# Class

- Class is a template/blueprint/logical format of the instance variables/properties and the methods (functions) put together. If Car is a class , then color is a property of the class , red can be the data of that property and run can be the behaviour ( method/function ) .

```
// sample java class for Car
public class Car{ // public and class are keywords.It’s public so that it’s visible to jvm. class keyword is to represent a java class.
  Integer numberOfWheels; // instance variable or property with integer data type
  String color; // color is the property with String which is an array(collection of similar datatypes ofcharacters ).
  public void run (){
   // code to determine the behaviour as to how the car runs either with high throttle or with less , something like that.
} // end of run method
} // end of Car class
```

## Object

- Object : It is the smallest manifestation of a class and is a real world existing entity of the class . It’s not a template / format but an existing thing/entity confirming to the format (class).

- If Book is a class , then the book of headfirst java in I&D library is one object of it . Another object can be a copy of headfirst java lying in a student’s home.

## Memory allocation to an object

- When a java program is made to run , then the jvm gets some memory allocated to it from the RAM by the operating system. This is called the  java heap space.
- RAM is the volatile memory unit where the CPU does the execution of programs and converts a program to a process . A process is a program during execution by the CPU ( or can be GPU also at times , depends ) . ROM is the part of the memory structure that has stable persistence and is not volatile ie. it remains saved even if system is restarted.
- To better understand RAM and ROM , we can correlate it with a scenario. If the files (programs ) are kept on a cabinet shelf ( ROM ) , then the desk on which you (CPU ) are working with the open file ( process ) is the RAM area. After your work is done , close the file and put it back to shelf (ROM) , else the data will be lost on the desk if left as is it say a wind can come and all the work can be lost.

## Object memory allocation

- Suppose there is a class Car with an instance variable color as follows :

```
public class Car{
   String color;
 }
```

- The object for the car can be created by using new keyword in a method :
  Car car = new Car(); // here car is the variable / reference to the car object that just got created on the heap space the moment " new Car() " got executed.
- The object that has been created is on the heap while the reference to that object , say a remote control to the object is on the stack . Stack is a part of the RAM allocated that internally leverages stack data structure ( last in first out ) , we will discuss it in detail in method that just follows .

## Method

- If the state of the object is maintained by the data of the instance variables , then the behaviour of the class is determined by the method ( function) written in the java class.

```
// sample java class for Car with instance variables and methods
public class Car{
  Integer numberOfWheels; // instance variable
  public void run(){
  // code to run  a car
 } // end of the method
}
```

# Method’s memory allocation

- Method runs on the stack . Stack is a part of the RAM allocated for execution/computation by CPU . A method put on the top of the stack is the currently executing method . It also stores the variables / data used inside the method . Additionally stack stores the reference variables to objects.
- Stack is the memory space allotted in the RAM for a single thread of execution . Thread we will talk later but as of now without threads our code is a single thread of execution.
- Currently executing method is on the top of the stack .
- Suppose that method m1() is internally calling method m2() .What internally happens is that first m1 is on the top of stack , and as m2 is called then m2 comes on the top of the stack. m2() remains on the top of the stack till method m2() is finished executing and then returns control to m1() . m1() is now top of the stack.

## Sample java class

- After we have talked of class , instance variables and methods , it’s time to say hello world in java. The code will be provided with vm and can be run on eclipse.

```
package javatutorial;
/*
 Class is the template or blueprint of data (not considered here) and methods( ex: show() ) put together.
 There can be only one class in a java file which can be having public keyword.
 The keyword "public" is given so that the class is visible to jvm.
 The JVM (java virtual machine) first executes public static void main method.
 */
public class HelloWorld {
	// HelloWorld is the name of the class and should be same as the name in the java file
	*/
```

## Hello World in java

```
public void show(){
                    // show is the name of the method
                    System.out.println(" HELLO WORLD ");
                    // System.out.println("") used to print on console in java language.
          }
   // starting point of execution of java code
          public static void main(String args[]){

                    System.out.println("execution starts here");

                    //JVM starts execution from public static void main method
                    HelloWorld helloWorld=new HelloWorld();
```

## Hello World in java

```
// HelloWorld class is getting instantiated,helloWorld is the instance.
                // Instance is the smallest manifestation of Class.Its a real world existing entity.
                helloWorld.show();
                // "show" method can be accessed by the instance ("helloWorld") created.
        }
}

// OUTPUT

execution starts here
HELLO WORLD
```

## Garbage Collection

- When does it happen ? Suppose there was an object on the heap and a reference on the stack pointing to that object . Let’s assume that the reference is now pointing to another object now , so it now renders the initial object dereferenced.
- The objects which are deferenced for a long time are destructed / destroyed time to time by jvm.
- Garbage collection can’t be enforced/guaranteed but it can be suggested to the jvm by writing System.gc() in a method where you suspect of some memory leak.
- Not to mention but since object creation happens on heap , likewise garbage collection also happens from heap memory.

# Interface basics

- It is a contract with declared methods in it which implemented by a java class makes sure that the contract is fulfilled ( or we can say that the methods declared ( no code/logic written) in the interface are appropriately overridden(filled with new java code/logic) in the implementing classes )
- ‘interface’ is the keyword used to mark an interface . Example interface :

```
public interface GenericMovement{
// method declared with no logic/code inside the method
public void move();
}
```

## Inheritance basics

- Inheritance means that if a class is extending another class then , the child class will inherit all the instance variables as well as the methods of the parent class . It's just like a parent child inheritance.

```
public class EngineeringStudent extends Student{

}
```

- Here the child class extends Student class by virtue of ‘extends’ keyword. Then it will have certains additional instance variables and methods like workInLab() and subjects like Physics , Chemistry and Maths particularly.

## Constructors

- Constructor is the method which has the same name as the class name with no return type and is responsible for memory allocation of the objects created on the heap as well as the value initialisation for the instance variables.
- Any method which returns nothing is having ‘void’ as the return type while if it returns the sum of two integers , then the method will have an Integer return type.

```
// Example code for constructors :
public class Book{
 public Book(){
     // instantiation of book objects happen here.
 }
}
```

# Default constructor

- **Default constructor** : If we know that we are going to instantiate bird objects and each object ie. each bird created will surely have two eyes , then in the constructor , we can mention that number of eyes should be two.

```
// sample default constructor
public Bird(){
  numberOfEyes = 2;
}
```

## Parameterized constructor

- Parameterized constructor : If we want to instantiate objects with custom values , then we use parameterized constructor . For example , if a shirt has to be made for a Shirt class , then custom values can be passed for it.
- ‘this’ keyword used in this code refers to the current object . If we don’t write ‘this’ keyword for a property it is implicitly used by compiler.

```
// code for parameterized constructor
public Shirt(Integer collarWidth,Integer shoulderWidth){
  this.collarWidth = collarWidth;
  this.shoulderWidth = shoulderWidth;
}
```

## Array of objects and memory allocation

- Java array is also an object in itself which contains objects of similar type or subtype. So if there is an array of five book objects , then in that case first of all a memory chunk is booked on the heap which is the size of one book object into five times.
- When we dig deep to understand the internal memory organisation we see that there are **five book objects adjacent to each other** in subscript style ordering . But point to note here is that all those five objects are in the same confined area on the heap allocated to the array.

## Array of objects and memory allocation steps

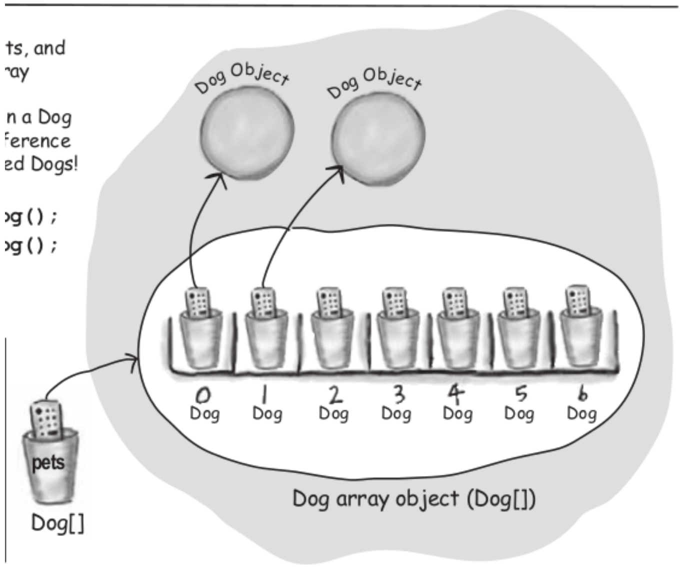
- The code for array initialization is as follows :

```
Dog[] pets; // [] marks an array of Dog objects and pets is the reference to the array of dog objects. At this
//point of time memory  is allocated as a chunk for size calculated for one dog object into seven times , but till
//yet not a single dog object really exists on  the heap

pets =  new Dog[7]; //  now seven dog objects are created on the heap in the memory chunk booked initially by
//the dogs array and the dog objects are placed side by side.
```

# Array of objects and memory allocation

- Diagrammatic representation of the previous array code and it’s memory allocation.



## Exercise

- Try ObjectInstantiation.java , HelloWorld.java and constructor related codes from com.musigma.javatraining.basic package at git repo : https://github.com/Mu-Sigma/poc.git in JavaTraining branch.
- Try out a code where you provide no constructor and afterwards provide a parameterized constructor . Just check whether now the default constructor works fine , play with it for some time.