ASSIGNMENT

Q1 ans-

In Java, the `static` keyword is used to define class-level members that are shared among all instances of the class. It is mainly used to create variables and methods that belong to the class itself, rather than to any specific instance of the class. This means that there is only one copy of the `static` member, regardless of how many objects (instances) of the class are created.

Here's an explanation of the `static` keyword with examples:

1. `static` Variables:

A `static` variable is also known as a class variable. It is shared among all instances of the class. When you modify the value of a `static` variable in one object, the change will be reflected in all other objects of that class. This is useful for maintaining a common value that is the same for all objects of the class.

```java
class MyClass {
    // Static variable
    static int count = 0;

    // Non-static variable
    int instanceVar;

    MyClass() {
        count++; // Increment count for each new instance created
        instanceVar = count; // Assign instanceVar the current count value
    }
}

public class Main {
    public static void main(String[] args) {
        MyClass obj1 = new MyClass();
        System.out.println("obj1 instanceVar: " + obj1.instanceVar); // Output: obj1 instanceVar: 1

        MyClass obj2 = new MyClass();
        System.out.println("obj2 instanceVar: " + obj2.instanceVar); // Output: obj2 instanceVar: 2

        System.out.println("Total instances created: " + MyClass.count); // Output: Total instances
created: 2
    }
}
```

2. `static` Methods:

A `static` method is also known as a class method. These methods belong to the class itself, not to any specific instance. They can be called directly using the class name without the need to create an object.

```java
class MathUtils {
    // A static method to calculate the square of a number
    static int square(int num) {
        return num * num;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        int result = MathUtils.square(5); // Calling the static method without creating an object
        System.out.println("Square of 5 is: " + result); // Output: Square of 5 is: 25
    }
}
```

In summary, the `static` keyword is used in Java to create class-level members that are shared among all instances of the class. It is commonly used for constants, utility methods, or variables that need to be maintained at the class level rather than the instance level.

Q2 ans-

Class loading is the process by which the Java Virtual Machine (JVM) loads the binary representation of a Java class into memory. When a Java program is executed, it goes through several phases, including class loading, linking, and initialization, before it can start executing the main method or other entry points of the application.

Here's an overview of the steps involved in the Java program execution process:

1. **Compilation:**

When you write a Java program, it is saved in a .java file. The first step is to compile this source code using the Java compiler (`javac`), which translates the human-readable Java code into bytecode, which is a platform-independent binary representation.

2. **Class Loading:**

Once the Java program is compiled, the JVM is responsible for loading the necessary classes into memory. Class loading is performed dynamically as the classes are needed during the execution of the program. There are three phases of class loading:

a. **Loading:** The class loader reads the binary data (bytecode) of the class from the .class file and creates a Class object representing the class in memory. It does not execute the static initializers yet.

b. **Linking:**

- **Verification:** The JVM verifies the bytecode to ensure it follows the Java language rules and does not violate security constraints.

- **Preparation:** Static fields are initialized with their default values (e.g., 0 for integers, null for objects). Memory for these fields is allocated.

- **Resolution:** This is the process of replacing symbolic references in the bytecode with direct references to the actual memory addresses of the referenced entities (e.g., methods, variables).

c. **Initialization:** Static initializers and static blocks are executed in the order they appear in the code. This is when static variables are assigned their explicit values.

3. **Execution:**

After class loading and initialization, the JVM starts executing the bytecode. The entry point for the program is usually the `main` method, which is a static method defined with the signature `public static void main(String[] args)`. The JVM invokes the `main` method, and the program execution proceeds from there.

4. **Runtime:**

During runtime, the JVM manages the program's memory, garbage collection, thread execution, and other runtime aspects to ensure proper execution and resource management.

In summary, the Java program execution starts with the compilation of the source code into bytecode. Then, the JVM loads the required classes, performs linking and initialization, and finally, the program starts executing from the `main` method or other entry points. The JVM handles the runtime environment and ensures the program's execution follows the Java language specifications.

Q3 ans-

No, in Java, you cannot mark a local variable as **`static`**. The **`static`** keyword is specifically used for class-level members (variables and methods) to indicate that they belong to the class itself rather than being tied to any particular instance of the class. Local variables, on the other hand, are used within methods, constructors, or blocks and are limited in scope to the block in which they are declared.

Q4 ans-

In Java, the `static` block is executed before the `main` method because it is a part of the class loading and initialization process. When a Java program is run, the JVM first loads the class into memory and then proceeds with the initialization phase. During this phase, `static` blocks are executed in the order they appear in the code.

Here's the sequence of events that occur during class loading and initialization:

- **Loading:** The JVM reads the bytecode of the class from the .class file and creates a `Class` object representing the class in memory. This Class object contains information about the class, including its methods, variables, and static blocks.
- **Linking:**
- **Verification:** The JVM verifies the bytecode to ensure it follows the Java language rules and does not violate security constraints.
- **Preparation:** Static fields are initialized with their default values (e.g., 0 for integers, null for objects). Memory for these fields is allocated.
- **Resolution:** Symbolic references in the bytecode are replaced with direct references to the actual memory addresses of the referenced entities (e.g., methods, variables).
- **Initialization:** Static initializers and static blocks are executed in the order they appear in the code. This is when static variables are assigned their explicit values and static blocks are executed.
- **Execution:** Finally, after the class loading and initialization phases, the JVM invokes the `main` method to start the program's execution.

Q5 ans-

A `static` method is also called a "class method" because it belongs to the class itself rather than any specific instance of the class. It is associated with the class definition rather than being tied to a particular object (instance) of the class. Therefore, it can be invoked using the class name without creating an instance of the class.

Here are some key characteristics of static methods (class methods):

- **Belongs to the Class:** Static methods are associated with the class itself, not with any specific instance of the class. They are defined at the class level, and there is only one copy of the static method that is shared among all instances of the class.
- **Direct Access:** As static methods are not associated with instances, they can be directly accessed using the class name. There is no need to create an object of the class to invoke a static method.
- **Cannot Access Instance Members:** Since static methods are not tied to any instance, they cannot directly access instance variables or instance methods of the class. Only other static members can be accessed directly within a static method.

Q6 ans-

The `static` block in Java is used to initialize the static members (variables or constants) of a class or to perform any other one-time tasks that need to be executed before the class is used. It is a block of code enclosed within curly braces `{}` and preceded by the `static` keyword.

Here are some key points and use cases for the `static` block:

- **Initialization of Static Members:** The primary use of the `static` block is to initialize static variables or constants. It allows you to perform complex computations or set values for static variables that cannot be done with simple initialization at the point of declaration.
- **Executed During Class Loading:** The `static` block is executed only once, when the class is first loaded into memory by the JVM. It ensures that the static members are initialized before any instance of the class is created or any static method is called.
- **No Access Modifier:** The `static` block does not have an access modifier like `public`, `private`, or `protected`. It is automatically accessible within the class and is executed in the order it appears in the code.

Q7 ans-

Static and instance variables are two types of variables in Java that serve different purposes and have distinct characteristics:

- **Static Variables:**
- Also known as class variables.
- Declared using the `static` keyword.
- Shared among all instances of the class; there is only one copy for the entire class.
- Initialized only once when the class is loaded into memory (during class loading and initialization).
- Can be accessed using the class name, without the need to create an instance of the class.
  Commonly used for constants, configuration parameters, or variables that need to be shared among all instances.

- **Instance Variables (Non-static Variables):**
- Declared within the class but outside any method or block.
- Each instance of the class (object) has its own copy of instance variables.
- Initialized when an object is created (when the constructor is called) and destroyed when the object is garbage collected.
- Can have different values for different instances of the class.
- Accessed through object references (instances) and not directly through the class name.
- Represent the state or attributes of individual objects created from the class.

Q8 ans-

The main differences between static (class-level) and non-static (instance-level) members in Java are as follows:

- **Ownership:**
- Static Member: Belongs to the class itself, and there is only one copy shared among all instances of the class.
- Non-static Member: Belongs to individual instances (objects) of the class. Each instance has its own copy of non-static members.
- **Accessing:**

- Static Member: Can be accessed using the class name directly, without creating an instance of the class.
- Non-static Member: Must be accessed through an object (instance) of the class.
- **Memory Allocation:**
- Static Member: Memory is allocated only once, during class loading and initialization.
- Non-static Member: Memory is allocated separately for each instance of the class.