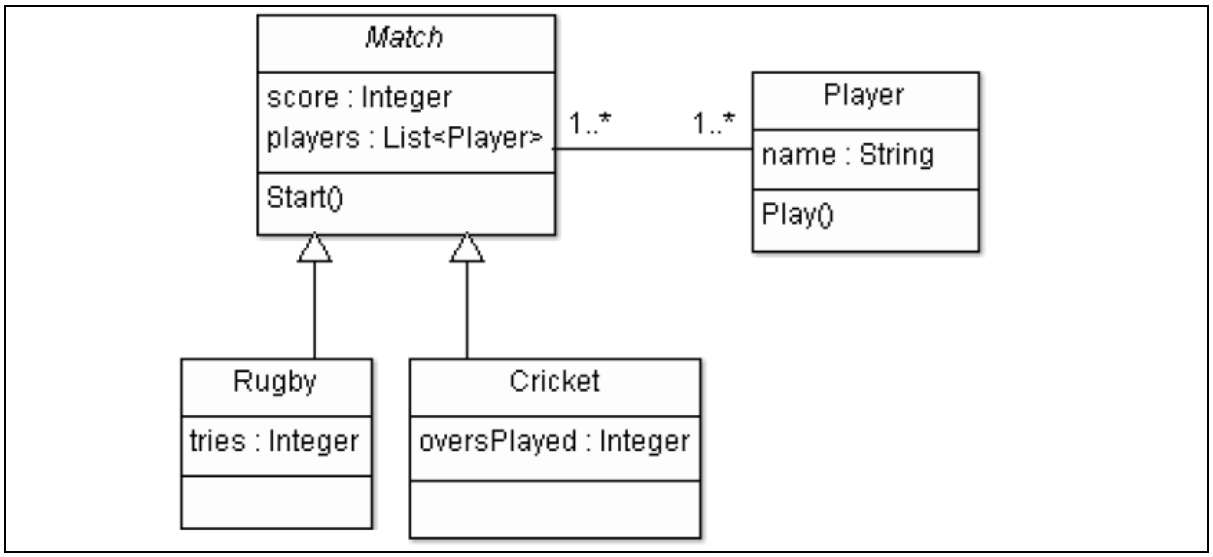
Tutorial 3 – Inheritance

This week we will look at classes, interfaces and abstraction, with a particular focus on inheritance. We'll use UML diagrams and standards to understand and answer questions about a system.

1: Examine the following simple relationship in the following diagram:



a. What type of diagram is this?

The above diagram is a UML Class Diagram.

b. What is the relationship between the Rugby and Match Classes?

Rugby is a subclass of Match. In other words, Match generalises Rugby. (This is inheritance).

c. What is the multiplicity between Match and Player?

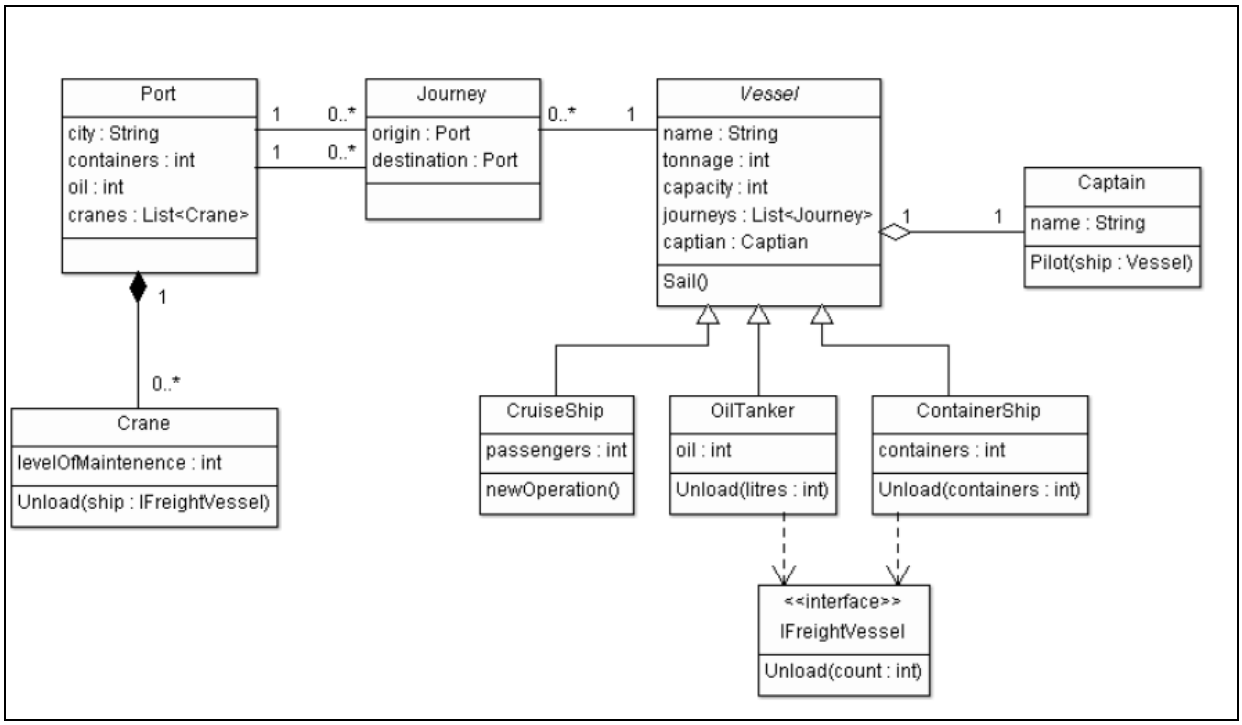
A match has between 1 and many players; a match has many players. A player can play many matches. You can't have a Match without a Player and vice versa.

d. What type of class do you think Match is likely to be?

Match could be an abstract class. Rugby and Cricket, conversely, would be concrete classes.

Expressive Relationships in UML

The following UML class diagram describes a way to model sea ports and large ships. We'll be examining this example in detail as it includes the association, aggregation, composition and interface relationships.



2. Name two classes that have an Association relationship

Vessel and Journey (or Port and Journey)

3. Name two classes that have an Aggregation relationship

Vessel and Captain

4: Name two classes that have a Composition relationship

Port and Crane

Refer to the class diagram on page 2 for the following questions.

5. Name a class that Implements an interface

OilTanker (or ContainerShip)

6. Name a class that extends another class (i.e. name a subclass)

CruiseShip (or OilTanker or ContainerShip)

7. Identify one abstract class

Vessel is an abstract class. In UML class diagrams, abstract classes are often represented by making the title of the class italicised.

8. List all Attributes of the OilTanker class (including inherited ones)

OilTanker has the attribute 'oil' and the inherited attributes:

'name', 'tonnage', 'capacity',
'journeys' and 'captain'.

9. Reference types are the different variable types that can be used to refer to an object and are related to polymorphism. What reference types can be used to refer to a ContainerShip object?

ContainerShip, IFreightVessel, Vessel and Object are valid reference types for a ContainerShip object.

Creating an inheritance hierarchy

Given is the abstract class Vehicle and the classes Car and Bicycle. Think carefully about what the relationship between the classes are.

Every vehicle has a name of type String, a production year of type int and can change gear, accelerate and brake (The car and bicycle classes implement this in their own way).

However, only Car objects have unique attributes of engine size of type String and can turn on their headlights and turn on their windscreen wipers which return Boolean values.

Similarly, bicycles have a unique attribute cadence (no. of revolutions per minute) of type int, as well as a number of unique responsibilities which including ringing a bell and changing the cadence which also return Boolean values.

Complete the skeleton code by writing the instance variable and method declarations as well as the constructor for each class. (NB: You do not actually have to implement each method, simply give the declaration).

Vehicle Class:

```
public abstract class Vehicle {

     private String name;
     private int productionYear;

     public Vehicle(String name, int productionYear){
          this.name = name;
          this.productionYear = productionYear;
     }

     public abstract void changeGear();
     public abstract void accelerate();
     public abstract void brake();
}
```

Car Class:

```
public class Car extends Vehicle{

    private String engineSize;

    public Car(String name, int productionYear,
String engineSize) {
        super(name, productionYear);
        this.engineSize=engineSize;
    }


    @Override
    public void changeGear() {
        //...No need to implement
    }

    @Override
    public void accelerate() {
        //...No need to implement
    }

    @Override
    public void brake() {
        //...No need to implement
    }

    public boolean turnOnHeadlights(){
        return true;
    }

    public boolean turnOnWindscreenWipers(){
        return true;
    }

}
```

Bicycle Class:

```
public class Bicycle extends Vehicle {

    private int cadence;

    public Bicycle(String name, int productionYear,
int cadence) {
        super(name, productionYear);
        this.cadence = cadence;

    }
    @Override
    public void changeGear() {
        //...No need to implement
    }

    @Override
    public void accelerate() {
        //...No need to implement
    }
    @Override
    public void brake() {
        //...No need to implement
    }

    public boolean ringBell(){
        return true;
    }

    public boolean changeCadence(){
        return true;
    }


}
```