Associations

How do objects communicate?

- In any OO system we would expect many objects of many different classes to communicate with each other.
- In order to do this we must provide links between object to allow them communicate.
- At the level of class design these links are known as associations and may be
  - `1:1 one object of a class has a link with one object of another class`
  - `1:N one object of a class has a link with many objects of another class`
  - `N:M many objects of a class has a link with many objects of another class`

- Associations are generally assumed to be bi-directional.
- Associations more frequently occur between objects of different classes, but also occur between objects of the same class.
- The term "**actor**" (Booch) is used to describe an object which acts upon other objects, "**server**" for an object that is acted upon and **agent** for an object which does both.

Aggregation

- Associations form hierarchies which are different from inheritance. These hierarchies are known as aggregations.
- Some associations form hierarchies, but these hierarchies are different from inheritance.
- These special types of association is described by terms which include:
  - `Aggregation, Composition, Part-Whole,`
  - `A-Part-Of, Has-a, Containment`

A Part Of

- In this type of hierarchy, classes do not inherit from other classes but are composed of other classes.
- Aggregation is **a part of** relationship in which objects representing the components of something are associated with an object representing an entire assembly:

Example:

- A sentence is **a part of** a paragraph which is **a part of** a document.

- **Containment**
  - A composition hierarchy defines how an object is composed of other objects in a fixed relationship. The aggregate object cannot exist without its components, which are usually a fixed and stable number.

- **Container**
  - A container is an object of a container class which is able to contain other objects. The existence of the container is independent of whether it actually contains any objects at a particular time. The contained objects are usually dynamic and may be of many different classes.

- **Composition**
  - internal objects are not seen from the outside e.g. Clock is composed of Hand, Face and Works
- **Aggregation**
  - internal objects can be directly accessed from outside objects have a separate existence e.g. Clock Battery

- Aggregation is primarily a relationship between objects rather than a relationship between classes. Elements of a composition hierarchy must be objects at run time in order to create objects of the aggregate class.
- **E.g** we must instantiate the objects face, hand and works before we use them for a “clock” object.

- Note the difference between this and inheritance hierarchies - we don’t have to instantiate the class vehicle before we instantiate the class bus (bus is a class which inherits from vehicle).
- Partial aggregations are the most common in OOP - one class uses one or more objects of another class in order to represent its internal state, but may have other attributes which are not objects.
- There are two basic ways in which associations and aggregations are implemented
  - .Objects contain objects (fixed aggregation)
  - .Objects contain pointers to objects (variable aggregations and simple associations)


```
class Wheel
{
private:
  int size;

public:
  Wheel(int wheel_size)      { size =
  wheel_size; }
  int getSize()    { return size; }
};
```



```
class car
{private:
  Wheel rightfront_wheel, rightback_wheel,
  leftfront_wheel, leftback_wheel;
  Engine engine;
  Int passengers;

public:
  car(int wheel_size, int cc_in, int
  passengers_in);
  void showcar();
  int getFrontWheelSize();
  int getBackWheelSize();
};
```

The Constructor

```
car::car(int wheel_size, int cc_in, int
passengers_in):
      rightfront_wheel(wheel_size),
      rightback_wheel(wheel_size),
      leftfront_wheel(wheel_size),
      leftback_wheel(wheel_size),
      engine(cc_in)
{
 passengers= passengers_in;
}
```


The Method

```
void car::showcar()
{
 cout<<"Wheel Size"
 << rightfront_wheel.getSize()<<endl;
 cout<<"EngineCapacity:"
 << engine.getCC()<<endl;
 cout<<"No. of Passengers"
 << passengers<<endl;
}
```


Main

```
void main()
{

 car mycar(24,500,2);
 mycar.showcar();
)
```