Object and Classes

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts −

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Data Hiding

For now, we will look into the concepts - **Classes and Objects**.

- **Object** − Objects have **states** and **behaviors**. **Example**: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An **object** is an instance of a class.
- **Class** − A class can be defined as a **template/blueprint** that describes the **behavior/state** that the object of its type support.

Objects in Java

Let us now look deep into what are objects. If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behavior.

If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging the tail, running.

If you compare the software object with a real-world object, they have very similar characteristics.

Software objects also have a state and a behavior. A software object's state is stored in **fields** and **behavior** is shown via methods.

Classes in Java

A class is a blueprint from which individual objects are created.

Following is a sample of a class.

Bishal Patel

Example

```
public class Dog {
   String breed;
    int age;
   String color;

   void barking() {
   }

   void running() {
   }

   void sleeping() {
   }

}
```

A class can contain any of the following variable types.

- **Local variables** − Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized 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 initialized 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.

A class can have any number of methods to access the value of various kinds of methods. In the above example, **barking(), hungry()** and **sleeping()** are methods.

```
class Student{
  int id;
 String name;

 public static void main(String args[]){
  Student s1=new Student()
  System.out.println(s1.id);
```

Bishal Patel

```
  System.out.println(s1.name);
 }
}
```

Another way

```
class Student{
  int id;
 String name;
  }
class Abcd{
 public static void main(String args[]){
  Student s1=new Student()
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
```

Method

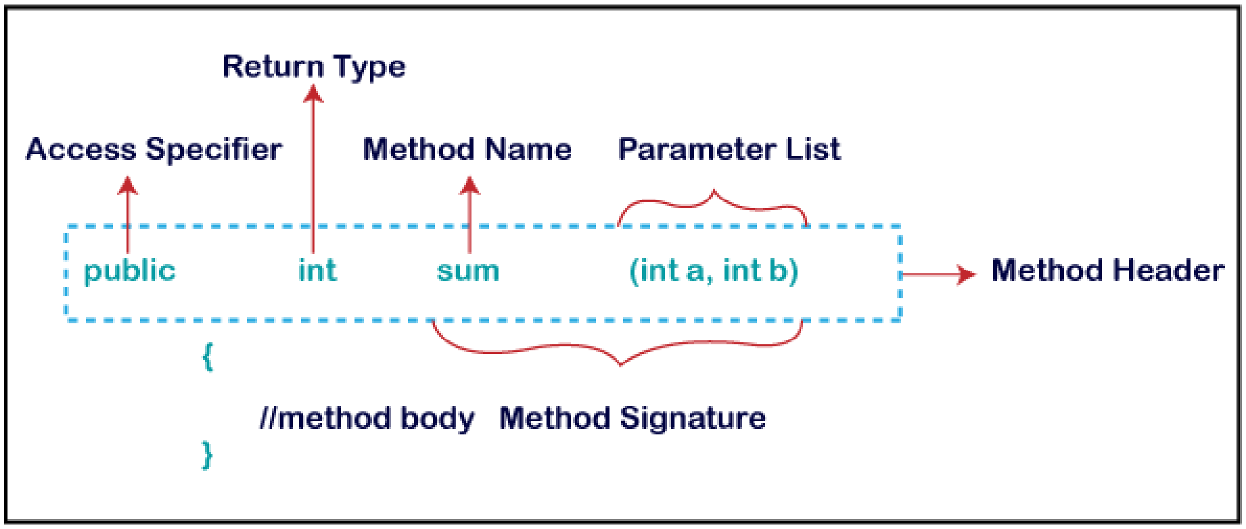
A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.

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.

Method Declaration

We can declare the method in the following ways. There are five component of method available in java Language.

Bishal Patel



Syntax

Access_Specifier return_Type method_Name (List of Parameter)
{
// body
}

Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides four types of access specifier:

- **Public:** The method is accessible by all classes when we use public specifier in our application.
- **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
- **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

Bishal Patel

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be subtraction (). A method is invoked by its name.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces

Naming a Method

While defining a method, remember that the method name must be a verb and start with a lowercase letter.

If the method name has more than two words, the first letter of each word must be in uppercase except the first word.

For example:

Single-word method name: **sum (), area ()**

Multi-word method name: areaOfCircle (), stringComparision ()

Types of method

- Pre-defined Method
- User-defined Method

Bishal Patel

```
import java.util.Scanner;
public class EvenOdd
{
        public static void findEvenOdd (int num)
```

```
         {
             if(num%2==0)
                  System.out.println(num+" is even");
            else
                 System.out.println(num+" is odd");
         }

   public static void main (String args[])
   {
       Scanner scan=new Scanner(System.in);
       System.out.print("Enter the number: ");
        int num=scan.nextInt();
         findEvenOdd(num);
    }
}
```

Another ways

```
class EvenOdd
{
            public static void findEvenOdd (int num)
         {
             if(num%2==0)
                  System.out.println(num+" is even");
            else
                 System.out.println(num+" is odd");
         }
}



Public class EvenOdd1
{
   public static void main (String args[])
   {
```

Bishal Patel

```
        Scanner scan=new Scanner(System.in);
        System.out.print("Enter the number: ");
         int num=scan.nextInt();
          findEvenOdd(num);
    }
}
```

Calling method without creating object

```
public class Addition
{
         public static int add(int n1, int n2)
         {
           int s;
            s=n1+n2;
            return s;
         }

     public static void main(String[] args)
     {
          int a = 19;
          int b = 5;
          int c = Addition.add(a, b);
          System.out.println("The sum of a and b is= " + c);
     }
}
```

Static Method

A method that has **static** keyword is known as static method. In other words, a method that belongs to a class is known as a static method. We can also create a static method by using the keyword static before the method name.

The main advantage of a static method is that we can call it without creating an object. It is invoked by using the **class name**.

```
import java.io. *;
```

Bishal Patel

```
public class StaticMethod
{
      static int a = 40;
      int b = 50;

    void simpleDisplay ()
    {
        System.out.println(a);
        System.out.println(b);
    }

    static void staticDisplay ()
    {
      System.out.println(a);
    }

    public static void main(String[] args)
    {
        StaticMethod obj=new StaticMethod ();
        obj.simpleDisplay();

         staticDisplay();
    }
}
```

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.

Eg Addition.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

Dhaval Patel

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.

```
class Adder
{
     static int add(int a,int b)
    {
       return a+b;
    }
     static int add(int a,int b,int c)
     {
       return a+b+c;
    }
}
class TestOverloading1
{
    public static void main(String[] args)
    {
      System.out.println(Adder.add(11,11));
      System.out.println(Adder.add(11,11,11));
    }

}
```

Overloading main method

```
public class OverLoadingConcept {


    public static void main(boolean args)
    {
           System.out.println("main() method overloaded" + " method 1 Executing");
           System.out.println(args);
```

Bishal Patel

```
    }
    public static void main(String a, String b)
    {
        System.out.println("main() overloaded" + " method 2 Executing");
        System.out.println(a + " " + b);
    }

    public static void main(int a)
    {
        System.out.println("main() overloaded" + " method 3 Executing");
        System.out.println(a);
    }

    public static void main(String[] args)
    {
        System.out.println("Original main()" + " Executing");
        System.out.println("Hello");


        OverLoadingConcept.main (true);

        OverLoadingConcept.main(bishal,patel);

                OverLoadingConcept.main(20);
    }
}
```

Constructors

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked.

Bishal Patel

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be **abstract**, **static**, **final**, and **synchronized**

Types of Java constructors

There are two types of constructors in Java:

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

Java Default Constructor

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

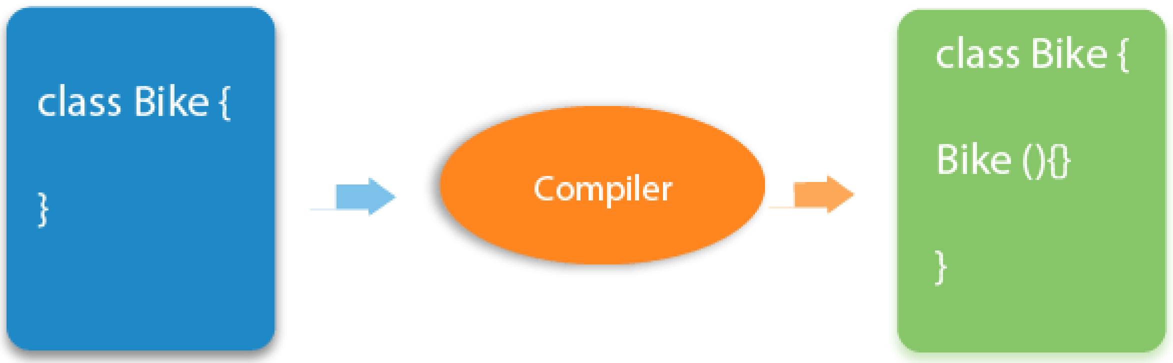
Syntax

```
class_name()
{
  // code
}
```

```
class Bike1
{
      Bike1()
      {
        System.out.println("Bike is created");
      }
    public static void main(String args[])
    {
       Bike1 b=new Bike1();
     }
}
```

Bishal Patel

***Rule**: If there is no constructor in a class, JVM automatically creates a default constructor*.



```
class Student3
{
    int id;
    String name;
    void display()
    {
      System.out.println(id+" "+name);
    }

    public static void main(String args[])
    {
       Student3 s1=new Student3();
       Student3 s2=new Student3();
       s1.display();
       s2.display();
    }
}
```

Bishal Patel

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

program

```
class Student4
{
   int id;
   String name;
   Student4(int i, String n)
   {
       id = i;
       name = n;
   }

   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();
  }
}
```

Bishal Patel

Constructor Overloading

In Java, we can overload constructors like methods. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

```
public class Student
{
    int id;
    String name;
    Student(){
    System.out.println("this a default constructor");
}

     Student(int i, String n)
    {
        id = i;
        name = n;
    }

    public static void main(String[] args)
    {
      //object creation
      Student s = new Student();
      System.out.println("\nDefault Constructor values: \n");
      System.out.println("Student Id : "+s.id + "\nStudent Name : "+s.name);

      System.out.println("\nParameterized Constructor values: \n");
      Student student = new Student(10, "David");

System.out.println("Student Id : "+student.id + "\nStudent Name : "+student.name);
   }
}
```

this Keyword

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference** variable that refers to the current object.

There are 6 usage of this keyword in java

- this can be used to refer current class instance variable.
- this can be used to invoke current class method.
- this() can be used to invoke current class constructor.
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.

Bishal Patel

- this can be used to return the current class instance from the method.

As a beginner we will learn only 3 usages of this keyword

1. this can be used to refer current class instance variable.

The **this** keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

```
class Student{
      int rollno;
      String name;
      float fee;
      Student(int rollno,String name,float fee){
          rollno=rollno;
          name=name;
          fee=fee;
       }
      void display(){
         System.out.println(rollno+" "+name+" "+fee);
      }
}
class TestThis1{
     public static void main(String args[]){
         Student s1=new Student(111,"ankit",5000f);
         Student s2=new Student(112,"sumit",6000f);
         s1.display();
         s2.display();
      }
}
```

Bishal Patel

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

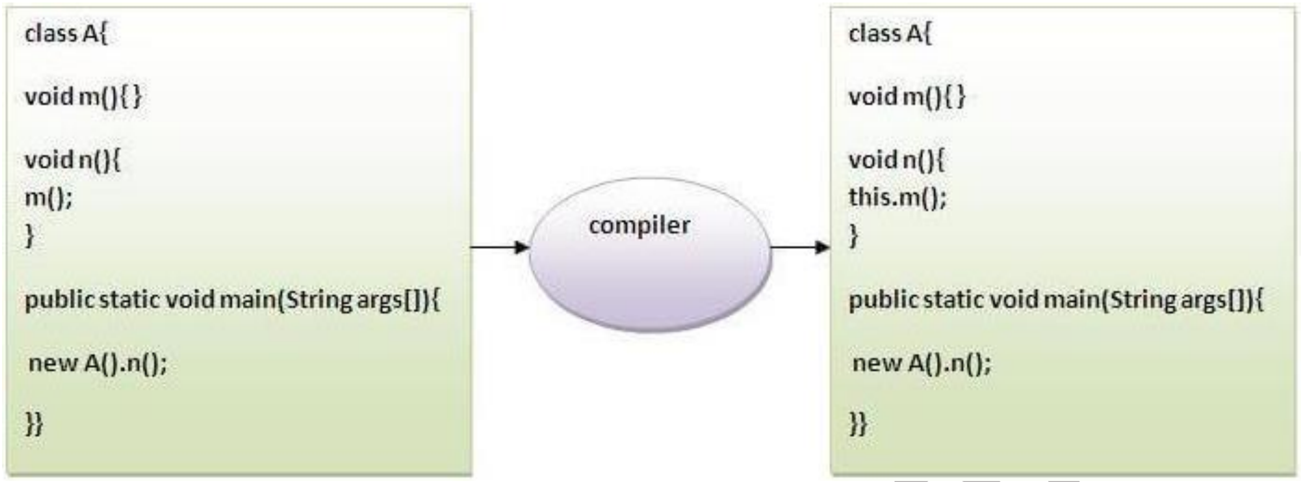
```
class Student{
    int rollno;
    String name;
    float fee;
    Student(int rollno,String name,float fee){
        this.rollno=rollno;
        this.name=name;
        this.fee=fee;
      }
      void display(){
         System.out.println(rollno+" "+name+" "+fee);
      }
}

class TestThis2{
    public static void main(String args[]){
     Student s1=new Student(111,"ankit",5000f);
     Student s2=new Student(112,"sumit",6000f);
       s1.display();
       s2.display();
     }
 }
```

this can be used to invoke current class method

You may invoke the method of the current class by using the this keyword. If you don't use **this** keyword, compiler automatically adds this keyword while invoking the method.

Bishal Patel



```
class A{
   void m1(){
     System.out.println("hello m1");
   }
   void n1(){
      System.out.println("hello n");
      this.m1();       //m();//same as this.m()
    }
}
class TestThis4{
     public static void main(String args[]){
         A a=new A();
          a.n1();
     }
}
```

Bishal Patel

this() can be used to invoke current class constructor

The **this()** constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

Example

```
class A
{
   A()
   {
     System.out.println("hello a");
   }
   A(int x)
   {
     this();
     System.out.println(x);
   }
}
class TestThis5
{
    public static void main(String args[])
    {
     A a=new A(10);
    }
}
```

Source File Declaration Rules

As the last part of this section, let's now look into the source file declaration rules. These rules are essential when declaring classes, *import* statements and *package* statements in a source file.

- There can be only one public class per source file.
- A source file can have multiple non-public classes.
- The public class name should be the name of the source file as well which should be appended by **.java** at the end. For example: the class name is *public class Employee{}* then the source file should be as Employee.java.
- If the class is defined inside a package, then the package statement should be the first statement in the source file.
- If import statements are present, then they must be written between the package statement and the class declaration. If there are no package statements, then the import statement should be the first line in the source file.

Bishal Patel

Classes have several access levels and there are different types of classes; abstract classes, final classes, etc. We will be explaining about all these in the access modifiers chapter.

Apart from the above mentioned types of classes, Java also has some special classes called Inner classes and Anonymous classes.

Bishal Patel