Ask who has attended before.

Take attendance.

Introduce self, year, course studying.

Build rapport.

The purpose of PASS is to help students to better understand the course content and become independent learners. A PASS leader is someone who has previously taken the course and helps facilitate the session and guide students towards finding solutions of their own. PASS is not a tutorial and PASS leaders do not teach and cannot help you with your homework or assignments. PASS sessions are informal and you do not need to attend the whole sessions but can instead drop in and leave at any time.

**Objects** - Objects have states and behaviours. Example: A bicycle has states which can be thought of as attributes - current gear, current pedal cadence, current speed, etc. which can be declared in fields. A bicycle also has behaviours, which can be thought of as ways of changing the states of the bicycle - changing gear, changing pedal cadence, applying brakes. Behaviours are declared in methods.

**Class** - A class can be defined as a template or blueprint that describes the states and behaviours (fields and methods) of objects. You use a class to create as many objects of the same type as you need. Example: A Dog class can be used to create many Dog objects. An object is an instance of a class. The act of creating an object is called instantiation.

A class can contain any of the variable types:

* **Local variables:** Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialised within the method and the variable will be destroyed when the method has completed.
* **Instance variables:** Instance variables are variables within a class but outside any method. These variables are initialised when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
* **Class variables:** Class variables are variables declared within a class, outside any method, with the static keyword. In object-orientated programming with classes, a class variable is a variable defined in a class of which a single copy exists, regardless of how many instances of the class exist. Example: A class variable for a Tomato class might be "static String genus = "Solanum";" where every Tomato object has the same value for the "genus" variable. Static variables can be accessed from a class without actually creating an object of the class type. Example: If there were no Tomato objects, we can still access the static variable such as "System.out.println(Tomato.genus);" without having to instantiate a Tomato object first. This is because a static variable belongs to a class, not to any particular object of that class. If we want to prevent a static variable ever being changed we use "public final static String GENUS = "Solanum";" - notice the variable name is in uppercase, which is good programming practice.

**Array:** An array is a collection of variables of the same type. An array is considered an object, but there is dispute over whether array types are classes or not. You can create an array in two different ways:

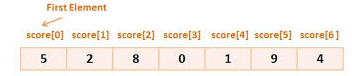
1. double[] score = new double[5];
   * This is declaring an array that contains 5 double values. If an array is not initialised, the default values will be used - for an array of doubles, each element of the array will be initialised to 0.0.
   * You can fill the array manually like so:
     + score[0] = 5;
     + score[1] = 7.5;
     + score[2] = 3.4;
     + score[3] = 8;
     + score[4] = 9.7;
   * You can fill the array using a for loop:
     + for(index = 0; index < score.length; index++)

score[index] = index + 2;

1. String[] dwarves = {“Dopey”, “Sleepy”, “Doc”, “Happy”};
   * This creates an array of 4 String values.

To access a value at a particular index, use the variable name and the index in brackets:

* variableName[index]
* System.out.println(“The third score is: “ + score[2]);
* System.out.println(“The first dwarf is: “ + dwarves[0]);



To find the size of an array, use the built in “length” instance variable for the array. The “length” variable is automatically set to the size of the array when an array is created.

* String[] dwarves = {“Dopey”, “Sleepy”, “Doc”, “Happy”};
* System.out.println(“Length of dwarves array: “ + dwarves.length);
  + Length of dwarves array: 4

**Enumerated Types:** A special data type that enables for a variable to be a set of predefined constants. The variable must be equal to one of the values that have been predefined for it.

* enum TypeName {VALUE1, VALUE2, VALUE3};
* enum ClothingSize {EXTRASMALL, SMALL, MEDIUM, LARGE, EXTRALARGE};
* enum Compass {NORTH, SOUTH, EAST, WEST};

A variable of an enumerated type can have a value that is either one of the values listed or null. You can set the value of a variable of an enumerated type as follows:

* TypeName aVariable = TypeName.VALUE2;
* ClothingSize mySize = ClothingSize.MEDIUM;
* ClothingSize yourSize = ClothingSize.EXTRALARGE;
* Compass direction = Compass.WEST;

You can then use these variables as normal:

* System.out.println("My clothing size is " + mySize);
  + My clothing size is MEDIUM
* System.out.println("We need to head " + direction);
  + We need to head WEST

When comparing two variables of an enumerated type, you can use the equals method or the == operator:

* if (mySize == yourSize)

**Inheritance:** The process where one class acquires the properties (methods and fields) of another. With the use of inheritance, the information is made manageable in a hierarchical order.

The class which inherits the properties of another is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

'extends' is the keyword used to inherit the properties of a class:

* class Super{

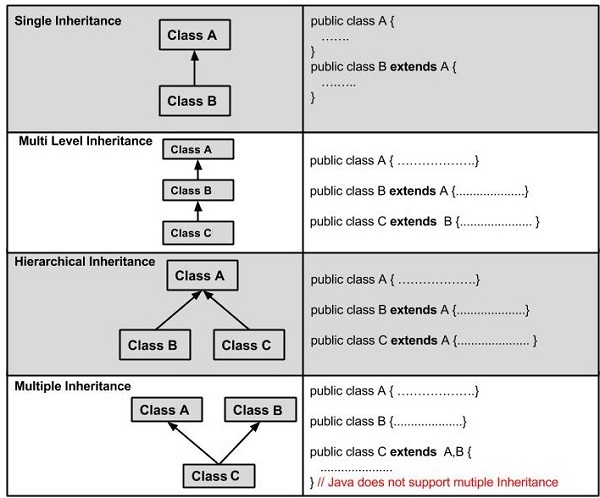
.....

}

* class Sub extends Super{

.....

}



The class ‘Object’ is an ancestor of every class, or in other words, every class is a derived class of the class ‘Object’. The class ‘Object’ has some methods that every class inherits, including the ‘equals’, ‘clone’, ‘getClass’ and ‘toString’ method. You will nearly always need to override the ‘equals’ and ‘toString’ methods in any class you define.

**Method Overriding:** A child class can override a method in its parent. An overridden method is essentially hidden in the parent class, and is not invoked unless the child class uses the super keyword within the overriding method. An overridden method is invoked at run time, no matter what data type the reference is that was used in the source code at compile time.

* Employee e = new Employee("Jane");

Person p = new Employee("Bob");

Assume that a method in the Person class was overridden in the Employee class. During compile time, the compiler sees the method in the Employee class for object 'e' and in the Person class for object 'p'. However, during run time the JVM will invoke the method in the Employee class for both object 'e' and 'p'.

**Polymorphism:** The ability of an object to take on many forms. The most common use of polymorphism occurs when a parent class reference is used to refer to a child class object. All Java objects are polymorphic since any object is a type of itself, as well as being a type of class Object. To access an object you must use a reference variable, which can only be one type. The type of the reference variable determines the methods that it can invoke on the object. A reference variable can refer to any object of its declared type or any subtype of its declared type.

In the below example, 'Deer' is an 'Animal', 'Vegetarian', 'Deer' and 'Object' type:

* public interface Vegetarian{}

public class Animal{}

public class Deer extends Animal implements Vegetarian{}

Therefore, the following declarations are valid and all refer to the same 'Deer' object:

* Deer d = new Deer();

Animal a = d;

Vegetarian v = d;

Object o = d;