Chapter:9

Constructor and keywords

Constructors in Java

- 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.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

1. Constructor name must be the same as its class name

```
class Test
{
        Test()
        {

        }
}
```

Constructor

2. A Constructor must have no explicit return type

```
class Test
{
        void Test()
        {
                System.out.println("Constructor");
        }
        public static void main(String [] args)
        {
                Test T1=new Test();
                Test T2=new Test();
        }
}
```

No output

- Return type concepts are not applicable for constructors by mistake if you are trying to declare return type for constructor compiler will treat constructor

```
class Test
{
        Test()
        {
                System.out.println("Constructor");
        }
        public static void main(String [] args)
        {
          // Default constructor
                Test T1=new Test();
                Test T2=new Test();
        }
}
```

Output

```
Constructor
Constructor
```

3. In Java constructor modifiers like abstract, static, final, and synchronized are not allowed only allowed modifiers are public, private, protected and default

Types of Java constructors

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.
- Syntax of default constructor:

```
class_name()
{
}
```

- **Example of default constructor**
- In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.
- Java Program to create and call a default constructor

```
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();
    }
}
```

Constructor

Constructor Calling

Output:

Bike is created

- Rule: If there is no constructor in a class, compiler automatically creates a default constructor.
- Java default constructor

What is the purpose of a default constructor?

- The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type
- Example of default constructor that displays the default values

//Let us see another example of default constructor

```
//which displays the default values
class Student3
{
    int id;
    String name;
}
class Main
{
    public static void main(String args[])
    {
            //creating objects
            Student3 s1=new Student3();
            Student3 s2=new Student3();
            System.out.println(s1.id+" "+s1.name);
            System.out.println(s2.id+" "+s2.name);
    }
}
```

Output:

0 null

0 null

- **Explanation**: In the above class, you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

Java Parameterized Constructor

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

Why use the parameterized constructor?

- The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.
- **Example of parameterized constructor**
- In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
//Java Program to demonstrate the use of 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[])
        {
             //creating objects and passing values
             Student4 s1 = new Student4(111,"Karan");
            Student4 s2 = new Student4(222,"Aryan");
            //calling method to display the values of object
             s1.display();
             s2.display();
         }
}
```

Output:

111 Karan

222 Aryan

Constructor Overloading in Java

- In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.
- 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 of Constructor Overloading**

```
//Java program to overload constructors
class Student5
{
    int id;
    String name;
    int age;
    //creating two arg constructor
    Student5(int i,String n)
    {
        id = i;
        name = n;
    }
    //creating three arg constructor
```

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

Output:

111 Karan 0

22 Aryan 25

Difference between constructor and method in Java

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Java Copy Constructor

- There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.

Copying values with constructors Example

- In this example, we are going to copy the values of one object into another using Java constructor.

```
//Java program to initialize the values from one object to another object.
class Student6
```

```
{
    int id;
    String name;
    //constructor to initialize integer and string
    Student6(int i,String n)
    {
        id = i;
        name = n;
    }
   //constructor to initialize another object
    Student6(Student6 s)
    {
        id = s.id;
        name =s.name;
    }
    void display()
    {
        System.out.println(id+" "+name);
    }
    public static void main(String args[])
    {
        Student6 s1 = new Student6(111,"Karan");
        Student6 s2 = new Student6(s1);
        s1.display();
        s2.display();
    }
}
```

Output:

111 Karan

111 Karan

Copying values without constructors Example

- We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

```
class Student7
{
    int id;
    String name;
    Student7(int i,String n)
    {
        id = i;
        name = n;
    }
```

```
    Student7()
    {
    }
    void display()
    {
        System.out.println(id+" "+name);
    }
     public static void main(String args[])
    {
        Student7 s1 = new Student7(111,"Karan");
        Student7 s2 = new Student7();
        s2.id=s1.id;
        s2.name=s1.name;
        s1.display();
        s2.display();
    }
}
```

Output:

111 Karan

111 Karan

Q) Does constructor return any value?

- Yes, it is the current class instance (You cannot use return type yet it returns a value).

Q) Can constructor perform other tasks instead of initialization?

- Yes, like object creation, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.

Q) Is there Constructor class in Java?

- Yes.

Q) What is the purpose of Constructor class?

- Java provides a Constructor class which can be used to get the internal information of a constructor in the class. It is found in the java.lang.reflect package.

Simple program with default and parameterized constructors

```
class Student
{
	String name,type;
	int no;
	Student()//Default Constructor
	{
		name="Jango";
		type="OTHER";
		no=1;
	}
	Student(String a,String b,int c)//Parameterized constructor
	{
		name=a;
		type=b;
		no=c;
	}
	void jobTime()//Condition
	{
		if(type.equals("OTHER"))
		{
			System.out.println("Hello "+name);
			System.out.println("You are from "+type);
			System.out.println("you will get job in 4 years");
			System.out.println();
		}
		if(type.equals("LJU"))
		{
			System.out.println("Hello "+name);
			System.out.println("You are from "+type);
			System.out.println("you will get job in 3 years");
			System.out.println();
		}
		if(type.equals("STAR"))
		{
			System.out.println("Hello "+name);
			System.out.println("You are from "+type);
			System.out.println("you will get job in 2 years");
		}
	}
}
```

```
class ConsEx4
{
	public static void main(String args[])
	{
		Student s1=new Student();//Default constructor called
		s1.jobTime();
		Student s2=new Student("Pintu","LJU",2);//Parameterized constructor called
		s2.jobTime();
		Student s3=new Student("Chaman","STAR",3);//Parameterized constructor called
		s3.jobTime();
	}
}
/*E:\java ARP>javac ConsEx1.java
E:\java ARP>java ConsEx4
Hello Jango
You are from OTHER
you will get job in 4 years

Hello Pintu
You are from LJU
you will get job in 3 years

Hello Chaman
You are from STAR
you will get job in 2 years*/
```

this Keyword in Java

- There can be a lot of usage of **Java this keyword**. In Java, this i.s a **reference variable** that refers to the current class object.
- **Lets take an example to prove that how this refers to current class object**

```
class objref
{
	void show()
	{
		System.out.println(this);
	}
	public static void main(String [] args)
	{
		objref OR=new objref();
		System.out.println(OR);
		OR.show();
	}
```

```
}
```

E:\java ARP>javac thisrefer.java
E:\java ARP>java objref

objref@2c7b84de
objref@2c7b84de

Same Reference address

Uses of this keyword

1) this can be used to refer current class instance variable.
2) this can be used to invoke current class method (implicitly)
3) this() can be used to invoke current class constructor.
4) this can be passed as an argument in the method call.
5) this can be passed as argument in the constructor call.
6) this can be used to return the current class instance from the method.

(1) 'this' to call current class object or variable or instance

```
class Fac
{
        int id;
        int no;

        /*void setFac(int id,int no)
        {
                id=id; here id and no are local variables
                no=no;
        }
        ->As local variables and class variables has same name and type ,
        here local variables hides class variables
        -> So now we will write          this.id=id; in place of   id=id;
                                 this.no=no;             no=no;
        */

        void setFac(int id,int no)
        {
                this.id=id;
                this.no=no;
        }
```

```
	void getFac()
	{
		System.out.println("id= "+id);
		System.out.println("no= "+no);
	}
}
class ConsEx1
{
	public static void main(String args[])
	{
		Fac f1=new Fac();
		f1.setFac(30,40);
		f1.getFac();
	}
}
/*E:\java ARP>javac ConsEx3.java
E:\java ARP>java ConsEx1
Output:
id= 30
no= 40*/
```

(2)'this' to call current class Method

```
class Fac1
{
	int id;
	int no;
	void setFac(int id,int no)
	{
		this.id=id;
		this.no=no;
	}
	void getFac()
	{
		System.out.println(id);
		System.out.println(no);
	}
	void display()
	{
		System.out.println("hi");
		System.out.println("Output of this.getFac()");
		this.getFac();   //called current class method
		System.out.println("Output of getFac()");
		getFac();  //This lakho k na lakho kai fark na pade
```

```
	}
}
class ConsEx2
{
	public static void main(String args[])
	{
		Fac1 f1=new Fac1();
		f1.setFac(30,40);
		f1.display();//Not required to call getFac() method bcz called in display method
	}
}
/*E:\java ARP>javac ConsEx3.java
E:\java ARP>java ConsEx2
hi
Output of this.getFac()
30
40
Output of getFac()
30
40*/
```

(3a)'this' to call current class Constructor

```
/*Default constructor called in parameterized constructor*/

class Fac3
{
	int id;
	int no;
	Fac3(int id,int no)
	{
		this(); //called current class default constructor Fac()
		this.id=id;
		this.no=no;
	}
	Fac3()
	{
		System.out.println("hi default");
	}
	void getFac()
	{
		System.out.println(id);
		System.out.println(no);
	}
}
```

```
class ConsEx3
{
	public static void main(String args[])
	{
		Fac3 f1=new Fac3(40,50); //object of parameterized constructor
		f1.getFac();
	}
}
/*E:\java ARP>javac ConsEx3.java
E:\java ARP>java ConsEx3
Output:
hi default
40
50*/
```

(3b)'this' to call current class Constructor

/* Parameterized constructor called in Default constructor*/

```
class Fac4
{
	int id;
	int no;
	Fac4(int id,int no)
	{
		this.id=id;
		this.no=no;
		System.out.println("hi Parameterized");
	}
	Fac4()
	{
		this(10,20);  //called current class Parameterized constructor.
		System.out.println("hi default");
	}
	void getFac()
	{
		System.out.println(id);
		System.out.println(no);
	}
}
class ConsEx4
{
	public static void main(String args[])
	{
```

```
		Fac4 f1=new Fac4(); //object of Default constructor
		f1.getFac();
	}
}
/*E:\java ARP>javac ConsEx3.java
E:\java ARP>java ConsEx4
hi Parameterized
hi default
10
20
*/
```

(4)'this'-as an argument in method call

```
class Fac5
{
	int id;
	int no;
	void setFac(int id,int no)
	{
		this.id=id;
		this.no=no;
	}
	void print(Fac5 f)   //  Method that receives 'this' keyword as argument(parameter)
	{
		System.out.println(f.id);
		System.out.println(f.no);
	}
	void obj_print()
	{
		print(this);
	}
}
class ConsEx5
{
	public static void main(String args[])
	{
		Fac5 f1=new Fac5();
		f1.setFac(10,20);
		f1.obj_print();
	}
}
/*E:\java ARP>javac ConsEx3.java
```

E:\java ARP>java ConsEx5
Output:
10
20*/

(5)'this'-as an argument in constructor call

```
class Fac6
{
	int id;
	int no;
	Fac6 f2;
	Fac6()//default constructor
	{
		id=10;
		no=20;
		f2=new Fac6(this);
	}
	Fac6(Fac6 f)   //Constructor that receives 'this' as an argument
	{
		System.out.println("object as argument");
		System.out.println(f.id);
		System.out.println(f.no);
	}
	void getF2()
	{
		System.out.println("hi f2");
		System.out.println(f2.id);
		System.out.println(f2.no);
	}
}
class ConsEx6
{
	public static void main(String args[])
	{
		Fac6 f1=new Fac6();//Default constructor called
		f1.getF2();
	}
}
```

/*E:\java ARP>javac ConsEx3.java
E:\java ARP>java ConsEx6
Output:

object as argument
10
20
hi f2
0
0
***/**

(7) "This" as Return from method

```
class Fac7
{
        int id;
        int no;

        Fac7(int id,int no)
        {
                this.id=id;
                this.no=no;
        }
        Fac7 add(Fac7 p,Fac7 q)
        {
                id=p.id+q.id;
                no=p.no+q.no;
                return this;
        }
        Fac7()
        {
                // do nothing constructor
        }
        void getFac()
        {
                System.out.println(id);
                System.out.println(no);
        }
}
class ConsEx7
{
        public static void main(String args[])
        {
                Fac7 f1=new Fac7(10,20);
                Fac7 f2=new Fac7(30,40);
                Fac7 f3=new Fac7();
```

```
        Fac7 f4=new Fac7();
        f4=f3.add(f1,f2);
        f4.getFac();
    }
}
```

/*E:\java ARP>javac ConsEx3.java
E:\java ARP>java ConsEx7
Output:
40
60*/

Static keyword in java

- The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.
- The static can be:

1) Variable (also known as a class variable)
2) Method (also known as a class method)
3) Block
4) Nested class

1) Java static variable

- If you declare any variable as static, it is known as a static variable.
- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

- It makes your program memory efficient (i.e., it saves memory).
- Understanding the problem without static variable

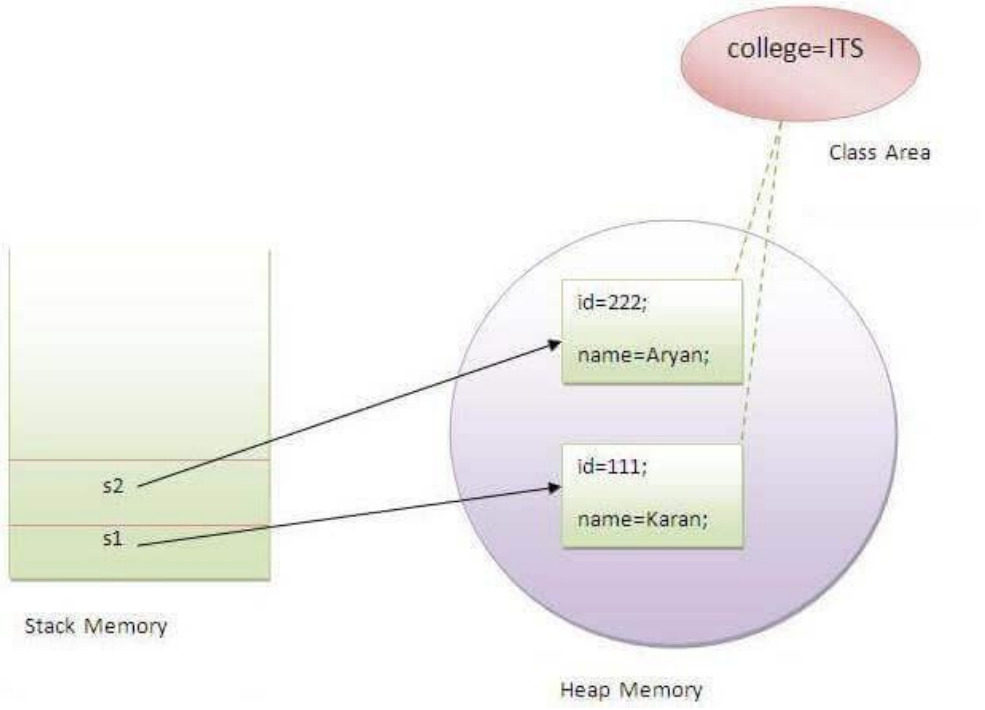
```
class Student
{
    int rollno;
    String name;
    String college="ITS";
}
```

- Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

Example of static variable

//Java Program to demonstrate the use of static variable

```
class Student
{
	int rollno;//instance variable
	String name;
	static String college ="LJIET";//static variable
	//constructor
	Student(int r, String n)// Constructor
	{
		rollno = r;
		name = n;
	}
	//method to display the values
	void display ()
	{
		System.out.println(rollno+" "+name+" "+college);
	}
}
//Main class to show the values of objects
class Main
{
	public static void main(String args[])
	{
		Student s1 = new Student(111,"Karan");
		Student s2 = new Student(222,"Aryan");
		//we can change the college of all objects by the single line of code
		//Student.college="NIRMA";
 /* can make changes in static variables using class name with dot operator*/
		s1.display();
		s2.display();
	}
}
/*
E:\java ARP>java Main
111 Karan LJIET
222 Aryan LJIET  */
```



Program of the counter without static variable

- In this example, we have created an instance variable named count which is incremented in the constructor.
- Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable.
- If it is incremented, it won't reflect other objects. So each object will have the value 1 in the count variable.

```
//Java Program to demonstrate the use of an instance variable
//which get memory each time when we create an object of the class.
class Counter
{
        int count=0;//will get memory each time when the instance is created
        Counter()
        {
                count++;//incrementing value
                System.out.println(count);
        }
        public static void main(String args[])
        {
                //Creating objects
                Counter c1=new Counter();
                Counter c2=new Counter();
```

```
        Counter c3=new Counter();
    }
}
/*E:\java ARP>java Counter
1
1
1  */
```

Program of counter by static variable

- As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
//Java Program to illustrate the use of static variable which
//is shared with all objects.
class Counter2
{
    static int count=0;//will get memory only once and retain its value

    Counter2()
    {
        count++;//incrementing the value of static variable
        System.out.println(count);
    }

    public static void main(String args[])
    {
        //creating objects
        Counter2 c1=new Counter2();
        Counter2 c2=new Counter2();
        Counter2 c3=new Counter2();
    }
}
/*
E:\java ARP>java Counter2
1
2
3   */
```

2) Java static method

- If you apply static keyword with any method, it is known as static method.
- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Example of static method

//Java Program to demonstrate the use of a static method.

```
class Student
{
	int rollno;
	String name;
	static String college = "LJIET";
	//static method to change the value of static variable
	static void change()
	{
		college = "DDIT";
	}
	//constructor to initialize the variable
	Student(int r, String n)
	{
		rollno = r;
		name = n;
	}
	 //method to display values
	void display()
	{
		System.out.println(rollno+" "+name+" "+college);
	}
}
//Test class to create and display the values of object
class Main
{
	public static void main(String args[])
	{
		Student.change();//calling change method
/* We can call the static method using class name with Dot operator*/
		//creating objects
		Student s1 = new Student(111,"Karan");
		Student s2 = new Student(222,"Aryan");
		Student s3 = new Student(333,"Sonoo");
		//calling display method
		s1.display();
		s2.display();
		s3.display();
	}
}
```

***/ E:\java ARP>java Main**
111 Karan DDIT
222 Aryan DDIT
333 Sonoo DDIT */

Restrictions for the static method

There are two main restrictions for the static method. They are:

- The static method cannot use non static data member or call non-static method directly.
- this and super cannot be used in static context.

3) Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.
-

Example of static block

```
class A2
{
	 Static
	{
		System.out.println("static block is invoked");
	}
	 public static void main(String args[])
	{
		System.out.println("Hello main");
	}
}
```

Run by: java A2

static block is invoked
Hello main

```
class Test
{
	int a;
	static int cnt=0;
	static
	{
		System.out.println("Hi static Block");
	}

	static void print()//static method
```

```
	{
		System.out.println("hi static");
		/*System.out.println(a);error: non-static variable 'a' cannot be
		referenced from a static context*/
	}
	void display()
	{
		System.out.println("hi non static");
		System.out.println(cnt);
	}
	Test()
	{
		System.out.println("Hi Test constructor");
	}
}
class Main

{
	public static void main(String args[])
	{
		Test.print();
		Test t1=new Test();
		t1.display();
	}
}
/*E:\java ARP>java Main
Hi static Block
hi static
Hi Test constructor
hi non static
0*/
```