Static keyword assignment

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

Answer: If we need a variable or method in more than one instances of class then we have to use static keyword before declaration of these variables and methods. Static keyword helps in memory management in java.

Example:

class Student{

    int a;//declaration of instance variable

    static int b; //declaration of static variable

    {

        a=10; //way of initialising a instance or non-static variable

    }

    static

    {

        b=20;//way of initialising a static variable

    }

    static void disp2(){

        System.out.println("static method of outside main class");

    }

}

public class Q3 {

    static void disp() {

        System.out.println("static method of main class");

    }

    public static void main(String[] args) {

        disp(); //way of calling static method of main class

        Student.disp2();//way of calling static method of outside main class

    }

}

1. What is class loading and how does the Java program actually executes?

Answer: In java, classloading is the process of loading class files into the JVM(Java Virtual Machine) at run time. It is a responsible for loading classes from various sources, such as the file system, network, and databases and working them available to the JVM for execution.

Java programs are executed through a multi-step process involving compilation, loading, and execution. Here's an overview of how a Java program is executed:

1. Writing the Java Code: You write the source code for your Java program using a text editor or an integrated development environment (IDE). Java programs are typically written in files with a .java extension.
2. Compilation: The Java source code is then compiled using the Java compiler (javac), which translates the human-readable source code into a platform-independent binary format called bytecode. Bytecode is a low-level representation of the Java program that can be understood by the Java Virtual Machine (JVM).
3. Bytecode Generation: The compilation process generates one or more bytecode files with a .class extension. Each .class file corresponds to a Java class within your program.
4. Class Loading: The JVM is responsible for loading the bytecode files into memory. The class loader subsystem of the JVM locates the required bytecode files and loads them dynamically as they are needed during program execution.
5. Bytecode Verification: Before executing the bytecode, the JVM performs bytecode verification to ensure that it is valid and doesn't violate any security or runtime constraints. This step helps prevent potentially harmful or malformed code from executing.
6. Just-In-Time (JIT) Compilation: At runtime, the JVM employs a Just-In-Time (JIT) compiler to further optimize the bytecode. The JIT compiler analyzes the bytecode and translates it into machine code specific to the underlying hardware platform, which can be executed directly by the processor. This optimization process helps improve the performance of the program.
7. Execution: Once the bytecode has been verified and optimized, the JVM starts executing the program from the entry point, typically the **main** method of the class specified during program launch. The JVM interprets the bytecode or executes the optimized machine code, depending on the specific JVM implementation.
8. Runtime Environment: During program execution, the JVM provides a runtime environment that manages memory allocation, garbage collection, thread management, and other essential runtime tasks. It ensures the proper execution of the Java program while providing a level of abstraction from the underlying operating system.
9. Program Termination: The Java program continues executing until it reaches the end or encounters an exception or a termination statement. At this point, the program terminates, and any allocated resources are released by the JVM.

Overall, the JVM acts as an intermediary layer between the Java program and the underlying hardware and operating system, enabling Java programs to be executed in a platform-independent manner.

1. Can we mark a local variable as static?

Answer: No

1. Why is the static block executed before the main method in java?

Answer: In Java, the static block is executed before the main method because it allows the initialization of static variables and the execution of other static code before the program starts executing the main method.

When a Java program is run, the Java Virtual Machine (JVM) first loads the class containing the main method. During this loading process, any static variables defined in the class are initialized and any static blocks are executed. This ensures that the necessary setup for the class is done before the main method is invoked.

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

Answer: static method is a method that belongs to a class rather than an instance of class. It means we can call it without creating the instance of class and the static method is also called class method.

1. What is the use of static blocks in java?

Answer: The static block is used to initialise the static variables and this block will be executed before the class method.

1. Difference between Static and Instance variables

Answer:

|  |  |
| --- | --- |
| Static variable | Instance variable |
| a) The ”static” keyword is used to declare the static variables. | a) Don’t use static keyword to declare the instance variable. |
| b) It’s memory will be allocated on method area. | b) It’s memory will be allocated on heap area. |
| c) It’s a globle/class variable. | c) It’s a non-static variable. |
| d) Static variables are initialised before the execution of program. | d) Instance variables are initialised during the execution of program. |
| e) If values don’t changes object to object then we need to use static variable. | e) If value changes object to object then we use instance variable. |

1. Difference between static and non static members

Answer:

|  |  |
| --- | --- |
| Static member | Non-static member |
| a) Static members are those which belongs to the class. | a) Non-static members are those which belongs to the object. |
| b) It can be made with the static keyword. | b) It can’t be made with static keyword. |
| c) It can be used before initialising the instance of class. | d) It can be used after initialising the instance of class. |