There are a few other reasons why you might want to use static methods:

- You can access static methods from outside of the class in which they are defined. This is not possible with non-static methods.
- Subclasses can override static methods, but non-static methods cannot.
- Static methods are executed when an instance of the class is created, whereas non-static methods are not.
- Static methods can be used to create utility classes that contain general-purpose methods.

Static Blocks

It is used to initialize static data members. It is used to initialize before the main method at the time of class loading. It gets executed only once when the class gets loaded. It is not necessary to execute it again when creating different objects after the first time.

Static Class

In Java, a "static class" is a class that can be instantiated without having to create an instance of the containing class. A static class is defined as a member of another class and can only access static members of the containing class.

How Java Program Actually executes:

Class Loading

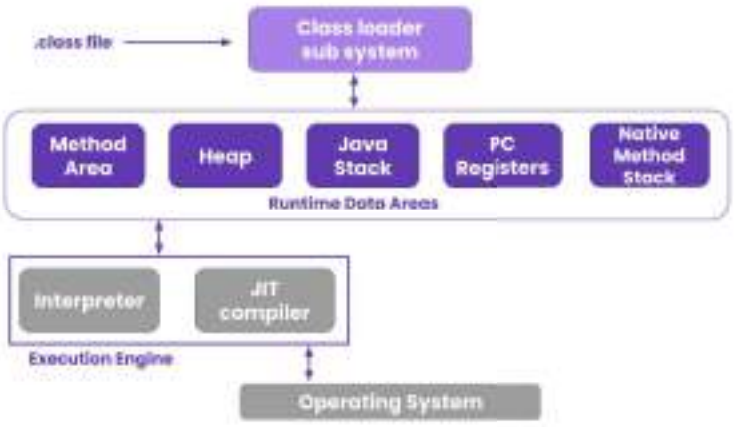
In Java, classloading is the process of loading class files into the JVM (Java Virtual Machine) at runtime. It is responsible for loading classes from various sources, such as the file system, network, and databases, and making them available to the JVM for execution.

The class loading process in Java is divided into three phases: loading, linking, and initialization.

1. Loading: In the loading phase, the classloader locates the class file using the fully qualified class name, reads the class file, and converts it into a Class object. The Class object contains the metadata of the class, such as the fields, methods, and constructors.

2. Linking: In the linking phase, the JVM performs several operations on the Class object, such as verifying the class file's integrity, resolving symbolic references, and allocating memory for the class variables.

3. Initialization: In the initialization phase, the JVM initializes the class variables with their default values, and runs the class's static initialization block (if any).



Principles of functionality of a Java classloader

A Java classloader's functionality is based on the following principles:

1. Delegation: A class loader delegates the responsibility of loading a class to its parent class loader before attempting to load it itself. This allows the class loader to take advantage of any classes that have already been loaded by the parent class loader.

2. Visibility: A class loaded by a class loader is only visible to that class loader and its child class loaders. This allows different class loaders to load different versions of the same class without interfering with each other.

3. Uniqueness: Each class is identified by its fully-qualified name (FQN) and is loaded by only one class loader. This ensures that the same class is not loaded multiple times by different class loaders.

4. Caching: A class loader caches the classes that it loads to improve performance. This allows the class loader to quickly return a class that has already been loaded, rather than having to load it again.

5. Security: A class loader enforces Java's security model by only allowing classes to access resources and execute code that is within their scope of permissions.

6. Extensibility: Java class loaders are extensible, allowing developers to create custom class loaders that can be used to load classes from specific locations or with specific behaviours.

7. Dynamic nature: Class loading is a dynamic process and can happen at runtime. Classes can be loaded, unloaded, and reloaded as the application runs.

8. Classpath: Java classloaders use the classpath to locate the classes that they need to load. The classpath is an ordered list of directories and JAR files that contain the class files.

9. Loading order: Java classloaders follow a specific order to load classes, first it checks the memory, if class is not found in memory it checks the cache, if class is not found in cache it checks the local file system, if class is not found in local file system it checks the network.

Different Members in the Java program

A Java program consists some of Members are:

1. Instance Member
2. Static Member

Instance Member: An instance member is essentially anything within a class that is not marked as static. That is, that it can only be used after an instance of the class has been made (with the new keyword). This is because instance members belong to the object, whereas static members belong to the class.

Static Member: Static members are those which belong to the class and you can access these members without instantiating the class. The static keyword can be used with methods, fields, classes (inner/nested), blocks.