OBJECT ORIENTED PROGRAMMING USING JAVA

OUTLINE

- Static Initialization Block (SIB)
- Instance initialization block (IIB)

STATIC INITIALIZATION BLOCK (SIB)

- A Block named as Static inside a class is called Static Initialization Block(SIB).
- Hence, at the time of class loading if we want to perform any activity we have to define that inside a static block.
- SIB's are invoked only once at the time of the corresponding loading class or referred in to the memory.
- Is used to initialize the static data member.
- Static block can only access static variables and static methods of its class.

Syntax:

```
static {
   // Logic here or Java code.
 }
```

STATIC INITIALIZATION BLOCK (SIB): EXAMPLE

```
class Main{
  static
  {
      System.out.println("static block is invoked");

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

STATIC INITIALIZATION BLOCK (SIB): EXAMPLE

```
class Main{
  static
  {
      System.out.println("static block is invoked");

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

```
static block is invoked
Hello main
```

WHY STATIC BLOCK IS EXECUTED BEFORE MAIN METHOD?

- When we execute a particular class, JVM performs two actions at the runtime. They are as:

1. JVM loads corresponding dot class file (byte code) into memory.
2. During the dot class file loading into memory, static block is executed. After loading the dot class file, JVM calls the main method to start execution. Therefore, static block is executed before the main method.

HOW MANY TIMES DOT CLASS FILE IS LOADED INTO MEMORY?

- Dot class file is loaded into the memory only one time. So, only one time static block will be executed.

ORDER OF EXECUTION OF MULTIPLE STATIC BLOCKS IN JAVA?

- A class can have any number of static initialization blocks that will execute in the same sequence as written in the program.
- That is, the order of execution of multiple static initialization blocks is executed automatically from top to bottom during the dot class file loading.

USE OF STATIC BLOCK IN JAVA

- There are mainly three uses of static block in java that are as follows:

1. The purpose of using a static initialization block is to write that logic inside static block that is executed during the class loading.

2. It is mostly used for changing default value of static variables.

3. It is used to initialize static variables of the class.

PREDICT THE OUTPUT

```
public class Main
{
   static String eName = "Akash";
   static int eID;
   static int age;
   String companyName = "Bennett"; // Instance variable.

    static { // Change the value of static variable in the static initialization block.
       eName = "Roy";
       System.out.println("Name of Employee: " +eName);
    }
    static { // Initialize the value of static variable in the S.B.
       eID = 1234;
       System.out.println("Employee's Id: " +eID);
     }
static { // If you don't assign the value of static variable then it will print default value.
//Here I am assigning any value to the employee's age. So it will print default value 'zero' on the console.
//Zero is the default value of an integer.
     System.out.println("Employee's age: " +age);
  }
static {
     System.out.println("Company name: " +companyName);
  }
public static void main(String[] args)
{}}
```

PREDICT THE OUTPUT

```java
public class Main
{
   static String eName = "Akash";
   static int eID;
   static int age;
   String companyName = "Bennett"; // Instance variable.

    static { // Change the value of static variable in the static initialization block.
       eName = "Roy";
       System.out.println("Name of Employee: " +eName);
    }
    static { // Initialize the value of static variable in the S.B.
       eID = 1234;
       System.out.println("Employee's Id: " +eID);
     }
static { // If you don't assign the value of static variable then it will print default value.
//Here I am assigning any value to the employee's age. So it will print default value 'zero' on the console.
//Zero is the default value of an integer.
     System.out.println("Employee's age: " +age);
  }
static {
     System.out.println("Company name: " +companyName);
  }
public static void main(String[] args)
{}}
```

```
Main.java:29: error: non-static variable companyName cannot be referenced from a static context
     System.out.println("Company name: " +companyName);
                                         ^
```

PREDICT THE OUTPUT

```
public class Main
{
   static String eName = "Akash";
   static int eID;
   static int age;
   String companyName = "Bennett"; // Instance variable.

    static { // Change the value of static variable in the static initialization block.
       eName = "Roy";
       System.out.println("Name of Employee: " +eName);
    }
    static { // Initialize the value of static variable in the S.B.
       eID = 1234;
       System.out.println("Employee's Id: " +eID);
     }
static { // If you don't assign the value of static variable then it will print default value.
//Here I am assigning any value to the employee's age. So it will print default value 'zero' on the console.
//Zero is the default value of an integer.
     System.out.println("Employee's age: " +age);
  }
public static void main(String[] args)
{}}
```

PREDICT THE OUTPUT

```
public class Main
{
   static String eName = "Akash";
   static int eID;
   static int age;
   String companyName = "Bennett"; // Instance variable.

    static { // Change the value of static variable in the static initialization block.
       eName = "Roy";
       System.out.println("Name of Employee: " +eName);
    }
    static { // Initialize the value of static variable in the S.B.
       eID = 1234;
       System.out.println("Employee's Id: " +eID);
     }
static { // If you don't assign the value of static variable then it will print default value.
//Here I am assigning any value to the employee's age. So it will print default value 'zero' on the console.
//Zero is the default value of an integer.
     System.out.println("Employee's age: " +age);
  }
public static void main(String[] args)
{}}
```

```
Name of Employee: Roy
Employee's Id: 1234
Employee's age: 0
```

INSTANCE INITIALIZATION BLOCK

- Instance Initializer block is used to **initialize the instance data member**. It **run each time when object of the class is created.**
- The initialization of the instance variable can be done directly but there can be performed extra operations while initializing the instance variable in the instance initializer block.
- **No keyword is required** to define an instance initialization block.
- An instance initialization block is only executed when there is a call to the constructor for creating an object.
- An instance initialization block can not only access static variables and static methods but also instance variables and instance methods of the class
- Instance initialization block can run many times, whenever there is a call to the constructor of the class

INSTANCE INITIALIZATION BLOCK: SYNTAX

```
{
// This is called as instance block
}
```

INSTANCE INITIALIZATION BLOCK: EXAMPLE

```
class Main
{
    Main( )
    {
        System.out.println("0 argument constructor");
     }

    {
        System.out.println("Instance block");
        }
    public static void main(String[ ] args)
    {
        Main obj= new Main( );
         }}
```

INSTANCE INITIALIZATION BLOCK: EXAMPLE

```
class Main
{
    Main( )
    {
        System.out.println("0 argument constructor");
     }

    {
        System.out.println("Instance block");
        }
    public static void main(String[ ] args)
    {
        Main obj= new Main( );
         }}
```

```
Instance block
0 argument constructor
```

INSTANCE INITIALIZATION BLOCK: EXAMPLE

```
class Main
{
   Main( )                                                    //
   {
      System.out.println("0 argument constructor");    }
   Main(int a)                                                //
   {
      System.out.println("1 argument constructor");    }
   Main(int a , int b)                                        //
   {
      System.out.println("2 arguments constructor");     }
   {                                                          //
      System.out.println("Instance block");          }
   public static void main(String[ ] args)
   {
      new Main( );
      new Main(10);
      new Main(10,20); }}
```

INSTANCE INITIALIZATION BLOCK: EXAMPLE

```
class Main
{
   Main( )                                                    //
   {
      System.out.println("0 argument constructor");    }
   Main(int a)                                                //
   {
      System.out.println("1 argument constructor");    }
   Main(int a , int b)                                        //
   {
      System.out.println("2 arguments constructor");     }
   {                                                          //
      System.out.println("Instance block");          }
   public static void main(String[ ] args)
   {
      new Main( );
      new Main(10);
      new Main(10,20); }}
```

```
Instance block
0 argument constructor
Instance block
1 argument constructor
Instance block
2 arguments constructor
```

INSTANCE INITIALIZATION BLOCK: EXAMPLE

```
public class Main {
    static int staticVariable;
    int nonStaticVariable;
static {
        System.out.println("Static initalization.");
        staticVariable = 5;
    }
    {
        System.out.println("Instance initialization.");
        nonStaticVariable = 7;
    }
    public Main() {
        System.out.println("Constructor.");
    }
    public static void main(String[] args) {
        new Main();
        new Main();   }}
```

INSTANCE INITIALIZATION BLOCK: EXAMPLE

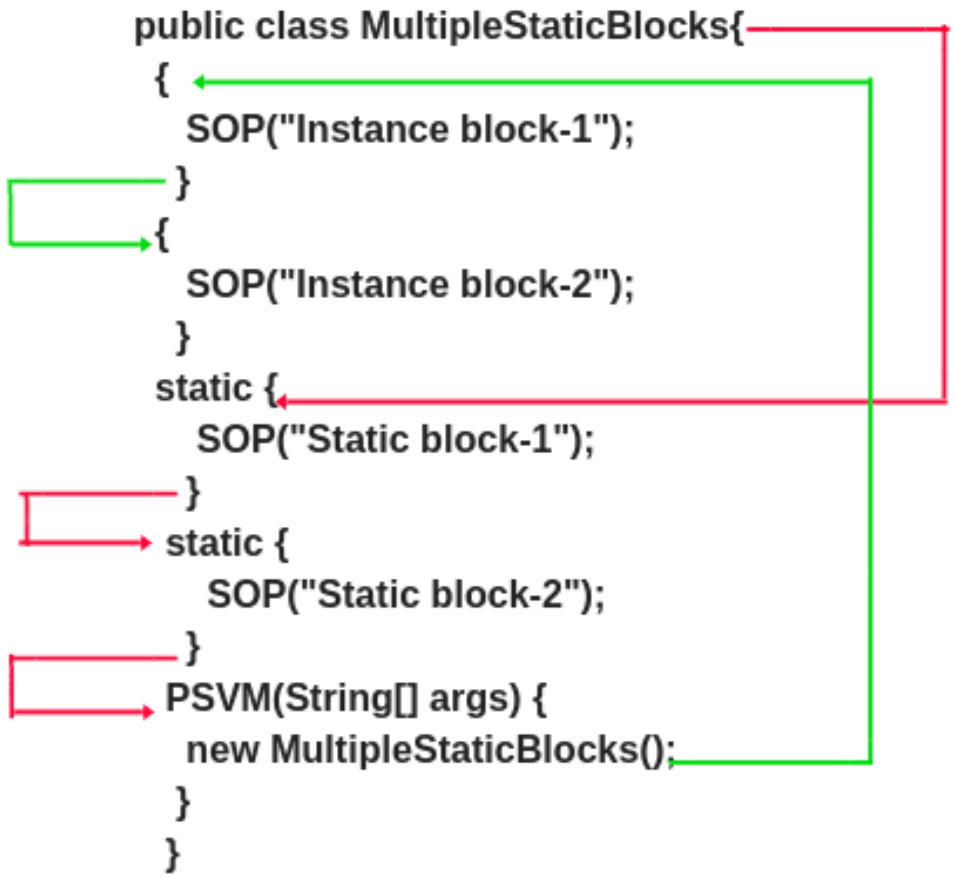
```
public class Main {
    static int staticVariable;
    int nonStaticVariable;
static {
        System.out.println("Static initalization.");
        staticVariable = 5;
    }
    {
        System.out.println("Instance initialization.");
        nonStaticVariable = 7;
    }
    public Main() {
        System.out.println("Constructor.");
    }
    public static void main(String[] args) {
        new Main();
        new Main();   }}
```

```
Static initalization.
Instance initialization.
Constructor.
Instance initialization.
Constructor.
```

COMPARISON BETWEEN INSTANCE BLOCK AND CONSTRUCTOR-

Instance block	Constructor
Instance block logic are common for all the objects.	Constructor logic are specific to the objects.
Instance block will be executed only once for each object during its creation.	We can execute the constructors multiple times for a single object by placing the call to them in other constructors.
Instance block will be executed only as many times as there are number of objects created in the program.	It is not necessary for the constructors as they may be called explicitly.
Instance blocks are mainly used for writing the common logic which we want to execute for all the objects.	Constructors are mainly used for the initializing of objects.

EXAMPLE:



Output:

Static block-1
Static block-2
Instance block-1
Instance block-2

THANK YOU
?