Classes and Objects: Basic concepts of OOPS, Classes and Objects, Modifiers, Passing arguments, Constructors, Overloaded Constructors, Overloaded Operators, Static Class Members, Garbage Collection.

Inheritance: Basics of inheritance, Inheriting and Overriding Superclass methods, Calling Superclass Constructor, Polymorphism, Abstract Classes, Final Class.

OOPs (Object-Oriented Programming System)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc.
Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- ❖ Object
- ❖ Class
- ❖ Inheritance
- ❖ Polymorphism
- ❖ Abstraction
- ❖ Encapsulation

Objects and Classes in Java

An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.

What is an object in Java

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (**tangible** and **intangible**). The example of an intangible object is the banking system.

An object has three characteristics:

State: represents the data (**value**) of an object.

Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.

Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

- ❖ An object is *a real-world entity*.
- ❖ An object is *a runtime entity*.
- ❖ The object is *an entity which has state and behavior*.
- ❖ The object is *an instance of a class*.

What is a class in Java?

A Class is a user defined data type which contain data member , member functions with different access specifiers like private, protected, public & default. A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

Fields

Methods

Constructors

Blocks

Nested class and interface

Syntax to declare a class:

```
class <class_name>
{
   field;      // data members
   method;   // members fucntiosn
}
```

Instance variable(data member) in Java

A variable which is created inside the class but outside the main method is known as an instance variable.

Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created.

That is why it is known as an instance variable.

```
Class    Rectangle
{
          int length, width;  // instance variable

}
```

Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

Code Reusability
Code Optimization

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in **Heap** memory area.

=>Write any Example to create any class to print object values

3 Ways to initialize object

There are 3 ways to initialize object in Java.

- ❖ By reference variable (with new keyword)
  - ❖ z
- ❖ By constructor

Anonymous object

Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

If you have to use an object only once, an anonymous object is a good approach.

For example:

```
Calculation obj = new Calculation();

new Calculation();      //anonymous object

new Calculation().fact(5);
```

Access Modifiers in Java

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

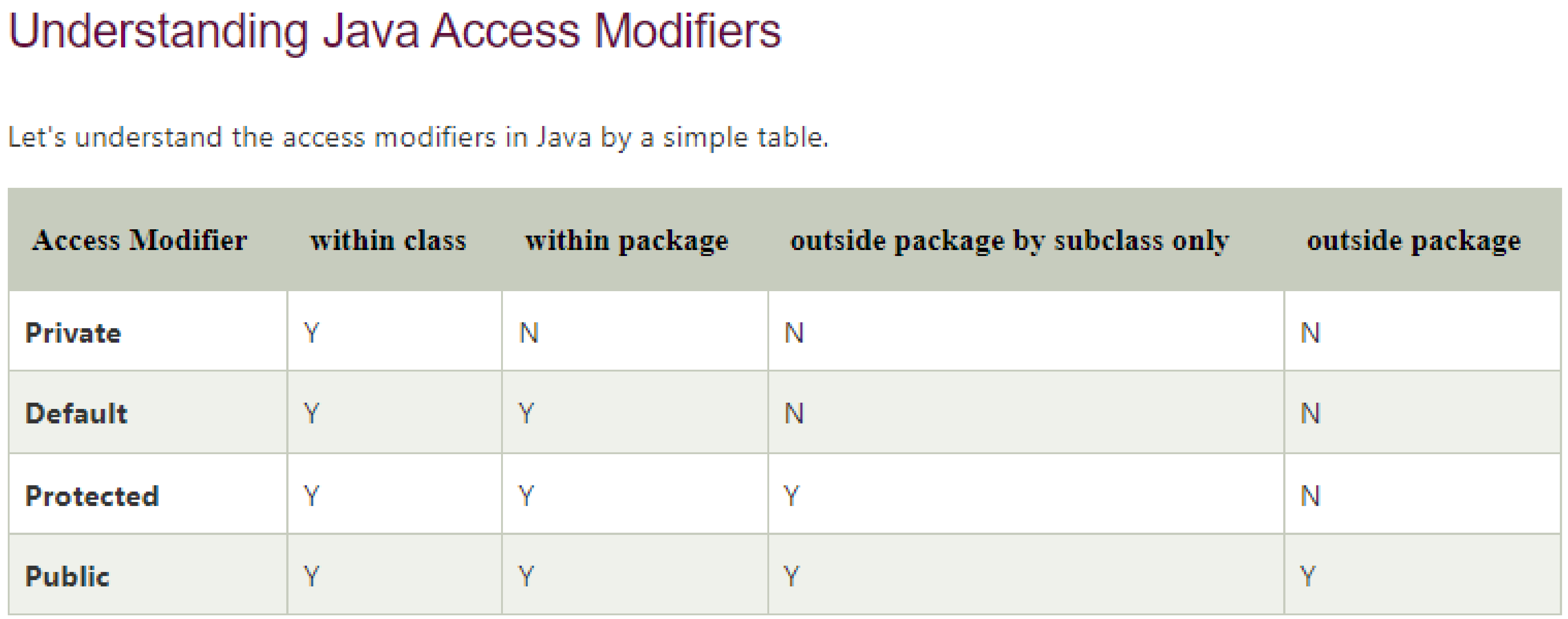
Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Access Modifiers in Java

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.



Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table.

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

Access Modifiers in Java

Private: The private access modifier is accessible only within the class.

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

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

Access Modifiers in Java

2) 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. But, it is more restrictive than protected, and public.

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

```
//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
           }
}
```

Access Modifiers in Java

3) 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, method and constructor. It can't be applied on the class.
It provides more accessibility than the default modifer.

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

```
//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

Access Modifiers in Java

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

```
//save by A.java

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

```
//save by B.java

package mypack;
import pack.*;

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

Passing Arguments

```
class V
{
        int a,b;  //States or Fields or Instance variables or data members
        void print() //Behaviour or member functions
        {
                System.out.println("The value of a is "+a+" b is "+b);
        }

}
class K
{
        double c,d;
        //By Intilizing through method
        public void set(double a,double b) {
                c=a;
                d=b;
        }
        void print() //Behaviour or member functions
        {
                System.out.println("The value of c is "+c+" d is "+d);
        }

}
```

```
public class ObjectVal {

        public static void main(String[] args) {


                V t2=new V();
                V t3=new V();
                t2.a=20;
                t2.b=100;
                t2.print();

                 V t1=new V();  // Creating an object
                  t1.a=100;
                  t1.b=200;  //By reference variable
                 t1.print();

                K a1=new K();
                a1.set(100.78,    200.89);
                a1.print();
        }

}
```

Constructors

Constructors are special member functions of the class. 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.

```
    Class   Rectangle
   {
        public Rectangle()  //  Constructors
        {


        }
  }
```

Rules:-

1. Constructor has the same name as class name.
2. A Constructor must have no explicit return type.
3. Constructor are called when we create the object of the same class.
4. A Java constructor cannot be abstract, static, final, and synchronized

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**.

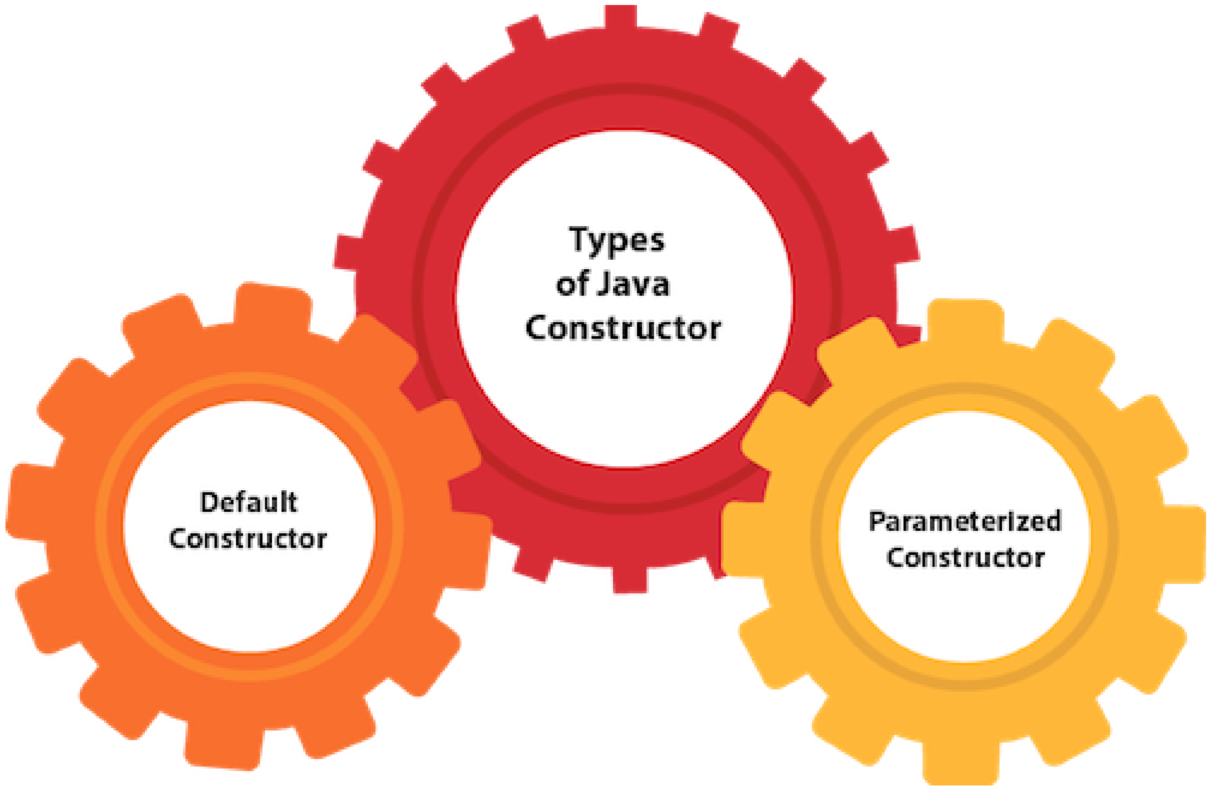
There are two types of constructors in Java: no-arg constructor, and **parameterized** constructor.

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.

Types of Java constructors



There are two types of constructors in Java:

1. Default constructor (no-arg constructor)

2. Parameterized 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 to demonstrate the default values of constructor

Java Default Constructor

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

Syntax of default constructor:

```
Class   class_name()
{

}
```

Example of 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 obj2 = new Bike1();
    }
}
```

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.

=> Program to demonstrate the use of the parameterized constructor.

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.

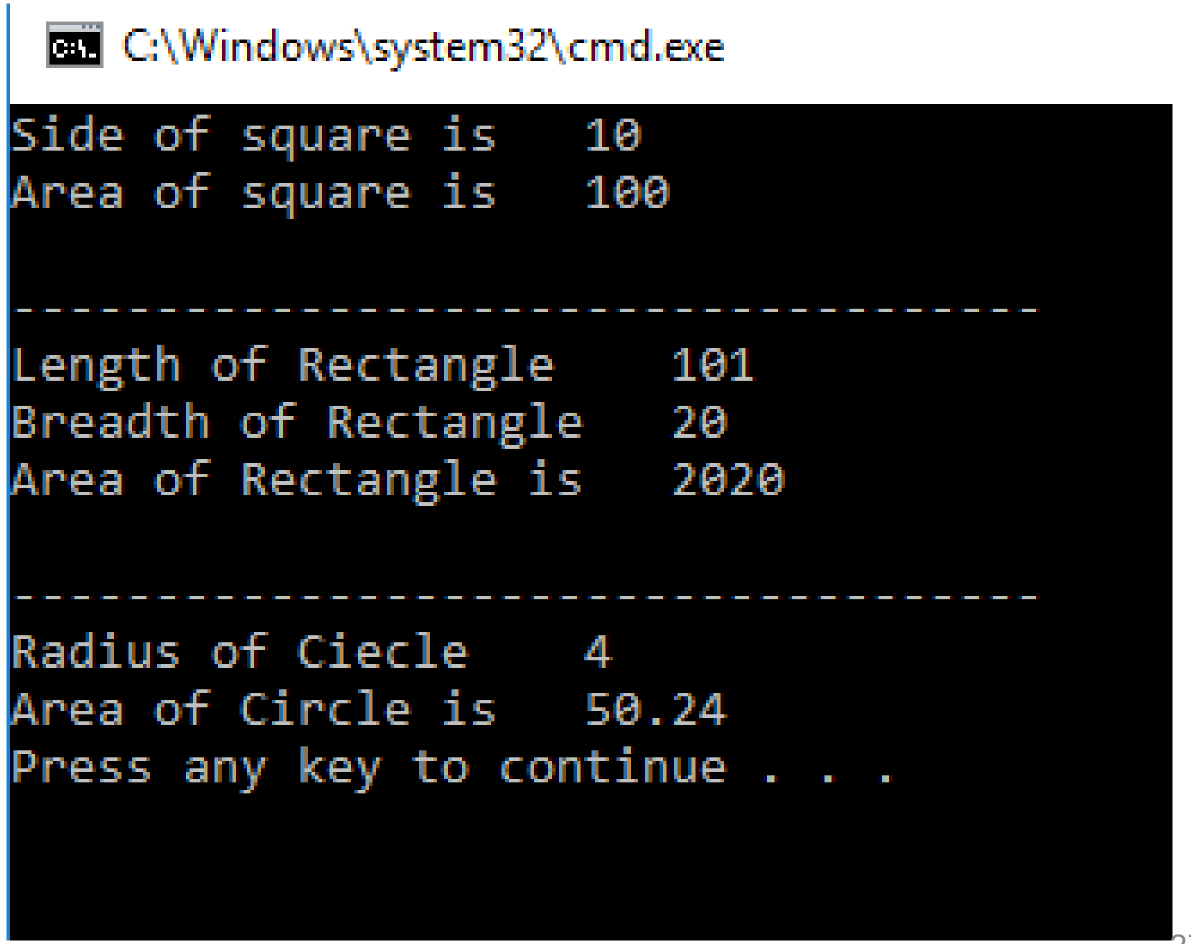
Constructor overloading in Java is a technique of having more than one constructor with **different parameter** lists.

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.

```
class    Area
{
    // Area of Square = side*side
                   public   Area(int a)
                   {
                        int area= a*a;
                        System.out.println("Side of square is    "+a);
                        System.out.println("Area of square is    "+area);
                   }
                   // Area of Rectangle
                   public  Area(int l,int b)
                   {
                        System.out.println("\n------------------------------------");
                       int area = l*b;
                        System.out.println("Length of Rectangle     "+l);
                        System.out.println("Breadth of Rectangle    "+b);
                           System.out.println("Area of Rectangle is    "+area);

                   }
                   // Area of Circle
                   public  Area(double pi, int r)
                   {
                                         System.out.println("\n------------------------------------");
                                                        double area = pi*r*r;
                                                         System.out.println("Radius of Ciecle     "+r);

                              System.out.println("Area of Circle is    "+area);
                   }
}
```

```
class   ConstrcutorOverloading
{
            public static void main(String args[])
            {
                  Area obj1 = new Area(10);
                  Area obj2 = new Area(101,20);
                  Area obj3 = new Area(3.14, 4);

            }
}
```



C:\Windows\system32\cmd.exe

```
Side of square is   10
Area of square is   100


------------------------------------

Length of Rectangle    101
Breadth of Rectangle   20
Area of Rectangle is   2020


------------------------------------

Radius of Ciecle    4
Area of Circle is   50.24
Press any key to continue . . .
```

```
//Java Program to demonstrate the use of the parameterized constructor.
class Student4
{
    int id;      // instance variable
    String name;    // instance variable
                                    //creating a parameterized constructor
    Student4(int id,String name)  // Here, I and n are local variables
    {
         this.id = id;
         this.name = name;
    }

    //method to display the values
    void display()     //  Member function
     {
          System.out.println(id+" "+name);
      }
```

//Java Program to demonstrate the use of the parameterized constructor.

```
    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();
   }
}
```

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

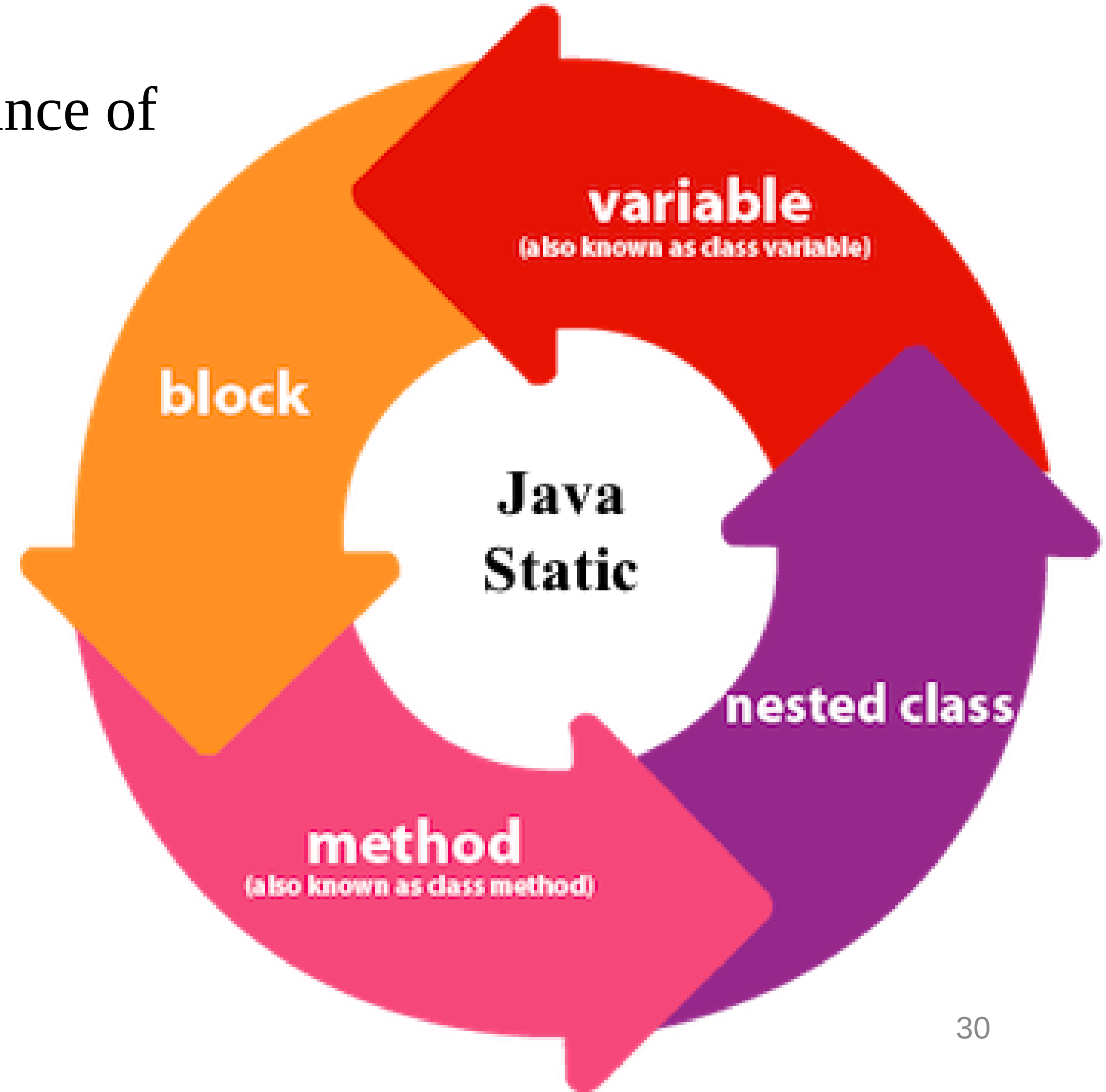
the class.
The static can be:

❖ Variable (also known as a class variable)

❖ Method (also known as a class method)

❖ Block

❖ 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";
}
```

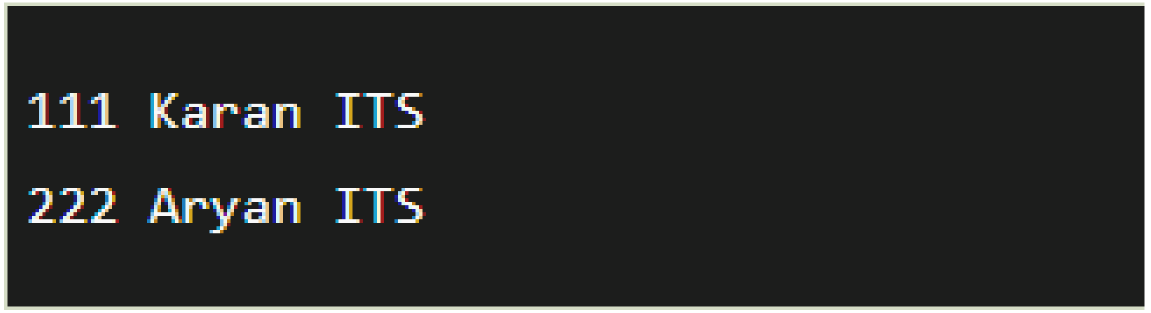
Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static,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

Java static property is shared to all objects.

```
//Java Program to demonstrate the use of static variable
class Student
{
   int rollno;//instance variable
   String name;
   static String college ="ITS";//static variable
   //constructor
   Student(int r, String n) // Local variable
   {
         rollno = r;
         name = n;
   }
   //method to display the values
   void display ()
   {
               System.out.println(rollno+" "+name+" "+college);
    }
}
```

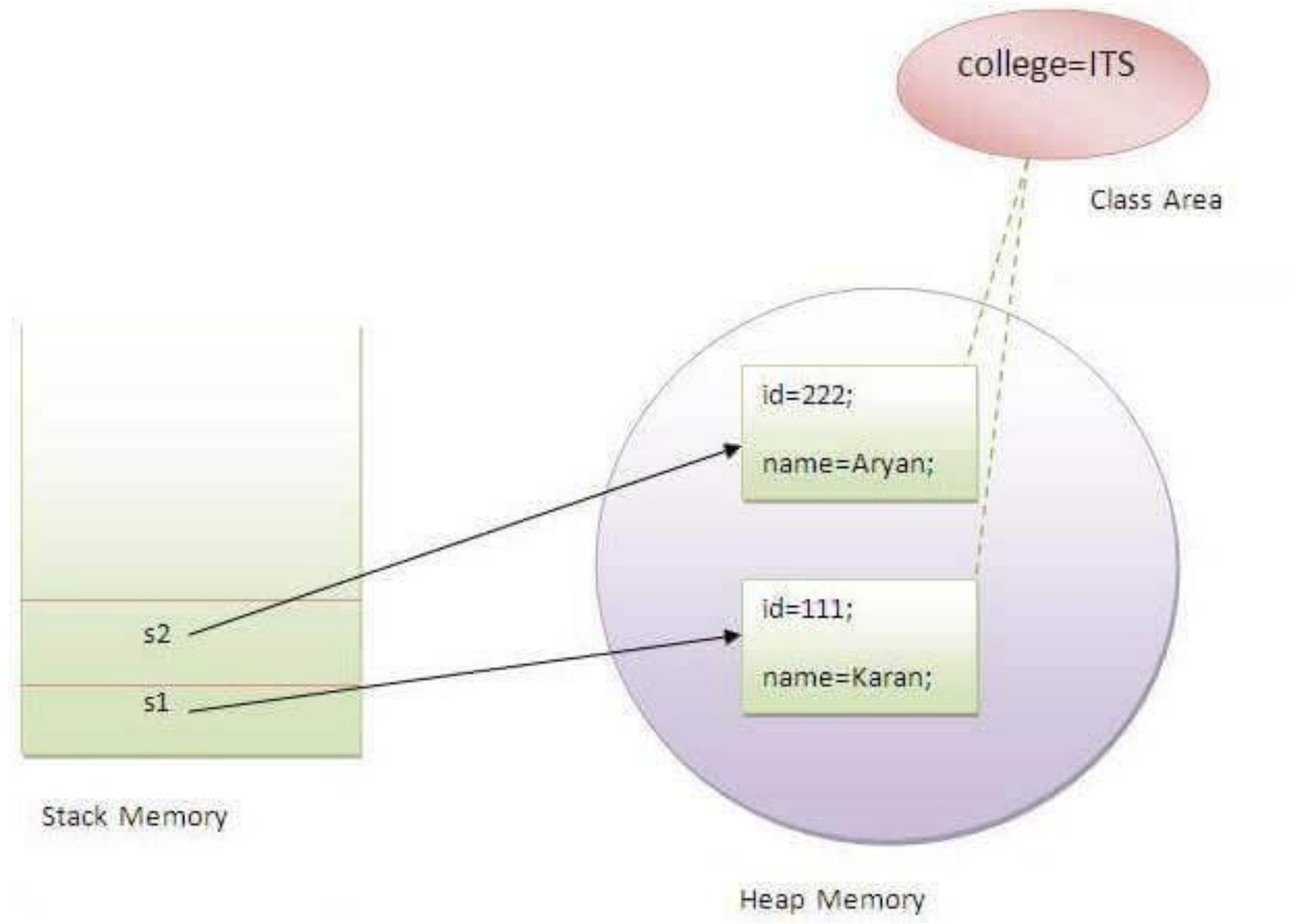
```
//Java Program to demonstrate the use of static variable
public class TestStaticVariable1
{
      public static void main(String args[])
      {
           Student s1 = new Student(111,"Karan");
           Student s2 = new Student(222,"Aryan");

           //we can change the college of all objects by the single line of code

          //  Student.college="BBDIT";

           s1.display();
           s2.display();
      }
}
```

Output:



```
111 Karan ITS
222 Aryan ITS
```

Static class members, Garbaze collection



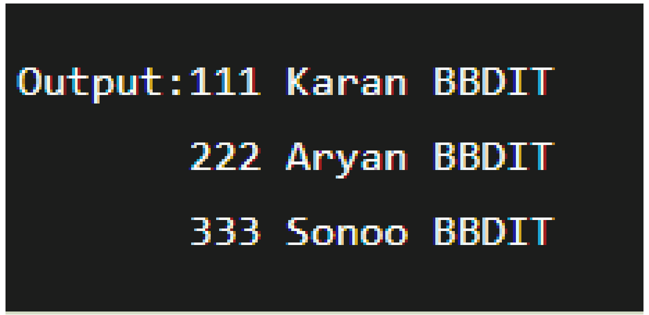
2) Java static method

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

```
//Java Program to demonstrate the use of a static method.
class Student{
     int rollno;
     String name;
     static String college = "ITS";
     //static method to change the value of static variable
     public static void change()
     {
            college = "BBDIT";
     }
     //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
public class TestStaticMethod
{
    public static void main(String args[])
    {
           Student.change();//calling change method

           //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();
    }
}
```



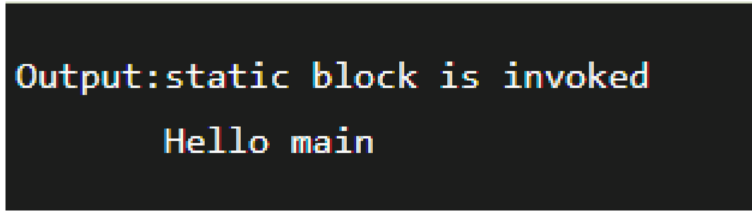
```
Output:111 Karan BBDIT
       222 Aryan BBDIT
       333 Sonoo BBDIT
```

3) Java static block

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

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");
   }
}
```



```
Output:static block is invoked
       Hello main
```