Assignment – 15th (Static Keyword)

1. Why do we need static keyword in Java, Explain with an example?

Ans: - In Java, the *`static`* keyword is used to define a variable or method that is associated with the class as a whole, rather than with individual instances of the class. This means that when a variable or method is declared as *`static`*, it can be accessed directly through the class name without needing to create an instance of the class first.

The *`static`* keyword can be useful for sharing data between instances of a class, improving performance by avoiding the need to create unnecessary objects, and organizing code by making it clear which methods and variables are associated with the class as a whole.

Here's an example that demonstrates the use of the static keyword in Java:

```
public class Calculator {
        public static final double PI = 3.14159;
        public static int add (int a, int b) {
                return a + b;
        }
}
```

In this example, the `Calculator` class contains two *`static`* members: a constant *`PI`* and a method **`add`**.

Basically, the *`static`* keyword allows us to define class-level constants and methods that can be accessed directly through the class name, making it easier to organize and use our code.

2. What is class loading and how does the Java program actually execute?

Ans: - Class loading is a process that occurs when a Java program is executed. It is the process by which a Java program loads and initializes classes at runtime. When a Java program is executed, the Java Virtual Machine (JVM) loads the necessary classes into memory as they are needed, and the class loading process can be divided into three steps: loading, linking, and initialization:

1. **Loading: -** In the first step, the JVM looks for the bytecode of the class and loads it into memory. The class can be loaded from a file or a network source. When a class is loaded for the first time, the JVM creates a Class object for the class.
2. **Linking: -** In the second step, the JVM verifies the bytecode to ensure that it is valid and compatible with the JVM. Then, it prepares the class for execution by allocating memory for the static variables and resolving all the symbolic references to other classes, methods, and variables.
3. **Initialization: -** In the final step, the JVM executes the static initialization code of the class. The static initialization code initializes the static variables and can perform other setup tasks. This code is executed once, the first time the class is used in the program.

When a Java program is executed, the JVM first loads and initializes the *`Main`* class, which contains the *`main`* method. It then loads and initializes the classes that are referenced by the *`Main`* class, and so on, until all the required classes have been loaded.

Class loading is important because it enables features such as dynamic class loading and reflection in Java programs. It also allows for more efficient use of memory because classes are loaded and initialized only when they are needed.

3. Can we mark a local variable as static?

Ans: - In Java, we cannot mark a local variable as static because it is not allowed by the Java syntax. The ***`static`*** keyword is used to declare class-level variables and methods, not local variables. Local variables are created and destroyed each time a method is called, so marking a local variable as static would not make sense.

Static variables and methods belong to the class rather than the instance of the class, which means they can be accessed without creating an instance of the class. Local variables, on the other hand, are only accessible within the method in which they are declared.

So, to summarize, we cannot mark a local variable as static in Java because it is not valid syntax and does not make logical sense in terms of how local variables mark as static.

4. Why is the static block executed before the main method in Java?

Ans: - In Java, the ***`static`*** block is executed before the ***`main`*** method because the ***`static`*** block is designed to initialize the static variables of a class. When a Java program starts running, the Java Virtual Machine (JVM) loads the class definition of the main class into memory, and it performs certain tasks before the ***`main`*** method is executed.

One of these tasks is to initialize the static variables of the class, which includes executing any static blocks that are defined in the class. The static block is executed in the order in which it appears in the source code, and it is executed only once, when the class is first loaded into memory.

The ***`main`*** method, on the other hand, is the entry point of the program, which means it is executed after the static block has been executed and the static variables have been initialized. The ***`main`*** method is executed sequentially, just like any other method in Java.

5. Why is a static method also called a class method?

Ans: - A static method in Java is also called a class method because it belongs to the class rather than to any instance of the class.

When we define a static method in a Java class, we are declaring a method that can be called on the class itself, rather than on any specific instance of the class. This means that we do not need to create an object of the class in order to call the static method; instead, we can call the method directly on the class name.

```java
class Calculate {
        public static int add (int a, int b) {
                return a + b;
        }
}

public class Main {
        public static void main (String[] args) {

                int total = Calculate.add(10, 5);
                System.out.println(total);
        }
}
```

6. What is the use of static blocks in Java?

Ans: - In Java, the ***`static`*** block is used to initialize the static variables of a class and perform any other necessary setup that should happen when the class is loaded into memory. The static block is executed only once, when the class is first loaded, and it is executed before any other code in the class.

In other word, ***`static`*** blocks in Java are used to initialize static variables, perform setup tasks, and handle exceptions. They are executed only once, when the class is loaded into memory, and they can be useful for ensuring that the class is properly initialized before it is used. Here are some of the uses of static blocks in Java:

- **Initialize static variables: -** If a class has static variables that need to be initialized with some specific value, we can use a static block to set the initial values of these variables.
- **Perform setup tasks:** - We can use static blocks to perform any setup tasks that should be done when the class is loaded. For example, we can read configuration files, connect to databases, or perform any other necessary initialization.
- **Exception handling: -** If a class has static variables that need to be initialized using methods that can throw exceptions, we can use a static block to handle those exceptions.

7. Difference between Static and Instance variables.

Ans: - In Java, there are two types of variables: static variables and instance variables. Here are the main differences between them:

- Static variables belong to the class and are allocated memory only once, while instance variables belong to the object of the class and are allocated memory each time an object is created. Static variables are accessible directly by the class name and have a class-level scope, while instance variables are accessed through an object of the class and have an object-level scope.

- Static variables are defined with the keyword ***`static`*** and belong to the class, while instance variables belong to the object of the class and are defined without the ***`static`*** keyword.

```
public class MyClass {
        static int staticVariable;
        int instanceVariable;
}
```

- Static variables are accessible directly by the class name, while instance variables are accessed through an object of the class.

```
public class MyClass {
        static int staticVariable;
        int instanceVariable;
}

// Access static variable
MyClass.staticVariable = 10;

// Access instance variable
MyClass obj = new MyClass();
obj.instanceVariable = 20;
```

8. Difference between Static and Non-Static members.

Ans: - In Java, there are two types of class members: static and non-static (also called instance members). Here are the main differences between them:

- Static members belong to the class and are allocated memory only once, while non-static members belong to the objects of the class and are allocated memory each time an object is created.
- Static members (fields and methods) belong to the class and are defined using the ***`static`*** keyword, while non-static members belong to the objects of the class and are defined without the ***`static`*** keyword.
- Static members are allocated memory only once when the class is loaded into memory, while non-static members are allocated memory each time an object of the class is created.
- Static members are accessible directly by the class name, while non-static members are accessed through an object of the class.
- Static members have a class-level scope and can be accessed from any method in the class, while non-static members have an object-level scope and can be accessed only within the instance methods of the class.
- Static members are initialized to their default values at the time of class loading, while non-static members are initialized to their default values when an object is created.
- Non-static members participate in polymorphism, which means that they can have different implementations in different subclasses. Static members, on the other hand, do not participate in polymorphism.
- Static methods cannot be overridden, while non-static methods can be overridden.