Subject - Object Oriented Programming

Text Book - Java - The Complete Reference By **Herbet Shield**, **Tata Mc-Graw Hill Publication**

OOP Languages

- C++
- JAVA
- C#
- Python
- R
- PHP
- etc

CHAPTER -5
CLASSES AND OBJECTS

Outline

- Basics of Object and Class in C++
- Dot and new operator
- Access Modifiers
- Constructors and their types
- Constructor Overloading
- This Keyword
- Method Overloading
- Passing object as parameter
- Inner class
- Garbage Collection
- Finalize method

Class and Object in OOP

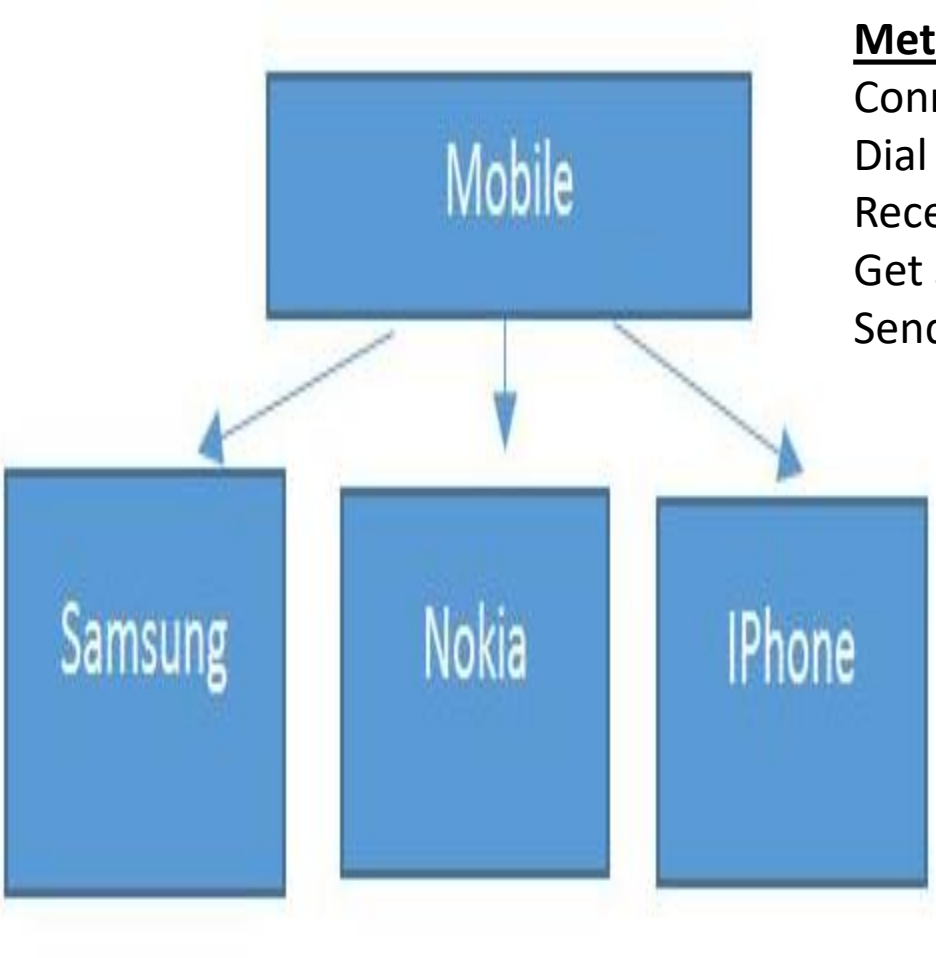
Class and Objects

Property
-Color
-Company Name
-IMEI CODE
-SIM CARD NO
-IsSINGLESIM

Methods
ConnectBlueTooth
Dial call
Receive call
Get SMS
Send SMS

Class and Objects

Property
-Color
-Company Name
-IMEI CODE
-SIM CARD NO
-IsSINGLESIM

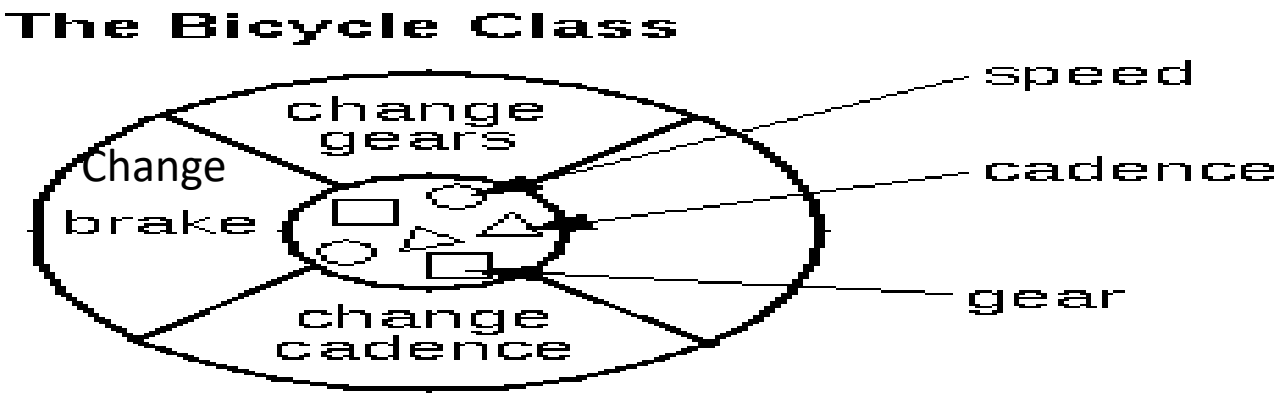


Methods
ConnectBlueTooth
Dial call
Receive call
Get SMS
Send SMS

What is Class and Object?

- In the real world, you often have many objects of the same kind.
- For example, **your bicycle**
- We can say your bicycle is **one bicycle among all bicycles**.
- All bicycles **have same properties and methods like**.

Properties



- Using object-oriented terminology, **we say that your bicycle is an object of the class bicycles.**

Class and objects

HUMAN

Object: Man

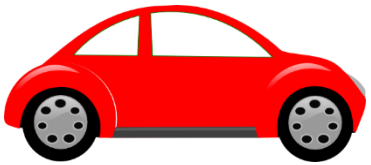
Property

Name
Age
Weight

Methods

Eat
Sleep
Walk

CAR



Object: Car-Hondacity

Property

Company
Color
Fuel type

Methods

Start
Drive
Stop

BANK ACCOUNT

Bank Account

Object: Account

Property

AccountNo
HolderName
AccountType

Methods

Deposit
Withdraw
Transfer

What is Class and object?

- If **different kinds of objects have similar characteristic** so we can represent them using class.
- **Example CAR Class**

Different **object of car class** like Baleno ,Dezire etc..

- **Example Human class**

Different **objects of Human class** – man ,woman ,child

- **Example Pencil Class**

Different **object s of Pencil class** Natraj , Apsara etc…

Class

- **Consider Example…**

Example class

- **Fruits**

Now Objects of this class is

- Fruits → Mango,banana ,Apple etc.

What is an Class and Object?

Pen

Laptop

Projector

Example objects…

Attributes/Properties and Methods

HUMAN

Object: Man

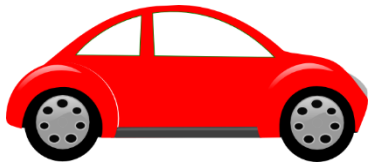
Attributes

Name
Age
Weight

Methods

Eat
Sleep
Walk

CAR



Object: Car-Hondacity

Attributes

Company
Color
Fuel type

Methods

Start
Drive
Stop

BANK ACCOUNT

Bank Account

Object: Account

Attributes

AccountNo
HolderName
AccountType

Methods

Deposit
Withdraw
Transfer

Class

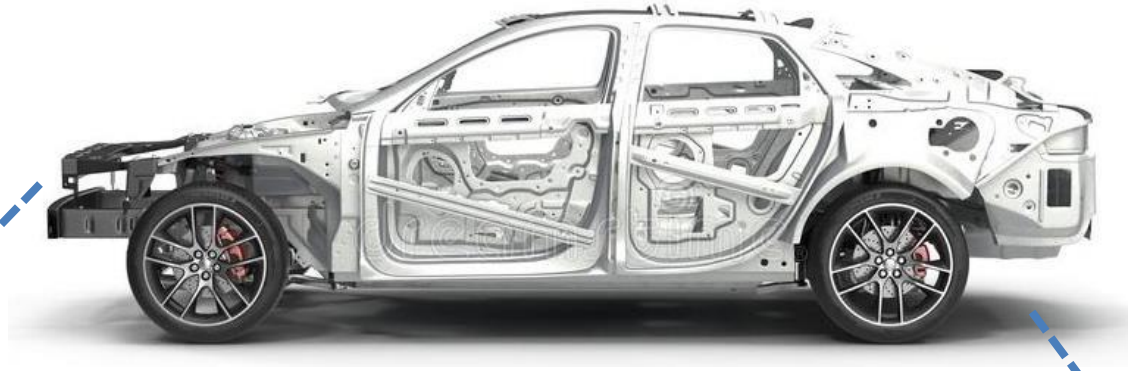
DEFINATION

A class is a **blueprint or template** that defines the **variables/attributes/properties** and methods **common to all objects of a certain kind.**

Class car

Class: Car

Class: Car



Properties (Describe)
Company
Model
Color
Mfg. Year
Price
Fuel Type
Mileage
Gear Type
Power Steering
Anti-Lock braking system

Methods (Functions)
Start
Drive
Park
On_break
On_lock
On_turn

Objects of Class Car

Honda City

Hyundai i20

Sumo Grand

Mercedes E class

Swift Dzire

Syntax of creating class

```
class classname {
    type instance-variable1;
    type instance-variable2;
  // ...
  type instance-variableN;

  type methodname1(parameter-list) {
    // body of method
  }
  type methodname2(parameter-list) {
    // body of method
}
// ...
type methodnameN(parameter-list) {
    // body of method
}
```

Class in OOP

- A **class** is a **blueprint** or **template** that describes properties and methods common to all objects of certain kind.

Example:

```
class car
{     int price;  // instance variables
     float mileage;

     void start(){//body }
     void drive(){ //body} //member functions
}
```

- In above example class name is **car**.

Specifying Class

How to declare / write class ?

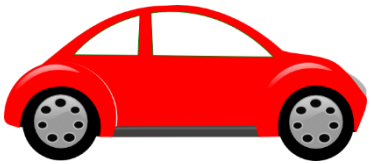
How to create an object (instance/variable of class)?

How to access class members ?

How to declare / write class ?

Class

```
class car
{

    int price;
    float mileage;

    void start(){}
    void drive(){}
 };
```



Car
Attributes/Property Price Mileage
Methods Start Drive

How to create an object ?

Syntax:

```
className  objectVariableName=new className();
```

Class

```
class car
{

    int price;
    float mileage;

    void start(){}
    void drive(){}
 };
```

Object

```
car c1=new car();
c1.start();
```

Object in OOP

- An **object** is an instance of a class
- An **object** is a variable of type class

Class

```
class car
{

    int price;
    float mileage;

    void start();
    void drive();
 };
```

Object

```
car c1=new car();
c1.start();
c1.drive();
```

Objects in OOP

Important Points

1. An **object** is an instance of a class
2. **An object is variable of type class.**
3. **An object is representative of that class.**
4. **Object is runtime Entity, it is created at runtime.**
5. **All the properties and methods of class can be accessed through object.**

New operator

- The Java **new operator is used to create an instance /object of the class**.
- In other words, **it instantiates a class by allocating memory for a new object** and returning a reference to that memory.
- Syntax

```
Car  c1 =  new  Car();
```

Points to remember

- It is used to **create the object.**
- It allocates the **memory at runtime.**
- All objects **occupy memory in the heap area.**
- It invokes the **constructor of the class.**

the dot operator

- Using . (dot) Operator we can access **data members/attributes/properties** and **methods** of class.

```
c1.price;  //  access class property/attributes
c1.mileage;   //  access class property/attributes

c1.start();     //  access class methods

c1.drive();     //  access class methods
```

JAVA FIRST PROGRAM

Print Welcome JAVA in OUTPUT.

```
class NewExample1
{


    public static void main(String[] args)
     {
            System.out.println("Welcome  JAVA");
      }

 }
```

Program: class, object

- Write a JAVA program to create class **Test** having data members **mark and spi.**
- Create member functions `SetData()` and `DisplayData()` to demonstrate class and objects.

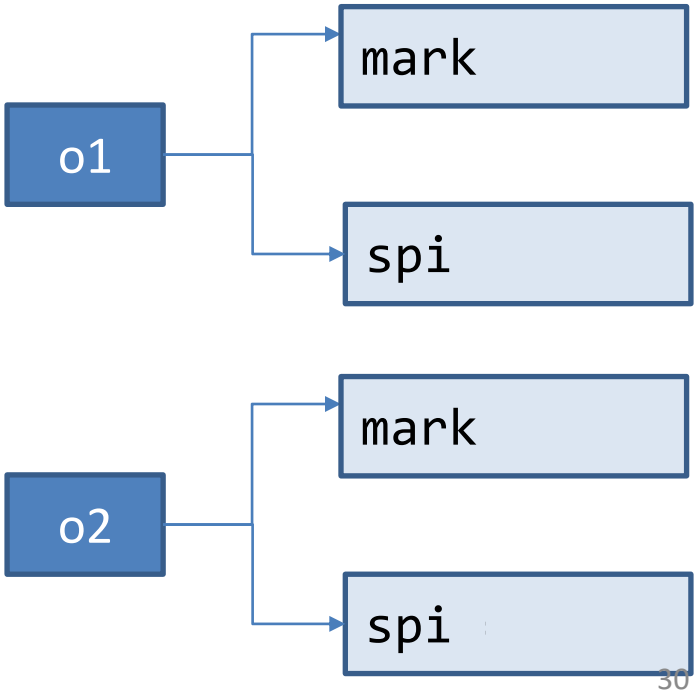
Program: class, object

```
class Test
{

    int mark;
    float spi;

    void SetData()
    {
      mark = 270;
      spi = 6.5;
    }
    void DisplayData()
    {
      System.out.println("Mark::"+mark)
      System.out.println("Spi::"+spi)
    }
} ;
```

```
Class A
{
   public static void main(String
args[])
    {
        Test o1=new Test();
        o1.SetData();
         o1.DisplayData();
   }

}
```

```
import java.util.Scanner;
class Test
{
      int mark;
     float spi;

    void SetData()
    {
     Scanner sc= new    Scanner(System.in);
     System.out.println("Enter  Marks::");
     mark= sc.nextInt();
     System.out.println("Enter  Marks::");
     mark= sc.nextFloat();
    }
    void DisplayData()
    {
System.out.println("Mark="+mark);
System.out.println("SPI="+spi);
    }
} ;
```

```
Class  A {
public static void main(String  args[])
{
      Test o1=new Test();
       o1.SetData();
        o1.DisplayData();

      Test o2;
       o2.SetData();
        o2.DisplayData();
    }
}
```



Program 2 : class, object

- Write a JAVA program to create **class Car** having **data members Company and Top_Speed.**
- Create member functions **`SetData()`** and **`DisplayData()`** and **create two objects of class Car**.

- Example in Netbeans –JAVA_EXAMPLES

Program 2: class, object

- Write a JAVA program to create **class Emp1** having data members Emp_Name, Salary, Age.
- Create member functions **`SetData()`** and **`DisplayData()`**.
- Create two objects of class Employee

Access Modifiers

- As the name suggests access modifiers in Java helps to **restrict the scope of a class, constructor , variable , method or data member.**
- There are four types of access modifiers available in java:

Access Modifiers

- public
- private
- protected
- default

1. Private Members

- **Private** members of the class **can be accessed within the class.**
- They **cannot be accessed outside the class** or from other programs, **not even from inherited class.**
- If **you try to access private data from outside of the class**, compiler throws error.
- This feature in OOP is known as **Data hiding / Encapsulation**.

Example Private Data Members

```
class A
{
   private int data=40;
   private void msg()
    {    System.out.println("Hello java");      }
}

class Simple{
       public static void main(String args[])  {
        A obj=new A();
        System.out.println(obj.data);//Compile Time Error  //private variable
        obj.msg();//Compile Time Error  because private method
        }
   }
```

2. Public Members

- The **public** keyword makes **variables and functions public**.
- Public members of the class are accessible **by any program from anywhere.**
- Class members that allow **manipulating or accessing the class data outside of class are made as public.**

Example Public

A.java

```
//save by A.java


package pack;
class A{
public void msg()
{ System.out.println("Hello");}
}
```

Example Public

```
Output:Hello
```

```
//save by B.java

package mypack;
import pack.*;

class B{
 public static void main(String args[]) {
   A obj = new A();
   obj.msg();
  }
}
```

Output ==?

3. Default

- If **you don't use any modifier,** it is treated as **default** by default.
- The **default modifier is accessible only within package.**
- It **cannot be accessed** from outside the package.
- It **provides more accessibility than private.**

Example

```
//save by A.java
   package pack;
   class A{
    void msg(){  System.out.println("Hello");}
   }
```

Example

Note:-In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

```
//save by B.java
package mypack;
import pack.*;
class B{
 public static void main(String args[]){
  A obj = new A(); //Compile Time Error
 obj.msg(); //Compile Time Error
 }
}
```

4.Protected

- The **protected access modifier** is accessible **within package and outside the package but through inheritance only.**
- The protected access modifier can be applied on the **data member and method** .
- It **can't be applied on the class.**
- It provides **more accessibility than the default modifier.**

Example

```
//save by A.java
package pack;
class A{
protected void msg(){System.out.println("Hello");}
}
```

Example Continue.

```
//save by B.java
package mypack;
import pack.*;

class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
   }
}
```

Output--Hello

Summary of Access modifiers

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

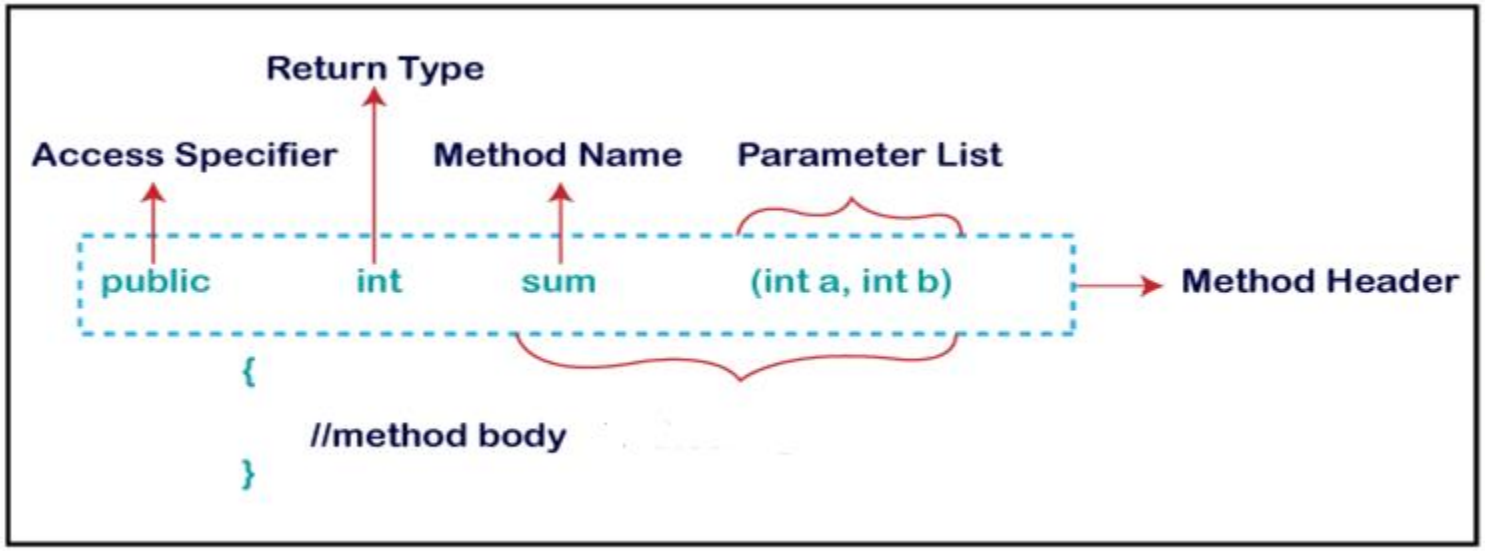
Working with Methods/Function

- In general, a **method** is a way to perform specific task.
- Similarly, the **method in Java** is a collection of instructions that performs a specific task.
- It provides the **reusability of code.**
- We can also easily modify code using **methods**

Introducing methods

- Syntax:

```
access_specifier return_type method_name(argument_list)
{
    // code OR BODY
}
```



Example 1 –Method/Function

```
  class Addition
{               public static void main(String[] args)
                {
                                 int a = 19;
                               int b = 5;
                               //method calling
                                A a1=new A();
                               int c = a1.add(a, b);   //a and b are actual parameters
                               System.out.println("The sum of a and b is= " + c);
                }
 };
class  A
{
                public  int add(int n1, int n2)   //n1 and n2 are formal parameters
                {
                                int s;
                               s=n1+n2;
                               return s; //returning the sum
                }
}
```

Output:

```
The sum of a and b is= 24
```

Example 2 (method)

```
C:\WINDOWS\system32\cmd.exe

D:\DegreeDemo\PPTDemo>javac Demo.java

D:\DegreeDemo\PPTDemo>java Demo
Samsung
```

```java
public class SmartPhone {
    String manufacturer;
    public String getManufacturer(){
        return manufacturer;
    }
    public void setManufacturer(String a){
        manufacturer = a;
    }
};
```

```java
public class Demo {
    public static void main(String args[]) {
        SmartPhone sp = new SmartPhone();
        sp.setManufacturer("Samsung");
        String name = sp.getManufacturer();
        System.out.println(name);
    }
}
```

Constructor

Constructor

- A **constructor** is a special method that is called automatically when an object is created.
- Here are the key differences between a constructor and a method:

OR Properties of Constructor

- A constructor **doesn't** have a **return** type. not even void.
- The **name** of the constructor **must** be **same** as the name of the **class**.
- Unlike methods, **constructors** are not considered **members of a class**.
- A constructor is called **automatically** when an object is **created**.
- Constructors cannot be private.
- A constructor can be overloaded.
- Constructors cannot return a value.
- By default it is public.

- Syntax :

```
public ClassName (parameter-list)
{
    statements...
}
```

Types of Java constructors

There are two types of constructors in Java:

- Default constructor (no-arg constructor)
- Parameterized constructor

1.Default constructor (no-arg constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.
- Example

```
class Bike1{
        //creating a default constructor
         Bike1(){ System.out.println("Bike is created");}
         //main method
         public static void main(String args[]) {
         //calling a default constructor
                    Bike1 b=new Bike1();
         }
}
```

Output:

```
Bike is created
```

Default constructor-Example 2

```
class Test
{
    int a, b;
     Test () {
          System.out.println ("I AM FROM DEFAULT CONSTRUCTOR...");
           a=10;
           b=20;
          System.out.println ("VALUE OF a = "+a);
         System.out.println ("VALUE OF b = "+b); }
 };
 class TestDemo { public static void main (String [] args)
{ Test t1=new Test (); }
};
```

2. Parameterized constructor

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

Example 1

Output:

```
111 Karan
222 Aryan
```

```
//Java Program to demonstrate the use o
f the parameterized constructor.
class Student4
{
        int id;
        String name;
//creating a parameterized constructor
    Student4(int i,String n)
   {
        id = i;
        name = n;
    }
  //method to display the values
  void display()
   {System.out.println(id+" "+name);
   }
```

```
public static void main(String args[]
) {

Student4 s1 = new Student4(111,”KARAN”;);
Student4 s2 = new Student4(222,"Aryan");
      s1.display();
      s2.display();
   }
}
```

Example 2

```
class Test
{
    int a, b;
    Test (int n1, int n2)
    {
        System.out.println ("I AM FROM PARAMETER CONSTRUCTOR...");
        a=n1;
        b=n2;
        System.out.println ("VALUE OF a = "+a);
        System.out.println ("VALUE OF b = "+b);
    }
};
```

```
class Test2
{
    public static void main (String k [])
    {
        Test t1=new Test (10, 20);
    }
};
```

Constructor Overloading

- Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.
- They are arranged in a way that each constructor performs a different task.
- They are differentiated by the compiler by the number of parameters in the list and their types.

Example

```
class Student5{
   int id;
   String name;
   int age;
  Student5(int i,String n)
  {
      id = i;
     name = n;
   }
Student5(int i,String n,int a)
 {
            id = i;
            name = n;
            age=a;
   }
```

Output:

```
111 Karan 0
222 Aryan 25
```

```
void display()
{ System.out.println(id+" "+name+"
"+age);
}
 public static void main(String args[])
{
 Student5 s1 = new Student5(111,"Karan");

Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
   s2.display();
 }//main

}//class end
```

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 .**
- If you omit the keyword in the example above, the output would be "0" instead of "5".

Example this keyword

```
5
```

```
public class MyClass
{
                int  x;

              // Constructor with a parameter
      public MyClass(int  x)
      {
            this.x = x;
    }
    // Call the constructor
  public static void main(String[] args)
{
    MyClass myObj = new MyClass(5);
    System.out.println("Value of x = " + myObj.x);
  }
}
```

Method overloading

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
- If we have to perform only one operation, having same name of the methods increases the readability of the program.
- Different ways to overload the method
- There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

1) Method Overloading: changing no. of arguments

- In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

Example

add(int, int)

add(int, int, int)

Example

```
class DisplayOverloading
{
    public void disp(char c)
    {
         System.out.println(c);
    }
    public void disp(char c, int num)
    {
         System.out.println(c + " "+num);
    }
}
class Sample
{
   public static void main(String args[])
   {
       DisplayOverloading obj = new DisplayOverloading();
       obj.disp('a');
       obj.disp('a',10);
   }
}
```

Output:

```
a
a 10
```

2) By changing the data type

- No of arguments are same ,but data types is different.
- In this example, we have created two methods that differs in data type.

Example

Add(int ,int);

Add(int ,float);

Example

```
class DisplayOverloading2
{
    public void disp(char c)
    {
        System.out.println(c);
    }
    public void disp(int c)
    {
       System.out.println(c );
    }
}

class Sample2
{
    public static void main(String args[])
    {
        DisplayOverloading2 obj = new DisplayOverloading2();
        obj.disp('a');
        obj.disp(5);
    }
}
```

Output:

```
a
5
```

Passing Object as Parameter

- We can pass object like any other variable as argument to a method in Java.

Example

```
class Rectangle {
    int length;
    int width;

    Rectangle(int l, int b) {
        length = l;
        width = b;
    }

    void area(Rectangle r1) {
        int areaOfRectangle = r1.length * r1.width;
        System.out.println("Area of Rectangle : "
                                + areaOfRectangle);
    }
}

class RectangleDemo {
    public static void main(String args[]) {
        Rectangle r1 = new Rectangle(10, 20);
        r1.area(r1);
    }
}
```

Output of the program :

```
Area of Rectangle : 200
```

Nested Class / Inner Class

- Java inner class or nested class is a class which is **declared inside** the **class or interface**.
- Inner classes are used to **logically group** classes and interfaces in one place so that it can be more **readable** and **maintainable**.
- Additionally, it **can access** all the **members of outer class** including private data members and methods.
- Advantages of Inner Class
  - It can **access** all the **members** of Outer Class
  - It is **more readable** and **maintainable**
  - It requires **less code** (Code Optimization)

Example

MyMainClass.java

```
class OuterClass {
  int x = 10;

  class InnerClass {
    int y = 5;
  }
}

public class MyMainClass {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}
```

Result:

```
15
```

Access Outer Class From Inner Class

- One advantage of inner classes, is that they can access attributes and methods of the outer class.

Example

MyMainClass.java

```
class OuterClass {
  int x = 10;

  class InnerClass {
    public int myInnerMethod() {
      return x;
    }
  }
}

public class MyMainClass {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.myInnerMethod());
  }
}
```

Result:

```
10
```

Garbage Collection

- When JVM starts up, it creates a heap area which is known as runtime data area.
- This is where all the objects (instances of class) are stored. Since this area is limited, it is required to manage this area efficiently by removing the objects that are no longer in use.
- The process of removing unused objects from heap memory is known as **Garbage collection** and this is a part of memory management in Java.
- Languages like C/C++ **don't** support automatic garbage collection, however in java, the garbage collection is automatic.
- **So, java provides better memory management.**

- In java, garbage means unreferenced objects.

OR IN SHORT

- Garbage Collection is process of reclaiming the runtime unused memory automatically. **In other words, it is a way to destroy the unused objects.**

Advantages

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

finalize() method

- The **finalize()** is belong to **java.lang.Object** class.
- finalize() method is called by the **garbage collector** on an object when garbage collector determines that there are no more references to the object.
- finalize() might be used to make sure that some **system resource** not managed by the JRE is **properly released**.
- A subclass **overrides** the finalize method to **dispose** of **system resources** or to perform other **cleanup**.

```
protected void finalize() throws Exception
{
        // code to dispose resources here
}
```

Example – finalize() method

- We can also manually call finalize method by activating garbage collector using **System.gc()** method.

```
public class DemoFinalize
{
    public static void main(String[] args)
    {
        DemoFinalize obj = new DemoFinalize();
        obj=null; // to remove the reference of the object
        System.gc();
    }
    public void finalize()
    {
         System.out.println("Garbage Collected");
    }
}
```

C:\WINDOWS\system32\cmd.exe

```
D:\DegreeDemo>javac DemoFinalize.java

D:\DegreeDemo>java DemoFinalize
Garbage Collected
```

Thank You