Lab 03
Classes and Objects

Objectives:

1. What is Class?
2. What is an Object?
3. Class vs Object
4. Assigning Object References Variables
5. Method in class
6. Constructor
7. this Keyword
8. Stack Class
9. Lab tasks
10. Post lab questions

1: What is Class?

Class:

A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

- **Modifiers**: A class can be public or has default access
- **class keyword**: class keyword is used to create a class.
- **Class name**: The name should begin with an initial letter (capitalized by convention).
- **Body**: The class body surrounded by braces, { }.

.

***Syntax*:**

```
class <class_name>{
    field;
    method;
  }
```

2: What is an Object?

Objects are the basic units of object-oriented programming. A simple example of an object would be a person. Logically, you would expect a person to have a name. This would be considered a property of the person. You could also expect a person to be able to do something, such as walking or driving. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

- **State**: It is represented by attributes of an object. It also reflects the properties of an object.
- **Behavior**: It is represented by methods of an object. It also reflects the response of an object with other objects.
- **Identity**: It gives a unique name to an object and enables one object to interact with other object

***Syntax*:**

ClassName ReferenceVariable = new ClassName();

Example:

Car car; //declare reference to object

car = new Car(); //allocate a car object

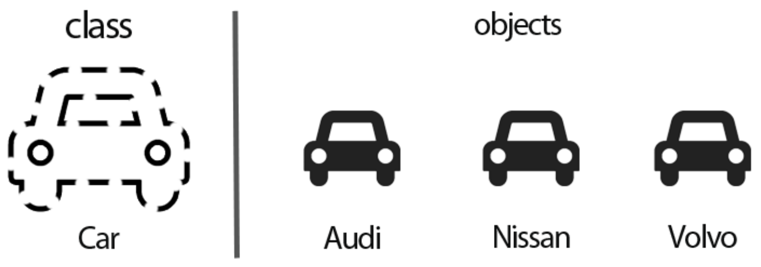
Creating an Object

As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.

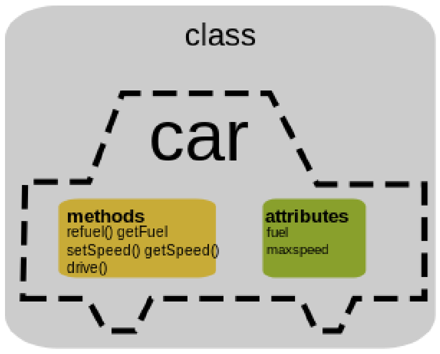
There are three steps when creating an object from a class −

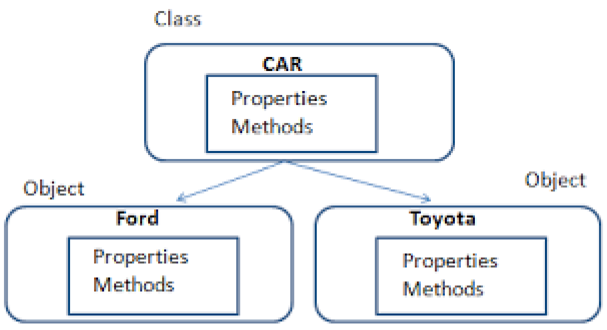
- **Declaration** − A variable declaration with a variable name with an object type.
- **Instantiation** − The 'new' keyword is used to create the object.
- **Initialization** − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

3: Class vs Object



Class containing attributes and methods.

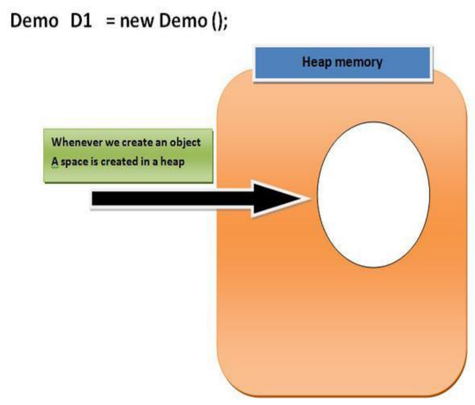




4: Assigning Object References Variables:

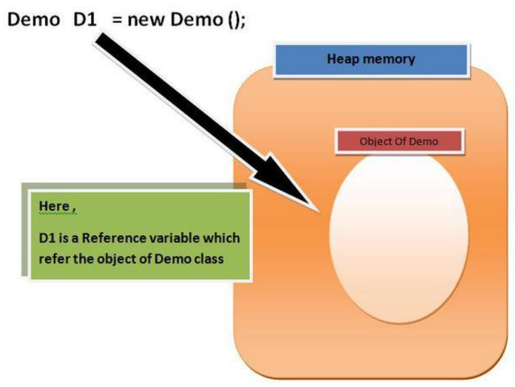
Before We get Started with the Reference variable we should know about the following facts.

When we create an object (instance) of class then space is reserved in heap memory.



```
Demo  D1  = new Demo ();
```

Then, We create a Pointing element or simply called Reference variable which simply points out the Object (the created space in a Heap Memory).


Demo D1 = new Demo ();
Heap memory
Object Of Demo
Here ,
D1 is a Reference variable which refer the object of Demo class

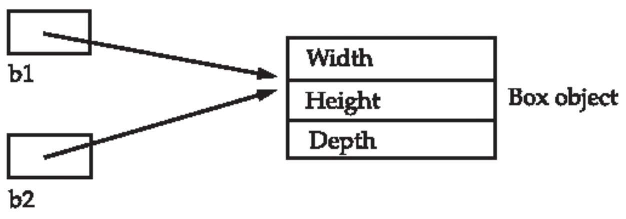
- Reference variable is used to point object/values.
- Reference variable can also store null value. By default, if no object is passed to a reference variable then it will store a null value.
- You can access object members using a reference variable using dot syntax.
- You can assign value of reference variable to another reference variable.
- Reference Variable is used to store the address of the variable.

Assigning Object Reference Variables does not:

1. Create Distinct Objects.
2. Allocate Memory.
3. Create duplicate Copy.

Consider below example:

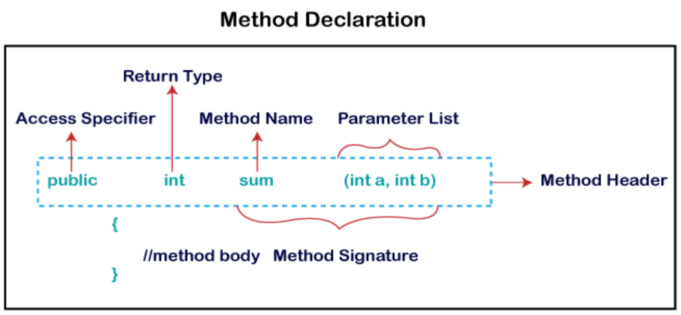
```
Box b1 = new Box();
Box b2 = b1;
```



- b1 is reference variable which contain the address of Actual Box Object.
- b2 is another reference variable
- b2 is initialized with b1 means – “b1 and b2” both are referring same object , thus it does not create duplicate object , nor does it allocate extra memory

5: Method in class

A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the reusability of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the easy modification and readability of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it. A function is a **combination of instructions** that are combined to achieve some result.



Syntax:

type name(parameter_list){
  //body of method
}

6: Constructor

A constructor in Java is a special method that is used to initialize objects. A constructor initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, *constructors have no explicit return type*. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes

A house needs a Builder and an object need a constructor in JAVA. For creating a new object/instance of a class you need a constructor. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes

Syntax:

```
class ClassName {
   ClassName() {
   }
}//constructor call

class GFG {
    public static void main (String[] args) {

        // creating an instance of Addition class
        Addition add = new Addition();

        // calling addTwoInt() method to add two integer using instance created
        // in above step.
        int s = add.addTwoInt(1,2);
        System.out.println("Sum of two integer values :"+ s);

    }
}
class Addition {
    int sum = 0;

    public int addTwoInt(int a, int b){

        // adding two integer value.
        sum = a + b;

        //returning summation of two values.
        return sum;
    }

}
```

Types of Constructors

There are three types of constructors in java

1. Default constructor
2. No-argument constructor.
3. Parameterized constructor

Default constructor

If you do not implement any constructor in your class, Java compiler inserts a default constructor into your code on your behalf.

If you implement any constructor then you no longer receive a default constructor from Java compiler.

No argument constructor

Constructor with no arguments is known as no-argument constructor. The signature is same as default constructor, however body can have any code unlike default constructor where the body of the constructor is empty.

Although you may see some people claim that that default and no-argument constructor is same but in fact they are not, even if you write public Demo() { } in your class Demo it cannot be called default constructor since you have written the code of it.

Syntax

```
class Demo
{
     public Demo()
     {
         System.out.println("This is a no argument constructor");
     }
     public static void main(String args[]) {
         new Demo();
     }
}
```

Parameterized constructor

Constructor with arguments(or you can say parameters) is known as Parameterized constructor.

In this example we have a parameterized constructor with two parameters id and name. While creating the objects obj1 and obj2 I have passed two arguments so that this constructor gets invoked after creation of obj1 and obj2.

```
public class Employee {

   int empId;
   String empName;

   //parameterized constructor with two parameters
   Employee(int id, String name){
       this.empId = id;
       this.empName = name;
   }
   void info(){
        System.out.println("Id: "+empId+" Name: "+empName);
   }

   public static void main(String args[]){
        Employee obj1 = new Employee(10245,"Ali");
        Employee obj2 = new Employee(92232,"saira");
        obj1.info();
        obj2.info();
   }
}
```

Copy Constrcutor

A copy constructor in a Java class is a constructor that creates an object using another object of the same Java class. This is specially helpful when we want to copy a complex object that has several fields .

```
import java.util.Scanner;

public class Student {

   private String name;

   private int age;

   public Student(String name, int age){
```

```java
        this.name = name;

        this.age = age;

    }

    public Student(Student std){

        this.name = std.name;

        this.age = std.age;

    }

    public void displayData(){

        System.out.println("Name : "+this.name);

        System.out.println("Age : "+this.age);

    }

    public static void main(String[] args) {

        Scanner sc =new Scanner(System.in);

        System.out.println("Enter your name ");

        String name = sc.next();

        System.out.println("Enter your age ");

        int age = sc.nextInt();

        Student std = new Student(name, age);

        System.out.println("Contents of the original object");

        std.displayData();

        System.out.println("Contents of the copied object");

        Student copyOfStd = new Student(std);

        copyOfStd.displayData();

    }

}
```

What is Garbage Collection in Java?

Garbage Collection in Java is a process by which the programs perform memory management automatically. The Garbage Collector(GC) finds the unused objects and deletes them to reclaim the memory. In Java, allocation of objects is achieved using the new operator that uses some memory and the memory remains allocated until there are references for the use of the object.

When there are no references to an object, it is assumed to be no longer needed, and the memory, occupied by the object can be reclaimed. ***There is no explicit need to destroy an object as Java handles the de-allocation automatically.***

But if you want to explicitly release the memory then there are various ways in which the references to an object can be released to make it a candidate for Garbage Collection.

By making a reference null

```
Student student = new Student();
student = null;
```

By assigning a reference to another

```
Student studentOne = new Student();
Student studentTwo = new Student();
studentOne = studentTwo; // now the first object referred by studentOne is available
for garbage collection
```

7: The this Keyword

The this keyword refers to the current object in a method or constructor.

The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

this can also be used to:

- Invoke current class constructor
- Invoke current class method
- Return the current class object
- Pass an argument in the method call
- Pass an argument in the constructor call

Syntax:

```
public class Main{
```

```
int x;

// Constructor with a parameter
public Main(int x) {
  this.x = x;
}

// Call the constructor
public static void main(String[] args) {
  Main myObj = new Main(5);
  System.out.println("Value of x = " + myObj.x);
}
```

Constructor of the current class can be accessed by using keyword this()

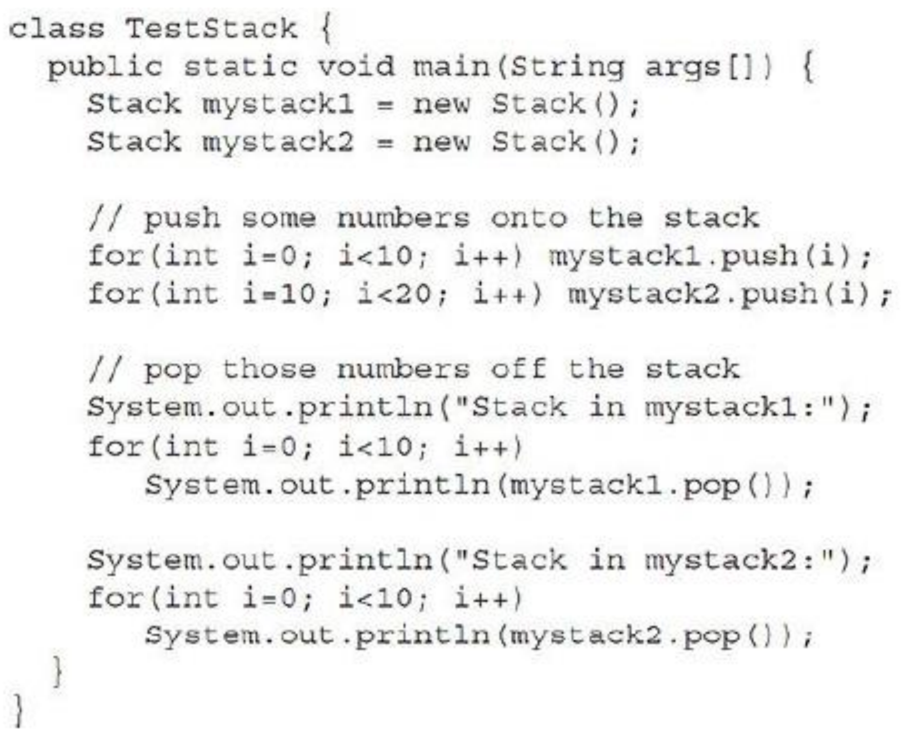
8: The Stack Class

A stack stores data using first-in, last-out ordering. That is, a stack is like a stack of plates on a table—the first plate put down on the table is the last plate to be used. Stacks are controlled through two operations traditionally called push and pop. To put an item on top of the stack, you will use push. To take an item off the stack, you will use pop. As you will see, it is easy to encapsulate the entire stack mechanism.

Here is a class called Stack that implements a stack for up to ten integers:

```
// This class defines an integer stack that can hold 10 values
class Stack {
  int stck[] = new int[10];
  int tos;

  // Initialize top-of-stack
  Stack() {
    tos = -1;
  }

  // Push an item onto the stack
  void push(int item) {
    if(tos==9)
      System.out.println("Stack is full.");
    else
      stck[++tos] = item;
  }

  // Pop an item from the stack
  int pop() {
    if(tos < 0) {
       System.out.println("Stack underflow.");

       return 0;
    }
    else
      return stck[tos--];
  }
}
```

The class TestStack, shown here, demonstrates the Stack class. It creates two integer stacks, pushes some values onto each, and then pops them off.



```
class TestStack {
  public static void main(String args[]) {
    Stack mystack1 = new Stack();
    Stack mystack2 = new Stack();

    // push some numbers onto the stack
    for(int i=0; i<10; i++) mystack1.push(i);
    for(int i=10; i<20; i++) mystack2.push(i);

    // pop those numbers off the stack
    System.out.println("Stack in mystack1:");
    for(int i=0; i<10; i++)
       System.out.println(mystack1.pop());

    System.out.println("Stack in mystack2:");
    for(int i=0; i<10; i++)
       System.out.println(mystack2.pop());
  }
}
```

This program generates the following output:

```
Stack in mystack1:          Stack in mystack2:
9                           19
8                           18
7                           17
6                           16
5                           15
4                           14
3                           13
2                           12
1                           11
0                           10
```

Methods calls are implemented through stack. Whenever a method is called a stack frame is created within the stack area and after that the arguments passed to and the local variables and value to be returned by this called method are stored in this stack frame and when execution of the called method is finished, the allocated stack frame would be deleted. There is a stack pointer register that tracks the top of the stack which is adjusted accordingly.

Lab Tasks

Marks : 10, All Questions carry equal marks

Exercise 1 — Employee.java

Write a program that would print the information (name, year of joining, salary, address) of three employees by creating a class named 'Employee'. The output should be as follows:

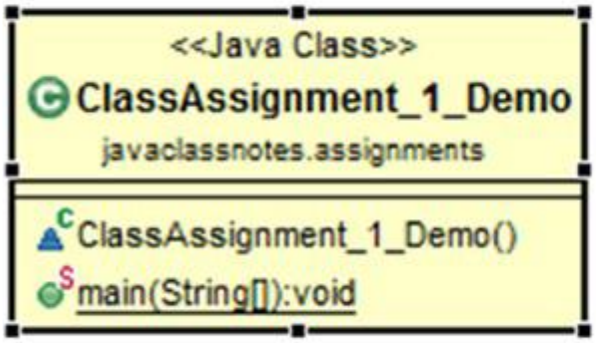
```
Name      Year of joining     Address
Robert         1994          64C- WallsStreat
Sam            2000          68D- WallsStreat
John           1999          26B- WallsStreat.
```

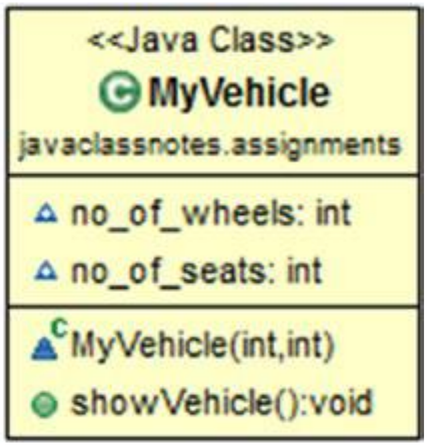
Exercise 2 — EmployeeInfo.java

Write a program by creating an 'EmployeeInfo' class having the following methods and print the final salary.
1 - 'getInfo()' which takes the salary, number of hours of work per day of employee as parameter
2 - 'AddWork()' which adds $5 to salary of employee if the number of hours of work per day is more than 6 hours.

Exercise 3 — Vehicle.java

Create a class Vehicle. The class should have two fields-no_of_seats and no_of_wheels. Create two objects-Motorcycle and Car for this class. Your output should show the descriptions for Car and Motorcycle.





Exercise 4 **Temprature.java**

Write a program that inputs temperature in Celsius and converts it into Fahrenheit.

Fahrenheit=1.8 x Celsius +32

Exercise 5 **Year.java**

Write a program that inputs the year a person is born in and returns the age of the person.

Example: Person born in 1991 then age is 30

Exercise 6 **ShipTest.java**

Develop an application that computes the total cost of an order.

Sample output:

```
Number of Bags Ordered: 52
The Cost of Order: $ 286.00

Boxes Used:
          2 Large  - $3.60
          1 Medium - $1.00
          1 Small  - $0.60

Your total cost is: $ 291.20
```

Post lab questions to ponder

1. Can constructors be private?
2. Can a non-static method access a static variable or call a static method?
3. State differences between java constructor and java methods

END