**Encapsulation**

Declaring variables, functions or methods as private to hide them when they are not needed.

Private float price;

**Inheritance**

When more than one class is arranged into a hierarchy so that one can inherit the properties of another.

Fruit = the super class

Apple = the subclass and inherits from fruit

public Class Apple extends Fruit {}

contained within the fruit could be a basic outline of what information would be needed to create a fruit object such as colour and size. Apple could inherit these properties as well as have its own properties specific to apples contained within the apple subclass.

**Abstraction**

When a superclass is made into an abstract class it ensures that its subclasses are implementing the functions that are necessary for their creation. For example the abstract superclass Fruit contains abstract methods for both colour and shape. That means that all subclasses of fruit (apple, orange) must have functions that return a colour and size. If a user fails to provide these functions, then the complier will not allow the program to run. In addition to this an abstract method cannot be instantiated on its own.

Allowed : Tickets sta = **new** Standard();

Not allowed: Tickets sta = **new** Tickets();

**Polymorphism**

When there are many methods of the same name but they have different outputs depending on what class they are used in.

Fruit app = **new** Apple();

for example in the fruit class their could be a return method called returnColour(). In the fruit superclass returnColour() would output this “fruit has colour” however in the Apple class returnColour() outputs this “apples are red”. The variable app has been created as type fruit, however it has been instantiated as type apple. So it when calling app.returnColour() it will return the colour of an apple.