**Static Keyword Assignment**

1.Why do we need static keyword in java Explain with an Example?

In Java, the **static** keyword is used to create members (variables and methods) that belong to the class itself, rather than to instances of the class. This means that **static** members can be accessed without creating an instance of the class, and they are shared among all instances of the class.

**package** Static\_Keyword;

**public** **class** Static\_Key {

**public** **static** **int** *no1*;

**public** **static** **void** m1()

{

System.***out***.println("this is a static method");

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.***out***.println(*no1*); // static datamembers are access directly

*m1*(); // static block also access directly

}

}

**The *static* keyword is a non-access modifier in Java that is applicable for the following:**

1. Blocks
2. Variables
3. Methods
4. Classes

2. What is class loading and how does the java program actually executes?

Class loading is the process by which the Java Virtual Machine (JVM) loads class files into memory so that they can be executed. When a Java program is run, the JVM performs the following steps:

1. **Loading**: The class loader loads the class file into memory. This involves reading the class file from disk and storing the bytecodes in memory.

2. **Linking**:

- Verification: The JVM verifies the class file to ensure that it is valid and doesn't violate Java's security rules.

- Preparation: Static variables are allocated memory and initialized with their default values.

- Resolution: Symbols in the class file are resolved to references to other classes, methods, or fields.

3. **Initialization**: Static initializers and static variables are initialized. This is the stage where static blocks and initializations are executed.

4. **Execution**: The JVM starts executing the `main()` method of the class specified when the program was launched. From there, the program executes as per its logic, interacting with objects and calling methods as needed.

The class loading process is important because it allows Java programs to be dynamic and flexible. Classes can be loaded at runtime, allowing for features like dynamic class loading, reflection, and more.

3.can we mark a local variable as static

No, you cannot mark a local variable as `static` in Java. The `static` keyword is used to declare members of a class (variables and methods) that are associated with the class itself, rather than with any instance of the class.

Local variables are variables declared inside a method or a block of code and are not associated with a class. They exist only within the scope of the method or block in which they are declared. Making a local variable `static` would not make sense, as `static` variables are shared among all instances of a class, while local variables are specific to a particular invocation of a method or block of code.

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

In Java, the static block is executed before the `main` method because it is part of the class loading process. When a class is loaded by the JVM, static blocks are executed as part of the class initialization process, even before any instances of the class are created or any other methods are called.

The sequence of execution is as follows:

1. **Class loading**: The class loader loads the class into memory.

2. **Static block execution**: Static blocks are executed in the order they appear in the class, from top to bottom.

3**. `main` method execution**: After the static blocks are executed, the `main` method is called to start the program execution.

This order ensures that any necessary initialization code in the static blocks is executed before the program starts executing the `main` method or creating any instances of the class. This is useful for initializing static variables, setting up static resources, or performing any other tasks that need to be done before the program starts running.

**package** Static\_Keyword;

**public** **class** Static\_block {

**static**

{

System.***out***.println("this is static block");

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.***out***.println("this is main methhod");

}

}

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

A static method is also called a class method because it belongs to the class itself rather than to any specific instance of the class. This means that you can call a static method without creating an instance of the class. Static methods are associated with the class definition and can be accessed using the class name, similar to how static variables are accessed.

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

Static blocks in Java are used for initializing static variables or performing any one-time initialization tasks that need to be done when a class is loaded. They are executed only once, when the class is first loaded into memory by the JVM. Static blocks are declared using the static keyword followed by a pair of curly braces {} containing the initialization code.

**Use cases for static blocks:**

1. **Initializing static variables:** Static blocks can be used to initialize static variables, especially when the initialization requires complex logic or multiple steps.
2. **Loading native libraries:** Static blocks can be used to load native libraries using **System.loadLibrary()**.
3. **Initializing static resources:** Static blocks can be used to initialize static resources such as database connections, thread pools, or configuration settings.
4. **One-time initialization:** Static blocks are executed only once when the class is loaded, making them suitable for performing one-time initialization tasks.

7.Difference between Static and Instance variable

|  |  |
| --- | --- |
| **Static Variable** | **Instance Variable** |
| Shared by all instances of the class. | Each instance has its own copy. |
| Stored in the class memory area. | Stored in heap memory for each object. |
| Can be accessed directly using the class name. | Can only be accessed through a class instance. |
| Only one copy exists regardless of the number of instances of the class. | Each object will have its own copy of the instance variable. |

8.Difference between static and non static members

| **Static variable** | **Non static variable** |
| --- | --- |
| Static variables can be accessed using class name | Non static variables can be accessed using instance of a class |
| Static variables can be accessed by static and non static methods | Non static variables cannot be accessed inside a static method. |
| Static variables reduce the amount of memory used by a program. | Non static variables do not reduce the amount of memory used by a program |
| In Static variable Memory is allocated only once, at the time of class loading. | In non Static variable Memory is allocated each time an instance of the class is created. |
| Static variables Can be accessed from any part of the program. | Non Static variables Can be accessed only within the class or its instance. |
| Static variables Exists for the entire lifetime of the program. | Non Static variables Exists for the lifetime of the object. |
| Static variables Default value is assigned automatically. | Non Static variables Default value is not assigned automatically. |
| Static variables are shared among all instances of a class. | Non static variables are specific to that instance of a class. |
| Static variable is like a global variable and is available to all methods. | Non static variable is like a local variable and they can be accessed through only instance of a class. |

-