add Staff");
  addBorrower = new JButton("Add Borrower");
  libId = new JTextField(20);
  cost = new JTextField(20);
  section = new JTextField(20);
  isbn = new JTextField(20);
  bookTitle = new JTextField(20);
  author = new JTextField(20);
  subject = new JTextField(20);
  copyId = new JTextField(20);
  addCopy = new JButton("Add Copy");
  removeCopy = new JButton("Remove Copy");
  staffIdQ = new JTextField(20);
  borrowerIdQ = new JTextField(20);
  copyIdQ = new JTextField(20);
  bookIdQ = new JTextField(20);
  authorQ = new JTextField(20);
  subjectQ = new JTextField(20);
  borrowCopy = new JButton("Borrow Copy");
  returnCopy = new JButton("Return Copy");
  findAuthor = new JButton("Find books by Author");
  findSubject = new JButton("Find books by Subject");
  what BorrowedBy = new JButton("Find copies borrowed by a borrower");
  who Borrowed = new JButton("Find who borrowed a copy last");

  uStatus = new JLabel("Result", SwingConstants.LEFT);
  cStatus = new JLabel("Result", SwingConstants.LEFT);
  sStatus = new JLabel("Result", SwingConstants.LEFT);

  // Listen to events from Convert button.
  addStaff.addActionListener(this);
  addBorrower.addActionListener(this);
  addCopy.addActionListener(this);
  removeCopy.addActionListener(this);
  borrowCopy.addActionListener(this);
  returnCopy.addActionListener(this);
  findAuthor.addActionListener(this);
  findSubject.addActionListener(this);
  whatBorrowedBy.addActionListener(this);
  whoBorrowed.addActionListener(this);

  // Add widgets to container.
  // Add to the User Panel
  userPanel.add(new JLabel("First Name:"));
  userPanel.add(firstName);
  userPanel.add(new JLabel("LastName:"));
  userPanel.add(lastName);
  userPanel.add(new JLabel("Date of Birth:"));
  userPanel.add(dateOfBirth);
  userPanel.add(new JLabel("User ID:"));
  userPanel.add(userId);
  userPanel.add(new JLabel("Salary:"));
  userPanel.add(salary);
  userPanel.add(new JLabel("Limit:"));
  userPanel.add(limit);
  userPanel.add(new JLabel("Lib ID:"));
  userPanel.add(libId);
  userPanel.add(new JLabel("Cost:"));
  userPanel.add(cost);
  userPanel.add(new JLabel("Section:"));
  userPanel.add(section);
  userPanel.add(new JLabel("ISBN:"));
  userPanel.add(isbn);
  userPanel.add(new JLabel("Book Title:"));
  userPanel.add(bookTitle);
  userPanel.add(new JLabel("Author:"));
  userPanel.add(author);
  userPanel.add(new JLabel("Subject:"));
  userPanel.add(subject);
  userPanel.add(new JLabel("Copy ID:"));
  userPanel.add(copyId);
  userPanel.add(new JLabel("Add Staff:"));
  userPanel.add(addStaff);
  userPanel.add(new JLabel("Add Borrower:"));
  userPanel.add(addBorrower);
  userPanel.add(new JLabel("Add Copy:"));
  userPanel.add(addCopy);
  userPanel.add(new JLabel("Remove Copy:"));
  userPanel.add(removeCopy);
  userPanel.add(new JLabel("Borrow Copy:"));
  userPanel.add(borrowCopy);
  userPanel.add(new JLabel("Return Copy:"));
  userPanel.add(returnCopy);
  userPanel.add(new JLabel("Find books by Author:"));
  userPanel.add(findAuthor);
  userPanel.add(new JLabel("Find books by Subject:"));
  userPanel.add(findSubject);
  userPanel.add(new JLabel("Find copies borrowed by a borrower:"));
  userPanel.add(whatBorrowedBy);
  userPanel.add(new JLabel("Find who borrowed a copy last:"));
  userPanel.add(whoBorrowed);
        }
userPanel.add(userId);
userPanel.add(new JLabel("Borrow Limit:"));
userPanel.add(limit);
userPanel.add(new JLabel("Staff Salary:"));
userPanel.add(salary);
userPanel.add(addStaff);
userPanel.add(addBorrower);
userPanel.add(uStatus);

// Add to the Catalog Panel
catalogPanel.add(new JLabel("Book ISBN:"));
catalogPanel.add(isbn);
catalogPanel.add(new JLabel("Book Title:"));
catalogPanel.add(bookTitle);
catalogPanel.add(new JLabel("Author(s):"));
catalogPanel.add(author);
catalogPanel.add(new JLabel("Subject(s):"));
catalogPanel.add(subject);
catalogPanel.add(new JLabel("Library Book ID:"));
catalogPanel.add(libId);
catalogPanel.add(new JLabel("Cost:"));
catalogPanel.add(cost);
catalogPanel.add(new JLabel("Section"));
catalogPanel.add(section);
catalogPanel.add(new JLabel("Library Copy ID"));
catalogPanel.add(copyId);
catalogPanel.add(addCopy);
catalogPanel.add(removeCopy);
catalogPanel.add(cStatus);

// Add to the Stock Panel
stockPanel.add(new JLabel("Operator ID:"));
stockPanel.add(staffIdQ);
stockPanel.add(new JLabel("Requestor ID:"));
stockPanel.add(borrowerIdQ);
stockPanel.add(new JLabel("Copy ID:"));
stockPanel.add(copyIdQ);
stockPanel.add(new JLabel("Book ID:"));
stockPanel.add(bookIdQ);
stockPanel.add(new JLabel("Author:"));
stockPanel.add(authorQ);
stockPanel.add(new JLabel("Subject"));
stockPanel.add(subjectQ);

stockPanel.add(borrowCopy);
stockPanel.add(returnCopy);
stockPanel.add(findAuthor);
stockPanel.add(findSubject);
stockPanel.add(whatBorrowedBy);
stockPanel.add(whoBorrowed);

stockPanel.add(sStatus);

uStatus.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
cStatus.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
sStatus.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
}

// Implementation of ActionListener interface.
public void actionPerformed(ActionEvent event)
{
    
    LibraryResultCode rslt=new LibraryResultCode();
    String output = "Ok";

    if (event.getSource() == addStaff)
{
    backend.addStaff (_eSystem._lString(firstName.getText()), 'a',
    _eSystem._lString(lastName.getText()), 'a',
    _eSystem._lString(dateOfBirth.getText()), 'a',
    _eSystem._lString(userId.getText()), 'a',
    _eSystem._lString(salary.getText()), 'a',
    rslt);
    setResult(rslt.value,output,uStatus);
}
else if (event.getSource() == addBorrower)
{
    backend.addBorrower(_eSystem._lString(firstName.getText()), 'a',
    _eSystem._lString(lastName.getText()), 'a',
    _eSystem._lString(dateOfBirth.getText()),'a',
    _eSystem._lString(userId.getText()), 'a',
    _eSystem._lString(limit.getText()), 'a',
    rslt);
    setResult(rslt.value,output,uStatus);
}
else if (event.getSource() == addCopy)
{
    backend.addLibraryItem (_eSystem._lString(copyId.getText()), 'a',
    _eSystem._lString(libId.getText()), 'a',
    _eSystem._lString(cost.getText()), 'a',
    _eSystem._lString(section.getText()), 'a',
    _eSystem._lString(isbn.getText()), 'a',
    _eSystem._lString(bookTitle.getText()),'a',
    _eSystem._lString(author.getText()), 'a',
    _eSystem._lString(subject.getText()), 'a',
    rslt);
    setResult(rslt.value,output,cStatus);
}
else if (event.getSource() == removeCopy)
{
    backend.removeLibraryItem (_eSystem._lString(copyId.getText()),'a',
    rslt);
    setResult(rslt.value,output,cStatus);
}
else if (event.getSource() == borrowCopy)
{
    backend.borrowLibraryItem (_eSystem._lString(copyIdQ.getText()), 'a',
    _eSystem._lString(borrowerIdQ.getText())),'a',
    rslt);
    setResult(rslt.value,output,sStatus);
}
else if (event.getSource() == returnCopy)
{
    backend.returnLibraryItem (_eSystem._lString(copyIdQ.getText()),'a',
    rslt);
    setResult(rslt.value,output,sStatus);
}
else if (event.getSource() == findAuthor)
{
    _eWrapper_eAny books = new _eWrapper_eAny();
    backend.findItemsByAuthor (_eSystem._lString(authorQ.getText()),'a',
    books, new _eSeq(), 'a',
    rslt);
    output = _eSystem._lJavaString(_eSeq)books.value);
    setResult(rslt.value,output,sStatus);
}
else if (event.getSource() == findSubject)
{
    _eWrapper_eAny books = new _eWrapper_eAny();
    backend.findItemsBySubject (_eSystem._lString(subjectQ.getText()),'a',
    rslt);
    output = _eSystem._lJavaString(_eSeq)books.value);
    setResult(rslt.value,output,sStatus);
}
books, new _eSeq(),
    output = _eSystem._lJavaString((_eSeq)books.value);
    setResult(rslt.value,output,sStatus);
}
else if (event.getSource() == whatBorrowedBy)
{
}
else if (event.getSource() == whoBorrowed)
{
} else sStatus.setText("Unknown event or return code");

public void setResult(int rslt, String value, JLabel status)
{
    if (rslt == LibraryResultCode.success)
    {
        status.setText(value+"Successful operation");
    }
    else if (rslt == LibraryResultCode.unauthorized)
    {
        status.setText(value+"Unauthorized Requestor");
    }
    else if (rslt == LibraryResultCode.unregistered)
    {
        status.setText(value+"Unregistered borrower");
    }
    else if (rslt == LibraryResultCode.notOwned)
    {
        status.setText(value+"Copy not owned by Library");
    }
    else if (rslt == LibraryResultCode.notAvailable)
    {
        status.setText(value+"Copy checked out");
    }
    else if (rslt == LibraryResultCode.available)
    {
        status.setText(value+"Copy available");
    }
    else if (rslt == LibraryResultCode.unknownAuthor)
    {
        status.setText("No books by this author");
    }
    else if (rslt == LibraryResultCode.unknownSubject)
    {
        status.setText("No books on this subject");
    }
    else if (rslt == LibraryResultCode.maxLimits)
    {
        status.setText(value+"User has reached their borrowing limit");
    }
    else if (rslt == LibraryResultCode.duplicateCopy)
    {
        status.setText(value+ "Copy already exists in Library");
    }
    else if (rslt == LibraryResultCode.duplicateBook)
    {
        status.setText(value+ "Book already exists in Library");
    }
    else if (rslt == LibraryResultCode.noCopy)
    {
        status.setText(value+ "Copy is not in Library Database");
    }
    else if (rslt == LibraryResultCode.knownUser)
private boolean isInteger(String input){
    boolean b=input.length()>0;
    if(b){
        for(int i=0;i<input.length();i++)
            b=b&&(input.charAt(i)>='0' && input.charAt(i)<='9');
    }
    return b;
}
// main method
public static void main(String jargs[])
{
    String pathToClass = System.getProperty("user.dir");
    if (pathToClass.length() != 0 &&
        pathToClass.charAt(pathToClass.length() - 1) != File.separatorChar)
    {
        pathToClass = pathToClass + File.separatorChar;
    }
    Environment context = new Environment(pathToClass);

    try{
        UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
    } catch(Exception e) {} 
    LibraryAccess applicationObjectb = new LibraryAccess(context);
}

// End

H.8 LibraryBookDescription.java
package oolibrary;

import Ertsys.*;

// Packages imported
import oolibrary.*;

public class LibraryBookDescription extends BookDescription
{
    protected int libraryId;
    protected int cost;
    protected _eSeq section;
    public LibraryBookDescription (int _vlibraryId, int _vcost, _eSeq _vsection, char _t0_vsection, int isb, _eSeq titl, char _t0titl, _eSet auth, Author _t0auth, _eSet subj, Subject _t0subj)
public int getId ()
{
    return libraryId;
}

public _eSeq _rtoString ()
{
    return myName () _oPlus (_eSystem _lString ("","",(_eTemplate_0) null)).
            _oPlusPlus (_eSystem _ltoString (libraryId), (_eTemplate_0) null);
}

public _eSeq getSection ()
{
    return section;
}

public boolean _lEqual (LibraryBookDescription _vArg_12_9)
{
    if (this == _vArg_12_9) return true;
    if (!super._lEqual (_vArg_12_9)) return false;
    return (((_vArg_12_9.cost == cost) && _vArg_12_9.section._lEqual (section)) && (_vArg_12_9.libraryId == libraryId));
}

public boolean equals (_eAny _lArg)
{
    return _lArg == this || (_lArg != null && _lArg.getClass () == LibraryBookDescription.class && _lEqual ((LibraryBookDescription) _lArg));
}

} // End of file.

H.9 LibraryCatalog.java
package oolibrary;

import Ertsys.*;

// Packages imported
import oolibrary.*;

public class LibraryCatalog extends _eAny
{
    public LibraryCatalogDescription (int _vlibraryId, int _vcost, _eSeq _vsection, char _t0_vsection, int isb, _eSeq titl, char _t0titl)
    {
        super (isb, titl, (char) 0);
        libraryId = _vlibraryId;
        cost = _vcost;
        section = _vsection;
    }

    public _eSeq _toString ()
    {
        return myName () _oPlus (_eSystem _lString ("","",(_eTemplate_0) null)).
                _oPlusPlus (_eSystem _ltoString (libraryId), (_eTemplate_0) null);
    }

    public _eSeq getSection ()
    {
        return section;
    }

    public boolean _lEqual (LibraryBookDescription _vArg_12_9)
    {
        if (this == _vArg_12_9) return true;
        if (!super._lEqual (_vArg_12_9)) return false;
        return (((_vArg_12_9.cost == cost) && _vArg_12_9.section._lEqual (section)) && (_vArg_12_9.libraryId == libraryId));
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () ==
                        LibraryBookDescription.class && _lEqual ((LibraryBookDescription) _lArg));
    }

} // End of file.
if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
  _eSystem.currentCheckNesting ++;
  try
  {
    boolean _vQuantifierResult_16_15;
    {
      _vQuantifierResult_16_15 = true;
      int _vCaptureCount_item1_16_35 = allLibraryItems ()._oHash ();
      int _vLoopCounter_16_22 = 0;
      for (;;)
      {
        if (((_vLoopCounter_16_22 == _vCaptureCount_item1_16_35) || (!
            _vQuantifierResult_16_15))) break;
        boolean _vQuantifierResult_16_28;
        {
          _vQuantifierResult_16_28 = true;
          int _vCaptureCount_item2_16_35 = allLibraryItems ()._oHash ();
          int _vLoopCounter_16_28 = 0;
          for (;;)
          {
            if (((_vLoopCounter_16_28 == _vCaptureCount_item2_16_35) || (!
                _vQuantifierResult_16_28))) break;
            boolean _vQuantifierResult_16_28;
            {
              _vQuantifierResult_16_28 = (!(!((LibraryItem)
                  allLibraryItems ()._oIndex (_vLoopCounter_16_22)).
                _lEqual (((LibraryItem) allLibraryItems ()
                    ._oIndex (_vLoopCounter_16_28)))) || (!((LibraryItem)
                  allLibraryItems () ._oIndex (_vLoopCounter_16_28)).
                _getId () == ((LibraryItem) allLibraryItems ()
                    ._oIndex (_vLoopCounter_16_28)). _getId ()))
            if (!(_vQuantifierResult_16_28))
              { }
            else
              { _vLoopCounter_16_28 = _eSystem._oSucc (_vLoopCounter_16_28); }
          }
        }
        _vQuantifierResult_16_15 = _vQuantifierResult_16_28;
        if (!(_vQuantifierResult_16_15))
          {
          }
        else
          { _vLoopCounter_16_22 = _eSystem._oSucc (_vLoopCounter_16_22); }
      }
    }
    if (!(_vQuantifierResult_16_15)) throw new _xClassInvariantItem ("LibraryCatalog.pd:16,15", _lArg);
  }
  catch (_xCannotEvaluate _lException)
  {
  }
  _eSystem.currentCheckNesting --;
}
if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
  _eSystem.currentCheckNesting ++;
  try
if (!((alltheBooks ()._oHash () <= allLibraryItems ()._oHash ())))) throw
new _xClassInvariantItem ("LibraryCatalog.pd:17,28", _lArg);
}
catch (_xCannotEvaluate _lException)
{
}
_eSystem.currentCheckNesting --;
}
if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
maxCheckNesting)
{
_eSystem.currentCheckNesting ++;
try
{
boolean _vQuantifierResult_18_15;
{
_eSet _vCaptureBound_book_18_28 = alltheBooks ();
_vQuantifierResult_18_15 = true;
int _vCaptureCount_book_18_28 = _vCaptureBound_book_18_28._oHash ();
int _vLoopCounter_18_22 = 0;
for (;;)
{
if (((_vLoopCounter_18_22 == _vCaptureCount_book_18_28) || (!
_vQuantifierResult_18_15))) break;
_eSet _vChoose_18_44 = null;
{
_vChoose_18_44 = new _eSet ();
int _vCaptureCount_copy_18_57 = allLibraryItems ()._oHash ();
int _vLoopCounter_18_51 = 0;
for (;;)
{
if ((_vLoopCounter_18_51 == _vCaptureCount_copy_18_57))
break;
._oIndex (_vLoopCounter_18_22))._lEqual (((LibraryItem)
allLibraryItems ()._oIndex (_vLoopCounter_18_51)).
getBook ()))
{
_vChoose_18_44 = _vChoose_18_44.append (allLibraryItems ()
._oIndex (_vLoopCounter_18_51));
}
else
{
}
_vLoopCounter_18_51 = _eSystem._oSucc (_vLoopCounter_18_51
);
}
}
_vQuantifierResult_18_15 = (_vChoose_18_44._oHash () <= 1000);
if (!(_vQuantifierResult_18_15))
{
} else
{
_vLoopCounter_18_22 = _eSystem._oSucc (_vLoopCounter_18_22);
}
}
if (!(_vQuantifierResult_18_15)) throw new _xClassInvariantItem ("LibraryCatalog.pd:18,15", _lArg);
}
catch (_xCannotEvaluate _lException)
{
}
_eSystem.currentCheckNesting --;

}

}  

void _lClassInvariantCheck (String _lArg)
{
    _lc_LibraryCatalog (_lArg);
}

private _eSet allLibraryItems()
{
    return allLibItems.ran();
}

protected _eSet alltheBooks()
{
    _eSet _vForYield_14_12;
    {
        _vForYield_14_12 = new _eSet();
        int _vCaptureCount_item_14_22 = allLibraryItems()._oHash();
        int _vLoopCounter_14_16 = 0;
        for (;;)
        {
            if (_vLoopCounter_14_16 == _vCaptureCount_item_14_22) break;
            _vForYield_14_12 = _vForYield_14_12.append(((eAny)((LibraryItem)_vCaptureCount_item_14_22).getBook()));
            _vLoopCounter_14_16 = _eSystem._oSucc(_vLoopCounter_14_16);
        }
    }
    return _vForYield_14_12;
}

private _n1_HashedBucket allLibItems;
public LibraryCatalog()
{
    super();
    allLibItems = new _n1_HashedBucket(1000);
    _lc_LibraryCatalog("LibraryCatalog.pd:34,9");
}

public boolean newItemId(int id)
{
    boolean _vQuantifierResult_37_12;
    {
        _eSet _vCaptureBound_items_37_26 = allKnownItems();
        _vQuantifierResult_37_12 = true;
        int _vCaptureCount_items_37_26 = _vCaptureBound_items_37_26._oHash();
        int _vLoopCounter_37_19 = 0;
        for (;;)
        {
            if (_vLoopCounter_37_19 == _vCaptureCount_items_37_26) break;
            _vQuantifierResult_37_12 = (!((_vQuantifierResult_37_12))) break;
            _vCaptureBound_items_37_26 = ((LibraryItem)_vCaptureBound_items_37_26._oIndex(_vLoopCounter_37_19)).getId();
            if (_vLoopCounter_37_19 == _vQuantifierResult_37_12)
            {
            }
            else
            {
                _vLoopCounter_37_19 = _eSystem._oSucc(_vLoopCounter_37_19);
            }
        }
    }
    return _vQuantifierResult_37_12;
}
public boolean newBook (LibraryBookDescription b)
{
    return allKnownBooks ()._ovIn (((_eAny) b));
}

public _eSet allKnownItems ()
{
    return allLibItems.ran ();
}

public _eSet allKnownBooks ()
{
    _eSet _vForYield_61_19;
    {
        _eSet _vCaptureBound_item_61_29 = allKnownItems ();
        _vForYield_61_19 = new _eSet ();
        int _vCaptureCount_item_61_29 = _vCaptureBound_item_61_29._oHash ();
        int _vLoopCounter_61_23 = 0;
        for (;;)
        {
            if ((_vLoopCounter_61_23 == _vCaptureCount_item_61_29)) break;
            _vForYield_61_19 = _vForYield_61_19.append (((_eAny) ((LibraryItem) _vCaptureBound_item_61_29._oIndex (_vLoopCounter_61_23)).getBook ()));
            _vLoopCounter_61_23 = _eSystem._oSucc (_vLoopCounter_61_23);
        }
    }
    return _vForYield_61_19;
}

public _eSet allKnownItemIds ()
{
    _eSet _vForYield_55_12;
    {
        _eSet _vCaptureBound_item_55_22 = allKnownItems ();
        _vForYield_55_12 = new _eSet ();
        int _vCaptureCount_item_55_22 = _vCaptureBound_item_55_22._oHash ();
        int _vLoopCounter_55_16 = 0;
        for (;;)
        {
            if ((_vLoopCounter_55_16 == _vCaptureCount_item_55_22)) break;
            _vForYield_55_12 = _vForYield_55_12.append (((_eAny) new _eWrapper_int (((LibraryItem) _vCaptureBound_item_55_22._oIndex (_vLoopCounter_55_16)).getId ())));
            _vLoopCounter_55_16 = _eSystem._oSucc (_vLoopCounter_55_16);
        }
    }
    return _vForYield_55_12;
}

public _eSet allKnownBookIds ()
{
    _eSet _vForYield_58_12;
    {
        _eSet _vCaptureBound_book_58_22 = allKnownBooks ();
        _vForYield_58_12 = new _eSet ();
        int _vCaptureCount_book_58_22 = _vCaptureBound_book_58_22._oHash ();
        int _vLoopCounter_58_16 = 0;
        for (;;)
        {
            if ((_vLoopCounter_58_16 == _vCaptureCount_book_58_22)) break;
            _vForYield_58_12 = _vForYield_58_12.append (((_eAny) new _eWrapper_int (((LibraryBookDescription) _vCaptureBound_book_58_22._oIndex (_vLoopCounter_58_16)).getId ())));
            _vLoopCounter_58_16 = _eSystem._oSucc (_vLoopCounter_58_16);
        }
    }
}
public boolean hasItem (LibraryItem copy)
{
    return allLibItems._ovIn (((_eAny) copy));
}

public boolean uniqueItemId (int id)
{
    return (!allKnownItemIds ()._ovIn (((_eAny) new _eWrapper_int (id))));
}

public void addLibraryItem (LibraryItem copy)
{
    if (_eSystem.enablePre & _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(hasItem (copy))) throw new _xPre ("LibraryCatalog.pd:70,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre & _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(uniqueItemId (copy.getId ()))) throw new _xPre ("LibraryCatalog.pd:71,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    _n1_HashedBucket _vUnshare_74_13 = ((_n1_HashedBucket) allLibItems._lClone ());
    allLibItems = _vUnshare_74_13;
    _vUnshare_74_13.add (((_eAny) copy));
    _lClassInvariantCheck ("LibraryCatalog.pd:75,9");
}

public void removeLibraryItem (LibraryItem copy)
{
    if (_eSystem.enablePre & _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(hasItem (copy))) throw new _xPre ("LibraryCatalog.pd:78,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
n1_HashedBucket _vUnshare_80_13 = ((n1_HashedBucket) allLibItems._lClone ());
allLibItems = _vUnshare_80_13;
_vUnshare_80_13.remove ((_eAny) copy);
_lClassInvariantCheck ("LibraryCatalog.pd:81,9");
}

public boolean _lEqual (LibraryCatalog _vArg_11_9)
{
    if (this == _vArg_11_9) return true;
    return _vArg_11_9.allLibraryItems ()._lEqual (allLibraryItems ());
}

public boolean equals (_eAny _lArg)
{
    return _lArg == this || (_lArg != null && _lArg.getClass () == LibraryCatalog.
        class && _lEqual ((LibraryCatalog) _lArg));
}

// End of file.

H.10 LibraryDB.java

package oolibrary;

import Ertsys.*;

// Packages imported
import oolibrary.*;

public class LibraryDB extends _eAny
{
    protected LibraryStock books;
    protected UserBase users;
    protected _eSeq today;
    public LibraryDB (Environment context)
    {
        super ();
        today = context.getCurrentDateTime ()._rtoString ();
        books = new LibraryStock ();
        users = new UserBase ();
    }

    public boolean isNumber (_eSeq val, char _t0val)
    {
        boolean _vQuantifierResult_23_12;
        {
            _vQuantifierResult_23_12 = true;
            int _vCaptureCount_i_23_26 = val._oHash ();
            int _vLoopCounter_23_19 = 0;
            for (;;)
            {
                if (((_vCaptureCount_i_23_26 <= _vLoopCounter_23_19) || (!
                    _vQuantifierResult_23_12))) break;
                _vQuantifierResult_23_12 = (_eSystem._lisDigit (((_eWrapper_char) val.
                    _oIndex (_vLoopCounter_23_19)).value) && (0 < val._oHash ()));
                if (((!_vQuantifierResult_23_12))
                {
                }
                else
                {
}
_vLoopCounter_23_19 = _eSystem._oSucc (_vLoopCounter_23_19);
}
}
return _vQuantifierResult_23_12;
}

public int getNumber (_eSeq val, char _t0val)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (val, (char) 0))) throw new _xPre ("LibraryDB.pd:26,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    return _eSystem._lstringToNat (val, (char) 0);
}

public _eSeq firstComma (_eSeq s, char _t0s)
{
    int _vLet_n_31_14 = s.findFirst (((_eAny) new _eWrapper_char (',')));
    return ((_vLet_n_31_14 < 0) ?
        s :
        s.take ((1 + _vLet_n_31_14)));
}

public _eSet makeAuthorSet (_eSeq s, char _t0s)
{
    _eSeq _vLet_stripped_38_18 = s;
    if (_vLet_stripped_38_18.empty ()
    {
        return new _eSet ();
    }
    else
    {
        _eSeq _vLet_a_41_20 = firstComma (_vLet_stripped_38_18, (char) 0);
        return makeAuthorSet (_vLet_stripped_38_18.drop (_vLet_a_41_20._oHash ()), (char) 0).append (((_eAny) new Author (_vLet_a_41_20, (char) 0));
    }
}

public _eSet makeSubjectSet (_eSeq s, char _t0s)
{
    _eSeq _vLet_stripped_49_18 = s;
    if (_vLet_stripped_49_18.empty ()
    {
        return new _eSet ();
    }
    else
    {
        _eSeq _vLet_su_52_20 = firstComma (_vLet_stripped_49_18, (char) 0);
        return makeSubjectSet (_vLet_stripped_49_18.drop (_vLet_su_52_20._oHash ()), (char) 0).append (((_eAny) new Subject (_vLet_su_52_20, (char) 0));
    }
}

public LibraryBookDescription makeLibraryBook (_eSeq libID, char _t0libID, _eSeq cost, char _t0cost, _eSeq section, char _t0section, _eSeq isbn, char _t0isbn, _eSeq
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
  _eSystem.currentCheckNesting ++;
  try
  {
    if (!(isNumber (libID, (char) 0))) throw new _xPre ("LibraryDB.pd:59,13");
    catch (_xCannotEvaluate _lException)
    {
    }
    _eSystem.currentCheckNesting --;
  }
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
  _eSystem.currentCheckNesting ++;
  try
  {
    if (!(isNumber (cost, (char) 0))) throw new _xPre ("LibraryDB.pd:59,30");
    catch (_xCannotEvaluate _lException)
    {
    }
    _eSystem.currentCheckNesting --;
  }
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
  _eSystem.currentCheckNesting ++;
  try
  {
    if (!(isNumber (isbn, (char) 0))) throw new _xPre ("LibraryDB.pd:59,46");
    catch (_xCannotEvaluate _lException)
    {
    }
    _eSystem.currentCheckNesting --;
  }
  _eSet _vLet_authset_60_16 = makeAuthorSet (authors, (char) 0);
  _eSet _vLet_subjset_60_53 = makeSubjectSet (subjects, (char) 0);
  return new LibraryBookDescription (getNumber (libID, (char) 0), getNumber (cost, (char) 0),
                                       section, (char) 0, getNumber (isbn, (char) 0), title, (char) 0,
                                       _vLet_authset_60_16, (Author) null, _vLet_subjset_60_53, (Subject) null);
}

public Staff makeStaffMember (_eSeq first, char _t0first, _eSeq last, char _t0last,
                            _eSeq doB, char _t0doB, _eSeq id, char _t0id, _eSeq salary, char _t0salary)
{
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
  _eSystem.currentCheckNesting ++;
  try
  {
    if (!(isNumber (id, (char) 0))) throw new _xPre ("LibraryDB.pd:65,13");
    catch (_xCannotEvaluate _lException)
    {
    }
    _eSystem.currentCheckNesting --;
  }
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
  _eSystem.currentCheckNesting ++;
  try
  {
    if (!(isNumber (salary, (char) 0))) throw new _xPre ("LibraryDB.pd:65,26");
  }
  catch (_xCannotEvaluate _lException)
  {
  }
  _eSystem.currentCheckNesting --;
}
return new Staff (first, (char) 0, last, (char) 0, doB, (char) 0, getNumber (id, (char) 0),
                      getNumber (salary, (char) 0));

public Borrower makeBorrowerMember (_eSeq first, char _t0first, _eSeq last, char
                                      _t0last, _eSeq doB, char _t0doB, _eSeq id, char _t0id, _eSeq limit, char _t0limit)
{
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
  _eSystem.currentCheckNesting ++;
  try
  {
    if (!(isNumber (id, (char) 0))) throw new _xPre ("LibraryDB.pd:69,13");
  }
  catch (_xCannotEvaluate _lException)
  {
  }
  _eSystem.currentCheckNesting --;
}
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
  _eSystem.currentCheckNesting ++;
  try
  {
    if (!(isNumber (limit, (char) 0))) throw new _xPre ("LibraryDB.pd:69,26");
  }
  catch (_xCannotEvaluate _lException)
  {
  }
  _eSystem.currentCheckNesting --;
}
return new Borrower (first, (char) 0, last, (char) 0, doB, (char) 0, getNumber (id,
                          (char) 0), getNumber (limit, (char) 0));

public LibraryItem makeLibraryItem (_eSeq id, char _t0id, _eSeq libID, char _t0libID,
                                        _eSeq cost, char _t0cost, _eSeq section, char _t0section,
                                        _eSeq isbn, char _t0isbn , _eSeq title, char _t0title, _eSeq authors, char _t0authors, _eSeq subjects, char
                                        _t0subjects)
{
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
  _eSystem.currentCheckNesting ++;
  try
  {
    if (!(isNumber (id, (char) 0))) throw new _xPre ("LibraryDB.pd:73,13");
  }
  catch (_xCannotEvaluate _lException)
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
    _eSystem.currentCheckNesting ++;
    try
    {
        if (!(isNumber (libID, (char) 0))) throw new _xPre ("LibraryDB.pd:73,26");
    }
    catch (_xCannotEvaluate _lException)
    {
    }
    _eSystem.currentCheckNesting --;
}
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
    _eSystem.currentCheckNesting ++;
    try
    {
        if (!(isNumber (cost, (char) 0))) throw new _xPre ("LibraryDB.pd:73,43");
    }
    catch (_xCannotEvaluate _lException)
    {
    }
    _eSystem.currentCheckNesting --;
}
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
    _eSystem.currentCheckNesting ++;
    try
    {
        if (!(isNumber (isbn, (char) 0))) throw new _xPre ("LibraryDB.pd:73,59");
    }
    catch (_xCannotEvaluate _lException)
    {
    }
    _eSystem.currentCheckNesting --;
}
LibraryBookDescription _vLet_libBook_74_17 = makeLibraryBook (libID, (char) 0, cost, (char) 0, section, (char) 0, isbn, (char) 0, title, (char) 0, authors, (char) 0, subjects, (char) 0);
return new LibraryItem (getNumber (id, (char) 0), today, (char) 0, _vLet_libBook_74_17);

public LibraryItem makeLibraryItem (_eSeq id, char _t0id, LibraryBookDescription libBook)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!isNumber (id, (char) 0))) throw new _xPre ("LibraryDB.pd:79,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
public void addStaff (_eSeq first, char _t0first, _eSeq last, char _t0last, _eSeq doB,
char _t0doB, _eSeq id, char _t0id, _eSeq salary, char _t0salary, LibraryResultCode rslt)
{
if (((!isNumber (id, (char) 0)) || (!isNumber (salary, (char) 0))))
{
rslt.value = LibraryResultCode.incorrectInput;
}
else
{
    Staff _vLet_s_85_23 = makeStaffMember (first, (char) 0, last, (char) 0, doB, (char) 0, id, (char) 0, salary, (char) 0);
    if (users.uniqueId (_vLet_s_85_23))
    {
        UserBase _vUnshare_86_37 = ((UserBase) users._lClone ());
        users = _vUnshare_86_37;
        _vUnshare_86_37.addStaff (_vLet_s_85_23);
        rslt.value = LibraryResultCode.success;
    }
    else
    {
        rslt.value = LibraryResultCode.knownUser;
    }
}
}

public void addBorrower (_eSeq first, char _t0first, _eSeq last, char _t0last, _eSeq doB,
char _t0doB, _eSeq id, char _t0id, _eSeq limit, char _t0limit, LibraryResultCode rslt)
{
if (((!isNumber (id, (char) 0)) || (!isNumber (limit, (char) 0))))
{
rslt.value = LibraryResultCode.incorrectInput;
}
else
{
    Borrower _vLet_b_95_22 = makeBorrowerMember (first, (char) 0, last, (char) 0, doB, (char) 0, id, (char) 0, limit, (char) 0);
    if (users.uniqueId (_vLet_b_95_22))
    {
        UserBase _vUnshare_96_37 = ((UserBase) users._lClone ());
        users = _vUnshare_96_37;
        _vUnshare_96_37.addBorrower (_vLet_b_95_22);
        rslt.value = LibraryResultCode.success;
    }
    else
    {
        rslt.value = LibraryResultCode.knownUser;
    }
}
}

public void nextDay (_eSeq next, char _t0next)
{
today = next;
}

public void addLibraryItem (_eSeq id, char _t0id, _eSeq libID, char _t0libID, _eSeq cost,
char _t0cost, _eSeq section, char _t0section, _eSeq isbn, char _t0isbn,
_eSeq title, char _t0title, _eSeq authors, char _t0authors, _eSeq subjects, char _t0subjects, LibraryResultCode rslt)
if ((((!isNumber (id, (char) 0)) || (!isNumber (libID, (char) 0))) || (!isNumber (cost, (char) 0))) || (!isNumber (isbn, (char) 0)))
{
    rslt.value = LibraryResultCode.incorrectInput;
} else
{
    LibraryItem _vLet_newCopy_109_22 = makeLibraryItem (id, (char) 0, libID, (char) 0, cost, (char) 0, section, (char) 0, isbn, (char) 0, title, (char) 0, authors, (char) 0, subjects, (char) 0);
    if (books.hasItem (_vLet_newCopy_109_22))
    {
        rslt.value = LibraryResultCode.duplicateCopy;
    } else
    {
        LibraryStock _vUnshare_111_21 = ((LibraryStock) books._lClone ());
        books = _vUnshare_111_21;
        _vUnshare_111_21.addLibraryItem (_vLet_newCopy_109_22);
        rslt.value = LibraryResultCode.success;
    }
}
}

public void removeLibraryItem (_eSeq id, char _t0id, LibraryResultCode rslt)
{
    if (((!isNumber (id, (char) 0))))
    {
        rslt.value = LibraryResultCode.incorrectInput;
    } else
    {
        int _vLet_copyId_119_22 = getNumber (id, (char) 0);
        if (((!books.hasId (_vLet_copyId_119_22)))
        {
            rslt.value = LibraryResultCode.notOwned;
        } else
        {
            LibraryItem _vLet_newCopy_121_27 = books.findItem (_vLet_copyId_119_22);
            if (((!books.itemAvailable (_vLet_newCopy_121_27)))
            {
                rslt.value = LibraryResultCode.notAvailable;
            } else
            {
                LibraryStock _vUnshare_123_26 = ((LibraryStock) books._lClone ());
                books = _vUnshare_123_26;
                _vUnshare_123_26.removeLibraryItem (_vLet_newCopy_121_27);
                rslt.value = LibraryResultCode.success;
            }
        }
    }
}

public void borrowLibraryItem (_eSeq itemId, char _t0itemId, _eSeq borrowerId, char _t0borrowerId, LibraryResultCode rslt)
{
    if ((((!isNumber (itemId, (char) 0)) || (!isNumber (borrowerId, (char) 0))))
    {
        rslt.value = LibraryResultCode.incorrectInput;
    } else
    {
int _vLet_copyId_132_22 = getNumber (itemId, (char) 0);
int _vLet_borrId_132_53 = getNumber (borrowerId, (char) 0);
if ((!books.hasId (_vLet_copyId_132_22))
    { rslt.value = LibraryResultCode.notOwned; }
else if ((!users.hasId (_vLet_borrId_132_53))
    { rslt.value = LibraryResultCode.unregistered; }
else
    { LibraryItem _vLet_copy_135_26 = books.findItem (_vLet_copyId_132_22);
      Borrower _vLet_borrower_135_60 = users.findUser (_vLet_borrId_132_53);
      if ((!books.itemAvailable (_vLet_copy_135_26))
        { rslt.value = LibraryResultCode.notAvailable; }
      else if (!books.withinLimits (_vLet_borrower_135_60))
        { rslt.value = LibraryResultCode.maxLimits; }
      else
        { LibraryStock _vUnshare_138_26 = ((LibraryStock) books._lClone ());
          books = _vUnshare_138_26;
          _vUnshare_138_26.borrowLibraryItem (_vLet_copy_135_26,
          _vLet_borrower_135_60, today, (char) 0);
          rslt.value = LibraryResultCode.success; }
    }
}

public void returnLibraryItem (_eSeq itemId, char _t0itemId, LibraryResultCode rslt)
{
    if (!isNumber (itemId, (char) 0))
        { rslt.value = LibraryResultCode.incorrectInput; }
    else
        { int _vLet_copyId_147_22 = getNumber (itemId, (char) 0);
          if (!books.hasId (_vLet_copyId_147_22))
            { rslt.value = LibraryResultCode.notOwned; }
          else
            { LibraryItem _vLet_copy_149_26 = books.findItem (_vLet_copyId_147_22);
              if (books.itemAvailable (_vLet_copy_149_26))
                { rslt.value = LibraryResultCode.available; }
              else
                { LibraryStock _vUnshare_151_26 = ((LibraryStock) books._lClone ());
                  books = _vUnshare_151_26;
                  _vUnshare_151_26.returnLibraryItem (_vLet_copy_149_26);
                  rslt.value = LibraryResultCode.success; }
            }
        }
}
public void findItemsByAuthor (_eSeq author, char _t0author, _eWrapper__eAny booksList , _eSeq _t0booksList, char _t1booksList, LibraryResultCode rslt)
{
  _eSet _vLet_authset_158_20 = makeAuthorSet (author, (char) 0);
  if (_vLet_authset_158_20.empty ()
  {
    booksList.value = _eSystem._lString ("No Author");
    rslt.value = LibraryResultCode.incorrectInput;
  }
  else
  {
    _eSet _vLet_bookset_161_22 = books.findBookByAuthor (_vLet_authset_158_20, (Author) null);
    booksList.value = _vLet_bookset_161_22._rtoString ();
    rslt.value = LibraryResultCode.success;
  }
}

public void findItemsBySubject (_eSeq subject, char _t0subject, _eWrapper__eAny booksList, _eSeq _t0booksList, char _t1booksList, LibraryResultCode rslt)
{
  _eSet _vLet_subjset_168_20 = makeSubjectSet (subject, (char) 0);
  if (_vLet_subjset_168_20.empty ()
  {
    booksList.value = _eSystem._lString ("No Subject");
    rslt.value = LibraryResultCode.incorrectInput;
  }
  else
  {
    _eSet _vLet_bookset_171_22 = books.findBookBySubject (_vLet_subjset_168_20, (Subject) null);
    booksList.value = _vLet_bookset_171_22._rtoString ();
    rslt.value = LibraryResultCode.success;
  }
}

public boolean _lEqual (LibraryDB _vArg_11_9)
{
  if (this == _vArg_11_9) return true;
  return ((_vArg_11_9.users._lEqual (users) && _vArg_11_9.today._lEqual (today)) &&
    _vArg_11_9.books._lEqual (books));
}

public boolean equals (_eAny _lArg)
{
  return _lArg == this || (_lArg != null && _lArg.getClass () == LibraryDB.class &&
    _lEqual ((LibraryDB) _lArg));
}

// End of file.

H.11 LibraryItem.java
package oolibrary;

import Ertsys.*;

// Packages imported
import oolibrary.*;
public class LibraryItem extends _eAny {
    protected int copyId;
    protected _eSeq dateAcquired;
    protected LibraryBookDescription myLibraryBookDescription;
    public LibraryItem (int _vcopyId, _eSeq _vdateAcquired, char _t0_vdateAcquired,
        LibraryBookDescription _vmyLibraryBookDescription)
    {
        super ();
        if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!((_vcopyId <= 999))) throw new _xPre ("LibraryItem.pd:17,19");
            } catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
        copyId = _vcopyId;
        dateAcquired = _vdateAcquired;
        myLibraryBookDescription = _vmyLibraryBookDescription;
    }
    public int getId ()
    {
        return ((1000 * myLibraryBookDescription.getId ()) + copyId);
    }
    public LibraryBookDescription getBook ()
    {
        return myLibraryBookDescription;
    }
    public int hash ()
    {
        return getId ();
    }
    public boolean _lEqual (LibraryItem _vArg_11_9)
    {
        if (this == _vArg_11_9) return true;
            myLibraryBookDescription._lEqual (myLibraryBookDescription)) && (_vArg_11_9.
            copyId == copyId));
    }
    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == LibraryItem.class
            && _lEqual ((LibraryItem) _lArg));
    }
}

// End of file.

H.12 LibraryResultCode.java
package oolibrary;

176
import Ertsys.*;

// Packages imported
import oolibrary.*;

public final class LibraryResultCode extends _eEnumBase
{
  public LibraryResultCode ()
  {
  }

  public LibraryResultCode (int _lArg1)
  {
    super (_lArg1);
  }

  public static _eSeq _oRange (int _lArg1, int _lArg2)
  {
    _eSeq _lResult = new _eSeq ();
    while (_lArg1 <= _lArg2) _lResult = _lResult._lAppend (new LibraryResultCode (_lArg1 ++));
    return _lResult;
  }

  public static final int success = 0, unauthorized = 1, unregistered = 2, notOwned = 3,
    notAvailable = 4, available = 5, unknownAuthor = 6, unknownSubject = 7, maxLimits
    = 8, duplicateCopy = 9, duplicateBook = 10, noCopy = 11, knownUser = 12,
    neverBeenReturned = 13, incorrectInput = 14;

  private static final String _lnames [] =
  {
    "success", "unauthorized", "unregistered", "notOwned", "notAvailable", "available"
    , "unknownAuthor", "unknownSubject", "maxLimits", "duplicateCopy",
    "duplicateBook", "noCopy", "knownUser", "neverBeenReturned", "incorrectInput"
  };

  public static _eSeq _ltoString (int _larg)
  {
    return _eSystem._lString (_lnames [_larg]);
  }

  public _eSeq _rtoString ()
  {
    return _eSystem._lString (_lnames [value]);
  }
}

// End of file.

H.13 LibraryStock.java
package oolibrary;

import Ertsys.*;

// Packages imported
import oolibrary.*;

public class LibraryStock extends _eAny
protected LibraryCatalog catalog;
private _eSet currentlyBorrowed ()
{
    return borrowed.ran ();
}

private _n1_PriorityQueue borrowed;
public LibraryStock ()
{
    super ();
    catalog = new LibraryCatalog ();
    borrowed = new _n1_PriorityQueue ();
}

public _eSet checkedOut ()
{
    _eSet _vForYield_33_18;
    {
        _eSet _vCaptureBound_b_33_35 = borrowed.ran ();
        _vForYield_33_18 = new _eSet ();
        int _vCaptureCount_b_33_35 = _vCaptureBound_b_33_35._oHash ();
        int _vLoopCounter_33_23 = 0;
        for (;;)
        {
            if ((_vLoopCounter_33_23 == _vCaptureCount_b_33_35)) break;
            _vForYield_33_18 = _vForYield_33_18.append (((_eAny) ((Borrowing)
                _vCaptureBound_b_33_35._oIndex (_vLoopCounter_33_23)).getItem ()));
            _vLoopCounter_33_23 = _eSystem._oSucc (_vLoopCounter_33_23);
        }
    }
    return _vForYield_33_18;
}

public _eSet available ()
{
    return catalog.allKnownItems ()._oMinusMinus (checkedOut (), (_eTemplate_0) null);
}

public boolean itemAvailable (LibraryItem c)
{
    return available ()._ovIn (((_eAny) c));
}

public _eSet booksHeld ()
{
    return catalog.allKnownBooks ();
}

public boolean itemCurrentlyBorrowed (LibraryItem copy)
{
    return checkedOut ()._ovIn (((_eAny) copy));
}

public Borrowing copyBorrowing (LibraryItem copy)
{
    if (_eSystem.enablePre & _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(!itemCurrentlyBorrowed (copy))) throw new _xPre ("LibraryStock.pd:49,13");
        }
        catch (_xCannotEvaluate _lException)
Borrowing _vChoose_51_18 = null;
{
    _eSet _vCaptureBound_b_51_36 = borrowed.ran ();
    boolean _vSelectorCondition_51_18 = false;
    int _vCaptureCount_b_51_36 = _vCaptureBound_b_51_36._oHash ();
    int _vLoopCounter_51_24 = 0;
    for (;;)
    {
        if (((_vLoopCounter_51_24 == _vCaptureCount_b_51_36) ||
             _vSelectorCondition_51_18)) break;
        _vSelectorCondition_51_18 = ((Borrowing) _vCaptureBound_b_51_36._oIndex (_vLoopCounter_51_24)).getItem ()._lEqual (copy);
        if (_vSelectorCondition_51_18)
            _vChoose_51_18 = ((Borrowing) _vCaptureBound_b_51_36._oIndex (_vLoopCounter_51_24));
        else
            _vLoopCounter_51_24 = _eSystem._oSucc (_vLoopCounter_51_24);
    }
    if (_eSystem.enableThatOrAny && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(_vSelectorCondition_51_18)) throw new _xThatOrAny ("LibraryStock.pd:51,18");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    return _vChoose_51_18;
}

public boolean hasItem (LibraryItem copy)
{
    return catalog.hasItem (copy);
}

public boolean hasId (int id)
{
    return catalog.allKnownItemIds ()._ovIn (((_eAny) new _eWrapper_int (id)));
}

public boolean withinLimits (Borrower bor)
{
    return true;
}
public LibraryItem findItem (int id)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!hasId (id)) throw new _xPre ("LibraryStock.pd:64,13");
            catch (_xCannotEvaluate _lException)
            {
            }
        }
        _eSystem.currentCheckNesting --;
    }
    LibraryItem _vChoose_65_12 = null;
    {
        _eSet _vCaptureBound_copy_65_31 = catalog.allKnownItems ();
        boolean _vSelectorCondition_65_12;
        _vSelectorCondition_65_12 = false;
        int _vCaptureCount_copy_65_31 = _vCaptureBound_copy_65_31._oHash ();
        int _vLoopCounter_65_17 = 0;
        for (;;)
        {
            if (((_vLoopCounter_65_17 == _vCaptureCount_copy_65_31) ||
                 _vSelectorCondition_65_12)) break;
            _vSelectorCondition_65_12 = (((LibraryItem) _vCaptureBound_copy_65_31._oIndex (_vLoopCounter_65_17)).getId () == id);
            if (_vSelectorCondition_65_12)
            {
                _vChoose_65_12 = ((LibraryItem) _vCaptureBound_copy_65_31._oIndex (_vLoopCounter_65_17));
            }
            else
            {
            }
        }
    }
    if (_eSystem.enableThatOrAny && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(_vSelectorCondition_65_12)) throw new _xThatOrAny ("LibraryStock.pd:65,12");
            catch (_xCannotEvaluate _lException)
            {
            }
        }
        _eSystem.currentCheckNesting --;
    }
    return _vChoose_65_12;
}

public _eSet findBookByAuthor (_eSet aut, Author _t0aut)
{
_eSet _vChoose_68_12 = null;
{
    _eSet _vCaptureBound_b_68_21 = booksHeld ();
    _vChoose_68_12 = new _eSet ();
    int _vCaptureCount_b_68_21 = _vCaptureBound_b_68_21._oHash ();
    int _vLoopCounter_68_18 = 0;
    for (;;)
    {
        if (_vLoopCounter_68_18 == _vCaptureCount_b_68_21) break;
        if (aut._oLessLessEq (((LibraryBookDescription) _vCaptureBound_b_68_21.
            _oIndex (_vLoopCounter_68_18)).getAuthors (), (_eTemplate_0) null))
        {
            _vChoose_68_12 = _vChoose_68_12.append (_vCaptureBound_b_68_21._oIndex
                (_vLoopCounter_68_18));
        } else
        {
        }
        _vLoopCounter_68_18 = _eSystem._oSucc (_vLoopCounter_68_18);
    }
}
return _vChoose_68_12;
}

public _eSet findBookBySubject (_eSet sub, Subject _t0sub)
{
    _eSet _vChoose_71_12 = null;
    {
        _eSet _vCaptureBound_b_71_21 = booksHeld ();
        _vChoose_71_12 = new _eSet ();
        int _vCaptureCount_b_71_21 = _vCaptureBound_b_71_21._oHash ();
        int _vLoopCounter_71_18 = 0;
        for (;;)
        {
            if (_vLoopCounter_71_18 == _vCaptureCount_b_71_21) break;
            if (sub._oLessLessEq (((LibraryBookDescription) _vCaptureBound_b_71_21.
                _oIndex (_vLoopCounter_71_18)).getSubjects (), (_eTemplate_0) null))
            {
                _vChoose_71_12 = _vChoose_71_12.append (_vCaptureBound_b_71_21._oIndex
                    (_vLoopCounter_71_18));
            } else
            {
            }
            _vLoopCounter_71_18 = _eSystem._oSucc (_vLoopCounter_71_18);
        }
    }
    return _vChoose_71_12;
}

public _eSet findCopiesBorrowedByBorrower (Borrower b)
{
    _eSet _vChoose_74_12 = null;
    {
        _eSet _vCaptureBound_item_74_24 = checkedOut ();
        _vChoose_74_12 = new _eSet ();
        int _vCaptureCount_item_74_24 = _vCaptureBound_item_74_24._oHash ();
        int _vLoopCounter_74_18 = 0;
        for (;;)
        {
            if (_vLoopCounter_74_18 == _vCaptureCount_item_74_24) break;
            if (copyBorrowing (((LibraryItem) _vCaptureBound_item_74_24._oIndex (  
                _vLoopCounter_74_18))).borrowedBy ()._lEqual (b))
            {
                    _oIndex (_vLoopCounter_74_18));
            } else
            {
            }
            _vLoopCounter_74_18 = _eSystem._oSucc (_vLoopCounter_74_18);
        }
    }
    return _vChoose_74_12;
}
public void borrowLibraryItem (LibraryItem copy, Borrower bor, _eSeq date, char _t0date)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(itemAvailable (copy))) throw new _xPre("LibraryStock.pd:77,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    _n1_PriorityQueue _vUnshare_79_14 = ((_n1_PriorityQueue) borrowed._lClone ());
    borrowed = _vUnshare_79_14;
    _vUnshare_79_14.insert (((_eAny) new Borrowing (copy, bor, date, (char) 0)));
}

public void returnLibraryItem (LibraryItem copy)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(itemCurrentlyBorrowed (copy))) throw new _xPre("LibraryStock.pd:83,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    _n1_PriorityQueue _vUnshare_85_14 = ((_n1_PriorityQueue) borrowed._lClone ());
    borrowed = _vUnshare_85_14;
    _vUnshare_85_14.remove ()
}

public void addLibraryItem (LibraryItem copy)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!hasItem (copy))) throw new _xPre("LibraryStock.pd:89,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
    }
public void removeLibraryItem (LibraryItem copy) {
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(hasItem (copy))) throw new _xPre ("LibraryStock.pd:93,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(itemAvailable (copy))) throw new _xPre ("LibraryStock.pd:94,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    LibraryCatalog _vUnshare_95_14 = ((LibraryCatalog) catalog._lClone ());
catalog = _vUnshare_95_14;
    _vUnshare_95_14.removeLibraryItem (copy);
}

public boolean _lEqual (LibraryStock _vArg_11_9)
{
    if (this == _vArg_11_9) return true;
    return (_vArg_11_9.catalog._lEqual (catalog) && _vArg_11_9.currentlyBorrowed (). _lEqual (currentlyBorrowed ()));
}

public boolean equals (_eAny _lArg)
{
    return _lArg == this || (_lArg != null && _lArg.getClass () == LibraryStock.class
        && _lEqual ((LibraryStock) _lArg));
}

// End of file.

H.14 Person.java
package oolibrary;

import Ertsys.*;
public class Person extends _eAny {
    protected _eSeq firstName;
    protected _eSeq lastName;
    protected _eSeq dateOfBirth;
    public Person (_eSeq _vfirstName, char _t0_vfirstName, _eSeq _vlastName, char 
_t0_vlastName, _eSeq _vdateOfBirth, char _t0_vdateOfBirth)
    {
        super ();
        firstName = _vfirstName;
        lastName = _vlastName;
        dateOfBirth = _vdateOfBirth;
    }
    public _eSeq getName ()
    {
        return firstName._oPlusPlus (_eSystem._lString (" "), (_eTemplate_0) null).
        _oPlusPlus (lastName, (_eTemplate_0) null);
    }
    public _eSeq _rtoString ()
    {
        return getName ();
    }
    public boolean _lEqual (Person _vArg_11_9)
    {
        if (this == _vArg_11_9) return true;
        return ((_vArg_11_9.lastName._lEqual (lastName) && _vArg_11_9.dateOfBirth._lEqual 
(dateOfBirth)) && _vArg_11_9.firstName._lEqual (firstName));
    }
    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == Person.class &&
        _lEqual ((Person) _lArg));
    }
}
// End of file.

H.15 PriorityQueue.java
package oolibrary;

import Ertsys.*;
// Packages imported
import oolibrary.*;

class _n1_PriorityQueue extends _eAny {
    protected _n1_Heap myQueue;
    public _n1_PriorityQueue ()
    {
        super ();
    }
}
myQueue = new _n1_Heap ();

public void insert (_eAny a)
{
    _n1_Heap _vUnshare_17_14 = ((_n1_Heap) myQueue._lClone ());
    myQueue = _vUnshare_17_14;
    _vUnshare_17_14.insert (a);
}

public _eAny getElement ()
{
    if (_eSystem.enablePre & _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!!(!!myQueue.empty ())) throw new _xPre ("PriorityQueue.pd:20,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    return myQueue.largest ();
}

public void remove ()
{
    if (_eSystem.enablePre & _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!!(!!myQueue.empty ())) throw new _xPre ("PriorityQueue.pd:24,14");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    _n1_PriorityQueue _vAntiAlias_self_25_22 = this;
    _n1_Heap _vUnshare_25_14 = ((_n1_Heap) myQueue._lClone ());
    myQueue = _vUnshare_25_14;
    _vUnshare_25_14.remove (_vAntiAlias_self_25_22.getElement ());
}

public _eSet ran ()
{
    return myQueue.ran ();
}

public boolean _lEqual (_n1_PriorityQueue _vArg_11_9)
{
    if (this == _vArg_11_9) return true;
    return _vArg_11_9.myQueue._lEqual (myQueue);
}

public boolean equals (_eAny _lArg)
{
    return _lArg == this || (_lArg != null && _lArg.getClass () == _n1_PriorityQueue._class && _lEqual ((_n1_PriorityQueue) _lArg));
}
class _n1_Heap extends _eAny
{
    final void _lc_Heap (String _lArg)
    {
        if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
            maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!(_n1_Heap.isHeap (l, (_eTemplate_0) null))) throw new
                    _xClassInvariantItem ("PriorityQueue.pd:43,15", _lArg);
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
    }
    void _lClassInvariantCheck (String _lArg)
    {
        _lc_Heap (_lArg);
    }
    protected _eSeq l;
    protected static _eSeq heapify (_eSeq xs, _eTemplate_0 _t0xs, int low, int high)
    {
        if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
            )
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!xs.empty ())) throw new _xPre ("PriorityQueue.pd:48,13");
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
        if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
            )
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!((low <= high) && (high <= xs._oHash ()) && (0 < low))) throw new
                    _xPre ("PriorityQueue.pd:49,14");
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
    }
    int _vLet_large_51_17 = (2 * low);
    if (_eSystem.enableEmbeddedAssert && _eSystem.currentCheckNesting <= _eSystem.
        maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
if (((1 < _vLet_large_51_17))) throw new _xEmbeddedAssert ("PriorityQueue.pd:51,43");
}
catch (_xCannotEvaluate _lException)
{
}
_eSystem.currentCheckNesting --;
} return ((._vLet_large_51_17 <= high) ? ((((_vLet_large_51_17 < high) && (xs._oIndex (_eSystem._oPred (_vLet_large_51_17)).priority () < xs._oIndex (_eSystem._oPred (low)).priority ()) && (xs._oIndex (_eSystem._oPred (_vLet_large_51_17)).priority () < xs._oIndex (_eSystem._oPred (low)).priority ()))) ? _n1_Heap.heapify (_n1_Heap.exchange (xs, (_eTemplate_0) null, low, (1 + _vLet_large_51_17)), (_eTemplate_0) null, (1 + _vLet_large_51_17), high) : (xs._oIndex (_eSystem._oPred (low)).priority () < xs._oIndex (_eSystem._oPred (_vLet_large_51_17)).priority ())) ? _n1_Heap.heapify (_n1_Heap.exchange (xs, (_eTemplate_0) null, low, _vLet_large_51_17), (_eTemplate_0) null, _vLet_large_51_17, high) : xs) : xs);
}
protected _eSeq sift_up (_eSeq xs, _eTemplate_0 _t0xs, int pos)
{
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting )
{
_eSystem.currentCheckNesting ++;
try
{
if ((((0 < pos) && (pos <= xs._oHash ()))) throw new _xPre ("PriorityQueue.pd:67,14");
}
catch (_xCannotEvaluate _lException)
{
}
_eSystem.currentCheckNesting --;
} int _vLet_parentPos_69_17 = _eSystem._oDiv (pos, 2);
return ((1 == pos) ? xs : (xs._oIndex (_eSystem._oPred (pos)).priority () <= xs._oIndex (_eSystem._oPred (_vLet_parentPos_69_17)).priority ()) ? xs : sift_up (_n1_Heap.exchange (xs, (_eTemplate_0) null, _vLet_parentPos_69_17, pos), (_eTemplate_0) null, _vLet_parentPos_69_17));
}
protected _eSeq sift_down (_eSeq xs, _eTemplate_0 _t0xs)
{
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting )
{
_eSystem.currentCheckNesting ++;
try
{
if (!(!xs.empty ()))) throw new _xPre ("PriorityQueue.pd:81,13");
}
catch (_xCannotEvaluate _lException)
{
}
_eSystem.currentCheckNesting --;
} return ((xs._oHash () == 1) ? 187
new _eSeq () :
    _n1_Heap.heapify (_n1_Heap.exchange (xs, (_eTemplate_0) null, 1, xs._oHash ()).front (), (_eTemplate_0) null, 1, _eSystem._oPred (xs._oHash ()));
}

public _n1_Heap ()
{
    super ();
    l = new _eSeq ();
    _lc_Heap ("PriorityQueue.pd:90,14");
}

public static boolean isHeap (_eSeq a, _eTemplate_0 _t0a)
{
    boolean _vQuantifierResult_93_12;
    {
        _vQuantifierResult_93_12 = true;
        int _vCaptureCount_p_93_25 = _eSystem._oDiv (a._oHash (), 2);
        int _vLoopCounter_93_19 = 1;
        for (;;)
        {
            if (((_vCaptureCount_p_93_25 < _vLoopCounter_93_19) || (!
                _vQuantifierResult_93_12))) break;
            _vQuantifierResult_93_12 = _n1_Heap.greaterPriority (_vLoopCounter_93_19,
                a, (_eTemplate_0) null);
            if (!_vQuantifierResult_93_12)
            {
            }
            else
            {
                _vLoopCounter_93_19 = _eSystem._oSucc (_vLoopCounter_93_19);
            }
        }
    }
    return _vQuantifierResult_93_12;
}

public static boolean greaterPriority (int p, _eSeq a, _eTemplate_0 _t0a)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((0 < p) && (p <= a._oHash ())))) throw new _xPre ("PriorityQueue.pd:96,14");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    int _vLet_child_97_17 = (2 * p);
    return ((_vLet_child_97_17 < a._oHash ()) ?
        ((a._oIndex (_eSystem._oPred (_vLet_child_97_17)).priority () <= a._oIndex (_eSystem._oPred (p)).priority () && (a._oIndex (_eSystem._oPred (_vLet_child_97_17) + 1)).priority () <= a._oIndex (_eSystem._oPred (p)).priority ()):
        (a._oHash () == _vLet_child_97_17) ?
        (a._oIndex (_eSystem._oPred (_vLet_child_97_17)).priority () <= a._oIndex (_eSystem._oPred (p)).priority () :
        true);
    }
public static _eSeq exchange (_eSeq xs, _eTemplate_0 _t0xs, int i, int j)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(((i < j) && (j <= xs._oHash ())) && (0 < i))) throw new _xPre ("PriorityQueue.pd:109,14");
        } catch (_xCannotEvaluate _lException) { }
        _eSystem.currentCheckNesting --;
    }
    return xs.take (_eSystem._oPred (i)).append (xs._oIndex (_eSystem._oPred (j)))._oPlusPlus (xs.take (_eSystem._oPred (j)).drop (i).append (xs._oIndex (_eSystem._oPred (i))), (_eTemplate_0) null)._oPlusPlus (xs.take (xs._oHash ()).drop (j), (_eTemplate_0) null);
}

public static void swap (_eWrapper__eAny xs, _eSeq _t0xs, _eTemplate_0 _t1xs, int i, int j)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(((i < j) && (j <= ((_eSeq) xs.value)._oHash ())) && (0 < i))) throw new _xPre ("PriorityQueue.pd:116,14");
        } catch (_xCannotEvaluate _lException) { }
        _eSystem.currentCheckNesting --;
    }
    _eAny _vLet_temp_117_19 = ((_eSeq) xs.value)._oIndex (_eSystem._oPred (i));
    ((_eSeq) xs.value)._oaIndex (_eSystem._oPred (i), ((_eSeq) xs.value)._oIndex (_eSystem._oPred (j)));
    ((_eSeq) xs.value)._oaIndex (_eSystem._oPred (j), _vLet_temp_117_19);
}

public void insert (_eAny a)
{
    l = sift_up (l.append (a), (_eTemplate_0) null, (l._oHash () + 1));
    _lClassInvariantCheck ("PriorityQueue.pd:124,14");
}

public void remove (_eAny a)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(!empty ())) throw new _xPre ("PriorityQueue.pd:129,13");
        } catch (_xCannotEvaluate _lException) { }
        _eSystem.currentCheckNesting --;
    }
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting )
{
    _eSystem.currentCheckNesting ++;
    try
    {
        if (!(_eSystem._lEqual (a, largest ()))) throw new _xPre ("PriorityQueue.pd:130,20");
    }
    catch (_xCannotEvaluate _lException)
    {
    }
    _eSystem.currentCheckNesting --;
}
l = sift_down (l, (_eTemplate_0) null);
_lClassInvariantCheck ("PriorityQueue.pd:131,14");
}

public _eAny largest ()
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!empty ()) throw new _xPre ("PriorityQueue.pd:136,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    return l.head ();
}

public boolean empty ()
{
    return l.empty ();
}

public _eSeq _rtoString ()
{
    return l._rtoString ();
}

public _eSet ran ()
{
    return l.ran ();
}

public boolean _ovIn (_eAny a)
{
    return l._ovIn (a);
}

public boolean _lEqual (_n1_Heap _vArg_41_9)
{
    if (this == _vArg_41_9) return true;
    return _vArg_41_9._lEqual (l);
}

public boolean equals (_eAny _lArg)
{
return _lArg == this || (_lArg != null && _lArg.getClass () == _n1_Heap.class &&
                 _lEqual ((_n1_Heap) _lArg));
}
}
H.17 StaffBase.java

public class StaffBase extends _eAny
{
    final void _lc_StaffBase (String _lArg)
    {
        if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
            maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                boolean _vQuantifierResult_13_15;
                {
                    _vQuantifierResult_13_15 = true;
                    int _vCaptureCount_s1_13_29 = allStaff._oHash ();
                    int _vLoopCounter_13_22 = 0;
                    for (;;)
                    {
                        if (((_vLoopCounter_13_22 == _vCaptureCount_s1_13_29) || (!
                            _vQuantifierResult_13_15)) break;
                        boolean _vQuantifierResult_13_25;
                        {
                            _vQuantifierResult_13_25 = true;
                            int _vCaptureCount_s2_13_29 = allStaff._oHash ();
                            int _vLoopCounter_13_25 = 0;
                            for (;;)
                            {
                                if (((_vLoopCounter_13_25 == _vCaptureCount_s2_13_29) || (1
                                    _vQuantifierResult_13_25))) break;
                                _vQuantifierResult_13_25 = true;
                                int _vCaptureCount_s3_13_29 = allStaff._oHash ();
                                int _vLoopCounter_13_25 = 0;
                                for (;;)
                                {
                                    if (((_vLoopCounter_13_25 == _vCaptureCount_s3_13_29) || (1
                                        _vQuantifierResult_13_25))) break;
                                    boolean _vQuantifierResult_13_25 = (!(!((Staff) allStaff._oIndex (
                                        _vLoopCounter_13_25))._lEqual (((Staff) allStaff.
                                            _oIndex (_vLoopCounter_13_25))).getId () == (1
                                                ((Staff) allStaff._oIndex (_vLoopCounter_13_25)).getId (1
                                                    ))));
                                    if (!(_vQuantifierResult_13_25))
                                    {
                                        _vLoopCounter_13_25 = _eSystem._oSucc (_vLoopCounter_13_25);
                                    
                                        break;
                                    }
                                    else
                                    {
                                        _vLoopCounter_13_25 = _eSystem._oSucc (_vLoopCounter_13_25);
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
        catch (_xCannotEvaluate _lException)
        {
        }
    }
    _eSystem.currentCheckNesting --;
}
void _lClassInvariantCheck (String _lArg)
{
    _lc_StaffBase (_lArg);
}

protected _eSet allStaff;
public StaffBase ()
{
    super ();
    allStaff = new _eSet ();
    _lc_StaffBase ("StaffBase.pd:16,14");
}

public boolean uniqueId (Staff s)
{
    return (!usedIds () _ovIn (((_eAny) new _eWrapper_int (s.getId ()))))
}

public _eSet usedIds ()
{
    _eSet _vForYield_22_12;
    {
        _vForYield_22_12 = new _eSet ();
        int _vCaptureCount_s_22_19 = allStaff _oHash ();
        int _vLoopCounter_22_16 = 0;
        for (;;)
        {
            if (_vLoopCounter_22_16 == _vCaptureCount_s_22_19) break;
            _vForYield_22_12 = _vForYield_22_12.append (((_eAny) new _eWrapper_int (((Staff) allStaff _oIndex (_vLoopCounter_22_16)).getId ())));
            _vLoopCounter_22_16 = _eSystem _oSucc (_vLoopCounter_22_16);
        }
    }
    return _vForYield_22_12;
}

public void addStaff (Staff sta)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (! (uniqueId (sta))) throw new _xPre ("StaffBase.pd:25,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    allStaff = allStaff.append (((_eAny) sta));
    _lClassInvariantCheck ("StaffBase.pd:26,14");
}

public boolean _lEqual (StaffBase _vArg_11_9)
{
    if (this == _vArg_11_9) return true;
    return _vArg_11_9.allStaff _lEqual (allStaff);
}

public boolean equals (_eAny _lArg)
{
    return _lArg == this || (_lArg != null && _lArg.getClass () == StaffBase.class && _lEqual ((StaffBase) _lArg));
}
public class UserBase extends _eAny
{
    protected BorrowerBase allBorrowers;
    protected StaffBase allStaff;
    public UserBase ()
    {
        super ();
        allBorrowers = new BorrowerBase ();
        allStaff = new StaffBase ();
    }

    public _eSet usedIds ()
    {
        return allBorrowers.usedIds ()._oPlusPlus (allStaff.usedIds (), (_eTemplate_0)
            null);
    }

    public boolean uniqueId (Borrower u)
    {
        return (!hasId (u.getId ()));}

    public boolean uniqueId (Staff u)
    {
        return (!hasId (u.getId ()));}

    public boolean hasId (int uId)
    {
        return usedIds ()._ovIn (((_eAny) new _eWrapper_int (uId)));
    }

    public Borrower findUser (int id)
    {
        if (_eSystem.enablePre & _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!hasId (id)) throw new _xPre ("UserBase.pd:33,13");
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
    }
Borrower _vChoose_34_12 = null;
{
_vCaptureBound_user_34_36 = allBorrowers.allTheBorrowers();
boolean _vSelectorCondition_34_12;
_vSelectorCondition_34_12 = false;
_captureCount_user_34_36 = _vCaptureBound_user_34_36._oHash();
_loopCounter_34_17 = 0;
for (;;)
{
if (((_loopCounter_34_17 == _captureCount_user_34_36) ||
_vSelectorCondition_34_12)) break;
_vSelectorCondition_34_12 = (((Borrower) _vCaptureBound_user_34_36._oIndex
(_loopCounter_34_17)).getId () == id);
if (_vSelectorCondition_34_12)
{
_vChoose_34_12 = ((Borrower) _vCaptureBound_user_34_36._oIndex (_loopCounter_34_17));
}
else
{
}
if (_vSelectorCondition_34_12)
{
}
else
{
_loopCounter_34_17 = _eSystem._oSucc (_loopCounter_34_17);
}
}
if (_eSystem.enableThatOrAny && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
_eSystem.currentCheckNesting ++;
try
{
if (!(_vSelectorCondition_34_12)) throw new _xThatOrAny ("UserBase.pd:34,12");
}
catch (_xCannotEvaluate _lException)
{
}
_eSystem.currentCheckNesting --;
}
return _vChoose_34_12;
}

public void addBorrower (Borrower bor)
{
if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
{
_eSystem.currentCheckNesting ++;
try
{
if (!uniqueId (bor)) throw new _xPre ("UserBase.pd:37,13");
}
catch (_xCannotEvaluate _lException)
{
}
_eSystem.currentCheckNesting --;
}
BorrowerBase _Unshare_38_14 = ((BorrowerBase) allBorrowers._lClone());
allBorrowers = _Unshare_38_14;
_Unshare_38_14.addBorrower (bor);
public void addStaff (Staff sta)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(uniqueId (sta))) throw new _xPre ("UserBase.pd:41,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    StaffBase _vUnshare_42_14 = ((StaffBase) allStaff._lClone ());
    allStaff = _vUnshare_42_14;
    _vUnshare_42_14.addStaff (sta);
}

public boolean _lEqual (UserBase _vArg_11_9)
{
    if (this == _vArg_11_9) return true;
    return (_vArg_11_9.allBorrowers._lEqual (allBorrowers) && _vArg_11_9.allStaff.
        _lEqual (allStaff));
}

public boolean equals (_eAny _lArg)
{
    return _lArg == this || (_lArg != null && _lArg.getClass () == UserBase.class &&
        _lEqual ((UserBase) _lArg));
}

// End of file.
I Resource Manager Specification Code

Here is the complete specification of the Resource Manager.

I.1 ProcessErrorCode.pd

```plaintext
class ProcessErrorCode :=
enum
    running, // the process is executing
    sleeping, // the process is waiting on a resource
    terminated // the process has terminated successfully
end;
```

I.2 ProcessItem.pd

```plaintext
class ProcessItem :=
inherits ResourceItem
abstract
var
    state: ProcessStateCode,
    hasResources: set of from ResourceItem,
    needsResources: set of from ResourceItem;

invariant hasResources**needsResources=set of from ResourceItem{};
interface

// constructor
build{i:nat,!state:ProcessStateCode,
     !hasResources: set of from ResourceItem,
     !needsResources: set of from ResourceItem}
pre hasResources**needsResources= set of from ResourceItem{}
inherits ResourceItem{i};

// constructor that creates a process with no needs.
build{i:nat}
^= ProcessItem{i, running@ProcessStateCode,
               set of from ResourceItem{},
               set of from ResourceItem{}};

// constructor that creates a process with needs.
build{i:nat, needs: set of from ResourceItem}
^= ProcessItem{i, running@ProcessStateCode,
               set of from ResourceItem{},
               needs};

/***
/***/
/**** Methods controlling & describing state of a processItem */
/***/
// determines if this process has met all it's needs
function allNeedsMet: bool ^= #needsResources=0;

// determines if this process has terminated
function terminated: bool ^= state=terminated@ProcessStateCode;

// determines if this process has running
function running: bool ^= state=running@ProcessStateCode;

// determines if the process is currently sleeping
function sleeping: bool ^= state=sleeping@ProcessStateCode;

// determines if this process is alive
function alive: bool ^= ~terminated;
```
// puts a process to sleep(symbolically),
// it must not be sleeping already
schema !sleep
    pre ~sleeping
    post state!=sleeping@ProcessStateCode;

// terminate the process, given that it isn't holding onto
// any resources and has met all it's needs
// => held all the resources it needed originally
schema !terminate
    pre ~holdingResources,
        allNeedsMet
    post state!=terminated@ProcessStateCode;

// force a termination on a process regardless of needs met
schema !forceTerminate
    post state!=terminated@ProcessStateCode;

// Methods concerned with resource acquisition
// and process execution

// determine if this process has needs
function hasNeeds:bool
    ^=needsResources.empty;

// determines if this process needs a given resource
function needs(r:from ResourceItem):bool
    ^= r in needsResources;

// determines if this process currently holds any resource
function holdingResources:bool
    ^= #hasResources ^= 0;

// determines if this process currently holds a resource
function holdingResource(r:from ResourceItem):bool
    ^= r in hasResources;

// determines the Resources that the process holds
redefine function next:set of from ResourceItem
    ^= hasResources;

// pick a random need of the process : Random???
function getNeed:from ResourceItem
    pre hasNeeds
    satisfy result in needsResources
    via
        value needsResources.min
    end;

// pick a random resource the process has : Random???
function getAcquired:from ResourceItem
    pre holdingResources
    satisfy result in hasResources
    via
        value hasResources.min
    end;

// allows us to add needs to process item
schema !addNeeds(needs:set of from ResourceItem)
    pre needs ** hasResources = set of from ResourceItem()
    post needsResources!=needsResources++needs;

// allocate a resource to this process item
schema !granted(r:from ResourceItem)
  pre needs(r), sleeping
  post needsResources!=needsResources.remove(r),
       hasResources!=hasResources.append(r),
       state!=running@ProcessStateCode;

  // called when the process item no longer needs a resource
  schema !remove(r:from ResourceItem)
  post hasResources!=hasResources.remove(r);

function progress:nat
  ^= for r::needsResources yield r.processPosition(self);
end;

I.3 ProcessStateCode.pd
class ProcessStateCode ^=
  enum
    running, // the process is executing
    sleeping, // the process is waiting on a resource
    terminated // the process has terminated
  end;

I.4 ResourceItem.pd
class ResourceItem ^=
  abstract
  var
    id:nat,
    lock:Semaphore;

    invariant forall p::lock.waitingOn :- p.sleeping;
    // invariant forall p::lock.waitingOn :- p.needs(self);

  interface
    // This is required to use the "from ResourceItem" for object
    // instantiation. This will allow equals to be defined on all
    // objects in the inheritance hierarchy of ResourceItem
    operator = (arg);

    // constructor.
    build{id:nat}
      post lock!=Semaphore{};

    // get the Id of the resource
    function getId:nat
      ^= id;

    // determine if the resource is available(i.e. lock is free)
    function isAvailable:bool
      ^= lock.isFree;

    // determine if the resource has processes waiting on it.
    function hasWaiting:bool
      ^= lock.hasWaiting;

    // get their next lot of resourceItems the system is waiting on.
    function next:set of from ResourceItem
      ^= for p::lock.waitingOn yield p as from ResourceItem;

    // declare a want of the resource
    schema !acquire(p!:ProcessItem)
      pre p.needs(self),"p.sleeping
      post lock!P1(p!);
// actually retrieve the resource
schema !grant(p!:out ProcessItem)
    pre hasWaiting, isAvailable
    post lock!P2(p!)
    assert p'.sleeping; //, p'.needs(self);

schema !free
    post ([lock.isFull]:
        // a process Item is freeing the resource
        // while the resource is not allocated
        // !!!!Indicates an error!!!!!
        pass,
        []:
        lock!V
    );

// A reset scheme, frees the resource from all responsibilities
schema !reset
    post lock!reset;

function processPosition(p:ProcessItem):nat
    ^= lock.lockPosition(p);
end;

I.5 ResourceManager.pd

class ResourceManager ^=
    abstract
    var
        id:nat;
        // The manager is the algorithm by which the resource allocation
        // scheme is implemented. By all rights it should be just a part
        // of the System, however we represent it as an object in order
        // that should the resource allocation scheme be changed, we can
        // use inheritance to re-use the resource manager code, and
        // change the the allocation scheme. The system will still be
        // represented by the no deadlock, no starvation, fairness
        // invariants but the resource manager will be the code that
        // implements them.

    interface
        build{!id:nat};

    // A process states want of a resource, will be put to sleep
    // and on a subsequent grant of the resource, will be acquired
    // two stage operation owing to lack of multi-threaded support.
    // This method needs to terminate.
    schema acquire(p!:ProcessItem, r!:from ResourceItem)
        pre ~p.sleeping, p.needs(r)
        post r!acquire(p!);

    // The granting of a resource to an instantiated process. The
    // input parameter of p could be anything, but the return value
    // will be important. We need to signal if the resources can't
    // be granted owing to lack of availability.
    schema grant(r!:from ResourceItem)
        pre r.hasWaiting
        post (var p:ProcessItem;
            [r.isAvailable]:
            r!grant(p!)then
            p!granted(r),
// Can't grant a process. Do we need to signal?
pass);

// Release a resource
schema release(p!:ProcessItem, r!:from ResourceItem)
pre p.holdingResource(r)
post p!remove(r), r!free;
end;

I.6 Semaphore.pd

class Semaphore ^=
abstract
  var
    maxSem:nat, // maximum Value of the Semaphore
    currentSem:nat, // value of semaphore
    waitingQ:seq of ProcessItem; //items waiting on it
  invariant currentSem<=maxSem;
  invariant forall p::waitingQ :- p.sleeping;

interface

  build{}
  ^= Semaphore{1};
  build{!maxSem:nat}
  pre maxSem>0
  post currentSem!=maxSem,
      waitingQ!=seq of ProcessItem{};

  // test whether the Semaphore is Free or has been locked
  function isFree:bool
  ^= currentSem~=0;

  function hasWaiting:bool
  ^= ~waitingQ.empty;

  // test if the semaphore has it's maximum value to ensure
  // the semaphore isn't released when it wasn't acquired
  function isFull:bool
  ^= currentSem=maxSem;

  // test on the number of processes waiting on the semaphore
  function count:nat
  ^= #waitingQ;

  // the method P needed to be segmented into two parts.
  // the first is always acceptable and deals with a process
  // wanting to be granted access to the semaphore
  schema !P1(item!:ProcessItem)
  pre ~item.sleeping
  post item!sleep then
      waitingQ!=waitingQ.append(item);

  // the second will only be accepted if the semaphore is free
  // and returns the process that now has the semaphore to allow
  // for it to be executed
  schema !P2(item!:out ProcessItem)
  pre isFree, ~waitingQ.empty
  post currentSem!=currentSem-1,
      item!=waitingQ.head,
      waitingQ!=waitingQ.tail;
the v method unlocks the semaphore, on a subsequent call P2
the semaphore will be allocated to the next item in the queue

```plaintext
schema !V
pre ~isFull
post currentSem!=currentSem+1;
```

// reset the semaphore object release it from responsibilities
schema !reset
post currentSem!=maxSem, waitingQ!=seq of ProcessItem{};

function waitingOn:set of ProcessItem
~= waitingQ.ran;

function lockPosition(p:ProcessItem):nat
~= ([p in waitingQ]:
   waitingQ.findFirst(p),
   []: #waitingQ);
end;

I.7 System.pd

// System with one resource, no processes is not in deadlock
property assert ~(System{
   set of from ResourceItem{ResourceItem{1}},
   set of ProcessItem{}}).hasDeadlock;

// System with one resource, one processes is not in deadlock
property assert ~(System{
   set of from ResourceItem{ResourceItem{1}, ProcessItem{2}},
   set of ProcessItem{ProcessItem{2}}} after it!step(1)).hasDeadlock;

class System ^= abstract
var
   resources:set of from ResourceItem,
   manager:ResourceManager,
   toBeExecuted:set of ProcessItem;

invariant forall r::resources :- r.next<<=resources;

// deadlock detection algorithm
// must be placed before the invariant that declares
// the system doesn’t contain deadlock
// function hasDeadlock:bool
// ^=([resources.empty]:
//   false,
//   []:
//     exists r::resources :- hasCycle(set of from ResourceItem{},r,r.next.rep(1))
//   );

function hasDeadlock:bool
~=self after it!step.progress > self.progress;

function hasCycle(viewed:set of from ResourceItem,at:from ResourceItem,
toBeViewed:bag of from ResourceItem):bool
pre viewed <<= resources,
at in resources,
toBeViewed.ran <<= resources
decrease #resources - #viewed
~= ([toBeViewed.empty]:
   false,
   [at in viewed]:

202
true,
[]: (let newAt=toBeViewed.min;
    hasCycle(viewed.append(at),newAt,toBeViewed.remove(newAt)++newAt.next.rep(1))
    )
);

invariant ~hasDeadlock;

interface

nonmember function processesAreResources(res:set of from ResourceItem,
    pro:set of ProcessItem):bool
  ^= (let processesAsResources ^=
      for p::pro yield p as from ResourceItem;
          processesAsResources <<= res
    );

// constructors
build{res:set of from ResourceItem,
    man:ResourceManager,
    toBeExe:set of ProcessItem}
pre processesAreResources(res,toBeExe),
    forall r::res :- r.next=set of from ResourceItem()
post resources!=res,
    manager!=man,
    toBeExecuted!=toBeExe;

build{res:set of from ResourceItem,
    toBeExe:set of ProcessItem}
pre processesAreResources(res,toBeExe),
    forall r::res :- r.next=set of from ResourceItem()
post resources!=res,
    manager!=ResourceManager{1},
    toBeExecuted!=toBeExe;

// empty system
build{} ^= System{set of from ResourceItem{},ResourceManager{1},set of ProcessItem{}};

// determines that all the processes have terminated
// (i.e. system has completed)
function systemEnd:bool
  ^= forall p::toBeExecuted :-p.terminated;

// collections of processes of particular state in our system
function deadProcesses:set of ProcessItem
  ^= those p::toBeExecuted :- p.terminated;

function activeProcesses:set of ProcessItem
  ^= those p::toBeExecuted :- p.running;

function hungryProcesses:set of ProcessItem
  ^= those p::toBeExecuted :- p.running & p.hasNeeds;

function hasHungry:bool
  ^= ~hungryProcesses.empty;

function sleepingProcesses:set of ProcessItem
  ^= those p::toBeExecuted :- p.sleeping;

function hasSleeping:bool
  ^= ~sleepingProcesses.empty;

function holdingResources:set of ProcessItem
  ^= those p::toBeExecuted :- p.holdingResources;
function hasAcquired:bool
  ^= ~holdingResources.empty;

function neededResources:set of ResourceItem
  ^= those r::resources :- r.hasWaiting;

function hasNeeded:bool
  ^= ~neededResources.empty;

// pick a random active process : Random???
function pickActive:ProcessItem
  pre ~activeProcesses.empty
  satisfy result in activeProcesses
  // current implementation just picks the minimum element
  // there should be some randomisation of this...?
  via value activeProcesses.min
  end;

// pick a random hungry process : Random???
function pickHungry:ProcessItem
  pre hasHungry
  satisfy result in hungryProcesses
  // current implementation just picks the minimum element
  // there should be some randomisation of this...?
  via value hungryProcesses.min
  end;

// pick a random sleeping process : Random???
function pickSleeping:ProcessItem
  pre hasSleeping
  satisfy result in sleepingProcesses
  // current implementation just picks the minimum element
  // there should be some randomisation of this...?
  via value sleepingProcesses.min
  end;

// pick a random process that holds a resource : Random???
function pickHasAcquired:ProcessItem
  pre hasAcquired
  satisfy result in holdingResources
  // current implementation just picks the minimum element
  // there should be some randomisation of this...?
  via value holdingResources.min
  end;

// pick a random needed resource : Random???
function pickNeededResource:ResourceItem
  pre hasNeeded
  satisfy result in neededResources
  // current implementation just picks the minimum element
  // there should be some randomisation of this...?
  via value neededResources.min
  end;

function hasDeadlock;

function progress:nat
  ^= for p::toBeExecuted yield p.progress;

// resource manager steps the system.
schema !step(choice:nat)
  pre ~systemEnd // there is work to do.
  post (var p:ProcessItem,r:ResourceItem;
    [choice=0]: // declare want of resource
    ([hasHungry]:
    204
p!=pickHungry then
    resources!=resources.remove(p) then
    toBeExecuted!=toBeExecuted.remove(p) then
    r!=p.getNeed then
    resources!=resources.remove(r) then
    manager.acquire(p!, r!) then
    resources!=resources.append(p) then
    toBeExecuted!=toBeExecuted.append(p) then
    resources!=resources.append(r),
[ ]:
    pass // System has allocated all it needs
    // we don't need to acquire any more
),
[choice=1]: // grant the resource to a process
([hasNeeded]):
    r!=pickNeededResource then
    resources!=resources.remove(r) then
    manager.grant(r!) then
    resources!=resources.append(r),
[ ]:
    pass // System has no processes waiting on
    // resources currently.
),
[ ]: // release a resource
([hasAcquired]):
    p!=pickHasAcquired then
    resources!=resources.remove(p) then
    toBeExecuted!=toBeExecuted.remove(p) then
    r!=p.getAcquired then
    resources!=resources.remove(r) then
    manager.release(p!, r!) then
    ([~p.holdingResources & p.allNeedsMet]:
        p!terminate, // end of life
    [ ]:
        pass
) then
    resources!=resources.append(p) then
    toBeExecuted!=toBeExecuted.append(p) then
    resources!=resources.append(r),
[ ]:
    pass // the system currently has no
    // resources allocated currently
)
// should the 'pass' routes of execution
// signal errors?
);

end;
Higher Order Functions with Perfect Developer

Here is the Perfect Developer code for our attempt to include a truly generic language of Higher Order Functions in Perfect Developer.

J.1 HOF.pd

defered class HOF of (X,Y)
  "=
  interface
      build{}
      // operations on Higher Order functions
      nonmember function broadcast(a:class A,list:seq of class B):seq of pair of (A,B)
          "= for i::0..<#list yield pair of (A,B){a,list[i]};
      nonmember function zip(a:seq of class A,b:seq of class B):seq of pair of (A,B)
          "= for i::0..<min(#a,#b) yield pair of (A,B){a[i],b[i]};
      nonmember function mapF(f:from HOF of (class A,class B),list:seq of A):seq of B
          "= for x::list yield f.myFunction(x);
      nonmember function filter(f:from HOF of (class A,bool),list:seq of A):seq of A
          "= those i::list :- f.myFunction(i);
      nonmember function foldL(f:from HOF of (pair of (class A,A),A),list:seq of A):A
          pre ~list.empty
          decrease #list
          "= ([list.tail.empty]:
              list.head,
              []: f.myFunction(pair of (A,A){list.head,foldL(f,list.tail)}))
          );
      nonmember function foldR(f:from HOF of (pair of (class A,A),A),list:seq of A):A
          pre ~list.empty
          decrease #list
          "= ([list.tail.empty]:
              list.head,
              []: f.myFunction(pair of (A,A){foldR(f,list.tail),list.head}))
          );
      nonmember function compose(f:from HOF of (X,class B),g:from HOF of (B,Y),a:X):Y
          "= g.myFunction(f.myFunction(a));
      // Deferred function that represents a single order function
      // Defined on class by class basis as required
      deferred function myFunction(x:X):Y;
  end;

J.2 Double.pd

class Double of (X,Y) require X has function double:Y end=" inherits HOF of (X,Y)
  interface
      build{} inherits HOF of (X,Y){};
      define function myFunction(a:X):Y
          "= a.double;
  end;

J.3 MyInteger.pd

class MyInteger "= 
abstract
    var val:int;
interface
    build(a:int)
       post val!=a;

    function val;

    function double:MyInteger
        ^= MyInteger((2*val));
end;

J.4 QuadInt.pd
class QuadInt ^=
    interface
        build{};

        function quad(a:int):int
            ^= (let f=Double of (MyInteger,MyInteger){} as HOF of (MyInteger,MyInteger);
                compose@HOF of (MyInteger,MyInteger)(f,f,MyInteger{a}).val
        );
end;
K  Vegetarian Problem in Perfect Developer
Here is a presentation of a sensible piece of software that Perfect Developer won’t allow.

K.1  Food.pd
class Food ~=
  abstract
  var foodName:string;// Add variable, invariant and private method declarations here...
  interface
    function foodName;
    build{!foodName:string};
  end;

K.2  Vegetable.pd
class Vegetable ~=
  inherits Food
  abstract
  interface
    build{vName:string} inherits Food{vName};
  end;

K.3  Person.pd
class Person ~=
  abstract
  var personName:string;
  interface
    build{!personName:string};
    function personName;
    function eat(f:from Food):string
      pre f within Food
      ^= personName ++ " has eaten "++ f.foodName;
    end;

K.4  Vegetarian.pd
class Vegetarian ~=
  inherits Person
  abstract
  interface
    build{pName:string} inherits Person{pName};
  end;
  redefine function eat(f:from Food):string
    pre f within Vegetable
    ^= personName ++ " has eaten "++ f.foodName;
  end;

K.5  Main.pd
schema main(context!: limited Environment, args: seq of string,
   ret!: out int)
  pre #args > 0
  post
    ( var
      meat:Food!= Food{"Beef"},
      isMeat:from Food!=Food{"Ham"},
      isVeg:from Food!=Vegetable{"Carrots"},
      person:Person!=Person{"John"},
      veget1:Vegetarian!=Vegetarian{"Paul"},
      ...)
veget2: from Person != Vegetarian("George");

context!print(person.eat(meat)++"\n") then
context!print(person.eat(isMeat)++"\n") then
context!print(person.eat(isVeg)++"\n") then
context!print(veget1.eat(meat)++"\n") then
context!print(veget1.eat(isVeg)++"\n") then
context!print(veget1.eat(isMeat)++"\n") then
context!print(veget2.eat(meat)++"\n") then
context!print(veget2.eat(isVeg)++"\n") then
context!print(veget2.eat(isMeat)++"\n")
context!print(makeThemEat(person,meat)) then
context!print(makeThemEat(person,isMeat)) then
context!print(makeThemEat(person,isVeg)) then
context!print(makeThemEat(veget1,meat)) then
context!print(makeThemEat(veget1,isVeg)) then
context!print(makeThemEat(veget1,isMeat)) then
context!print(makeThemEat(veget2,meat)) then
context!print(makeThemEat(veget2,isVeg)) then
context!print(makeThemEat(veget2,isMeat))
)

function makeThemEat(p: from Person, f: from Food): string
pre p.edible(f)
"= p.eat(f)++ "\n";

ret! = 0;
L. Vegetarian Solution in Perfect Developer

Here is the solution to the Vegetarian Problem that Perfect Developer supports

L.1 Person.pd

```plaintext
class Person ^=
  abstract
  var personName: string;
  interface
  build{!personName: string};

  function personName;

  function eat(f: from Food): string
    pre edible(f)
    ^= personName ++ " has eaten " ++ f.foodName;

  function edible(f: from Food): bool
    ^= true;
end;
```

L.2 Vegetarian.pd

```plaintext
class Vegetarian ^=
  inherits Person
  abstract

  interface
  build{pName: string} inherits Person{pName};

  redefine function edible(f: from Food): bool
    ^= f within Vegetable
end;
```