Q>
Consider below code of Test.java file:

```
public class Test {
    public static void main(String[] args) {
        String [][] arr = { {"%", "$$"}, {"***", "@@@@", "#####"}};
        for(String [] str : arr) {
            for(String s : str) {
                System.out.println(s);
                if(s.length() == 4) //Line n1
                    break; //Line n2
            }
            break; //Line n3
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. %

B. %
   $$

C. %
   $$
   ***

D. %
   $$
   ***
   @@@@

E. %
   $$
   ***
   @@@@
   #####

Answer: B

Q>
What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        int [] arr = {2, 1, 0};
        for(int i : arr) {
            System.out.println(arr[i]);
        }
    }
}
```

A. 2
   1
   0

B. 0
   1
   2

C. Compilation Error.

D. ArrayIndexOutOfBoundsException is thrown at runtime.

Answer: B

Q>
What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        int [] arr = {3, 2, 1};
        for(int i : arr) {
            System.out.println(arr[i]);
        }
    }
}
```

A. 3
   2
   1
B. 1
   2
   3
C. Compilation Error
D. ArrayIndexOutOfBoundsException is thrown at runtime

Answer: D

Q>
What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        for:
        for (int i = 2; i <= 100; i = i + 2) {
            for(int j = 1; j <= 10; j++) {
                System.out.print(i * j + "\t");
            }
            System.out.println();
            if(i == 10) {
                break for;
            }
        }
    }
}
```

A. Total 5 rows will be their in the output
B. Total 50 rows will be their in the output
C. Total 100 rows will be their in the output
D. Compilation Error.

Answer: D

Q>
What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        int i;
        outer:
        do {
            i = 5;
            inner:
            while (true) {
                System.out.println(i--);
                if (i == 4) {
```

```
                    break outer;
                }
            }
        } while (true);
    }
}
```

A. Print 5 infinite times
B. Print 5 once
C. Compilation Error
D. 5
   3
   2
   1

Constructor
++++++++++
 => A constructor is a method whose name is same as that of the classname
 => A constructor would not have a return type.
 => Constructors gets called during the creation of an object
 => Constructors are normally used to give meaningful value to the instance variables of the class.

Note: In a class, if we don't write any constructor only then compiler will add default consturctor to our class.

 => Default constructor would not supply any meaningful values to the instance variables of the class.
 => To supply meaningful values to the instance variables, we need to write "Parameterized constructor".

++++
Code
++++

```
class Student
{
      //instance variables
       String name;
       int age;
       float height;

      Student(String name, int age, float height)
      {
            this.name = name;
            this.age  = age;
            this.height = height;
      }

}
class Test
{
      public static void main(String[] args)
      {
                  //Constructing the object
                  Student std = new Student("sachin",49,5.5f);
```

```
                //getting the values from instance variables
                System.out.println("Name   is :: "+std.name);
                System.out.println("Age    is :: "+std.age);
                System.out.println("Height is :: "+std.height);
        }
}
```

Can a Constructor be Overloaded?
Ans. Yes, it is possible to Overload a constructor,but it is not a good practise to write zero argument constructor with a logic of
"initialization".

```
class Student
{
        //instance variables
         String name;
         int age;
         float height;

        Student(String name, int age, float height)
        {
                this.name = name;
                this.age  = age;
                this.height = height;
        }

        Student()
        {
                name   = "dhoni";
                age    = 41;
                height = 5.6f;
        }

}
class Test
{
        public static void main(String[] args)
        {
                //Constructing the object
                Student std1 = new Student("sachin",49,5.5f);

                //getting the values from instance variables
                System.out.println("Name   is :: "+std1.name);
                System.out.println("Age    is :: "+std1.age);
                System.out.println("Height is :: "+std1.height);

                System.out.println();

                //Constructing the object
                Student std2 = new Student();

                //getting the values from instance variables
                System.out.println("Name   is :: "+std2.name);
                System.out.println("Age    is :: "+std2.age);
                System.out.println("Height is :: "+std2.height);

                System.out.println();

                //Constructing the object
```

```
                Student std3 = new Student();

                //getting the values from instance variables
                System.out.println("Name   is :: "+std3.name);
                System.out.println("Age    is :: "+std3.age);
                System.out.println("Height is :: "+std3.height);
      }
}
```

```
Output
Name   is :: sachin
Age    is :: 49
Height is :: 5.5

Name   is :: dhoni
Age    is :: 41
Height is :: 5.6

Name   is :: dhoni
Age    is :: 41
Height is :: 5.6
```

Can we have normal method with the name same as classname and also constructor?
Ans. yes, it is possible, but the constructor will be called during the creation of object where as normal method should be called by the programmer explicitly.

eg::

```
class Student
{
      //instance variables
       String name;
       int age;
       float height;

      //Parameterized constructor
      Student(String name, int age, float height)
      {
            System.out.println("CALLING THE CONSTRUCTOR");
            this.name = name;
            this.age  = age;
            this.height = height;
      }

      //Normal method
      void Student(String name,int age, float height)
      {
            System.out.println("CALLING THE METHOD");
            this.name = name;
            this.age = age;
            this.height = height;
      }
}
class Test
{
      public static void main(String[] args)
      {
                //Constructing the object
```

```
                Student std1 = new Student("sachin",49,5.5f);

                //getting the values from instance variables
                System.out.println("Name   is :: "+std1.name);
                System.out.println("Age    is :: "+std1.age);
                System.out.println("Height is :: "+std1.height);
      }
}
```

```
Output
CALLING THE CONSTRUCTOR
Name   is :: sachin
Age    is :: 49
Height is :: 5.5
```

Can we Overload main()?
Ans. yes we can overload main(),but jvm will always call main() with the following signature

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

eg#1.

```
class Test
{
      public static void main(String[] args)
      {
            System.out.println("Inside String[] args");
      }
      public static void main(int arg)
      {
            System.out.println("Inside int arg");
      }
      public static void main()
      {
            System.out.println("Inside zero argument");
      }
}
```

```
output
Inside String[] args
```

static
++++++
=> This access modifier is appicable at
a. class
b. variable
c. method
d. block

variable : if we mark a variable as static, then those variables are recognized as "class level variables".
static variables are unique with respect to class, they are not w.r.t Object.
memory for static variables will be give in "MethodArea".
if we don't initialize the static variables, then memory for static variables will be taken care by "JVM".

```
eg#1.
class Student
{
	static String nationality = "IND";
	String name;
	int age;

	Student(String name,int age)
	{
		this.name = name;
		this.age  = age;
	}
}
class Test
{
	public static void main(String[] args)
	{
		Student std= new Student("sachin",49);
		System.out.println("Name        is :: "+std.name);
		System.out.println("Age         is :: "+std.age);
		System.out.println("Nationality is :: "+Student.nationality);
	}
}

Output
Name        is :: sachin
Age         is :: 49
Nationality is :: IND


method   : if we mark a method as static, then those methods can be called in 2
ways.
	a. using ClassName(best practise)
	b. using objectName


block    : we can mark a block with static access modifier.
	This block is manily meant for "initializing  the static variables".
	This block will be excuted only once, so we normally keep "Driving code"
in static block.



eg#1.
class Student
{
	String name;
	int age;
	static String nationality = "IND";

	Student(String name,int age)
	{
		System.out.println("Constructor got called");
		this.name = name;
		this.age  = age;
	}
	static
	{
```

```
            System.out.println("Static Block :: Loading of Student.class file");
      }
      public void dispStdDetails()
      {
            System.out.println("Inside instance method");
            System.out.println("Name is         :: "+name);
            System.out.println("Age  is         :: "+age);
            System.out.println("Nationality  is :: "+nationality);
      }

}
class Test
{
      static
      {
            System.out.println("Loading of Test.class file");
      }
      public static void main(String[] args)
      {
            System.out.println("Inside main()");

            Student std= new Student("sachin",49);
            std.dispStdDetails();

      }
}

Output
Loading of Test.class file
Inside main()
Static Block :: Loading of Student.class file
Constructor got called
Inside instance method
Name is         :: sachin
Age  is         :: 49
Nationality  is :: IND
```