**Basic Level:**

1. **Introduction to Java**
   * History of Java
   * Installing Java Development Kit (JDK)
   * Setting up the development environment (IDEs like IntelliJ IDEA, Eclipse, etc.)
   * Writing and running a basic Java program
2. **Java Syntax and Fundamentals**
   * Variables and data types
   * Operators
   * Control flow statements (if, else, switch, loops)
   * Arrays
3. **Object-Oriented Programming (OOP) Concepts**
   * Classes and objects
   * Constructors
   * Encapsulation
   * Inheritance
   * Polymorphism
4. **Exception Handling**
   * Try-catch blocks
   * Handling checked and unchecked exceptions
   * Throwing exceptions

**Intermediate Level:**

1. **Collections Framework**
   * Lists, Sets, Maps
   * Iterating collections
   * Sorting and searching
2. **File Handling**
   * Reading from and writing to files
   * Working with streams
3. **Generics**
   * Generic classes and methods
   * Bounded generics
4. **Concurrency**
   * Threads and multi-threading
   * Synchronization
   * Thread pools

**Advanced Level:**

1. **Java API**
   * String manipulation
   * Date and time API
   * Regular expressions
2. **Database Connectivity**
   * JDBC (Java Database Connectivity)
   * Connecting to databases
   * Executing SQL queries
3. **Java GUI Programming**
   * Swing or JavaFX
   * Creating graphical user interfaces
4. **Networking**
   * Socket programming
   * HTTP connections
5. **Design Patterns**
   * Creational, structural, and behavioral patterns
6. **Frameworks and Libraries**
   * Spring Framework
   * Hibernate ORM (Object-Relational Mapping)
   * Apache Maven
7. **Advanced Topics**
   * Lambda expressions
   * Streams API
   * Annotation processing
   * Reflection
8. **Project Work**
   * Applying the learned concepts to build real-world projects
9. **What is Java?**
10. **What are the main features of Java?**
11. **What is the difference between JDK, JRE, and JVM?**
12. **Explain the principle of OOP (Object-Oriented Programming) in Java.**
13. **What are the main pillars of OOP?**
14. **What is a class and an object in Java?**
15. **What is inheritance in Java?**
16. **What is the difference between interface and abstract class in Java?**
17. **What is method overloading and method overriding?**
18. **What is encapsulation?**
19. **What is a constructor?**
20. **What is the purpose of the static keyword in Java?**
21. **What is a package in Java?**
22. **Explain the concept of exception handling in Java.**
23. **What is the difference between checked and unchecked exceptions?**
24. **What are the access modifiers in Java?**
25. **What is the difference between public, protected, default, and private access modifiers?**
26. **What are the different types of loops in Java?**
27. **What is the difference between == and .equals() method in Java?**
28. **What are wrapper classes in Java?**
29. **What is the final keyword in Java?**
30. **What is a singleton class and how do you implement it in Java?**
31. **Explain the concept of multithreading in Java.**
32. **What is synchronization and why is it important in multithreading?**
33. **What is the difference between Thread class and Runnable interface?**
34. **What is a deadlock in Java?**
35. **What is JDBC and how do you connect to a database using JDBC?**
36. **What is serialization in Java?**
37. **Explain the concept of garbage collection in Java.**
38. **What is the purpose of the toString() method in Java?**

Will a constructor have a return type? And parameters?  
native stack method  
java static method

**Constructor in Java:**

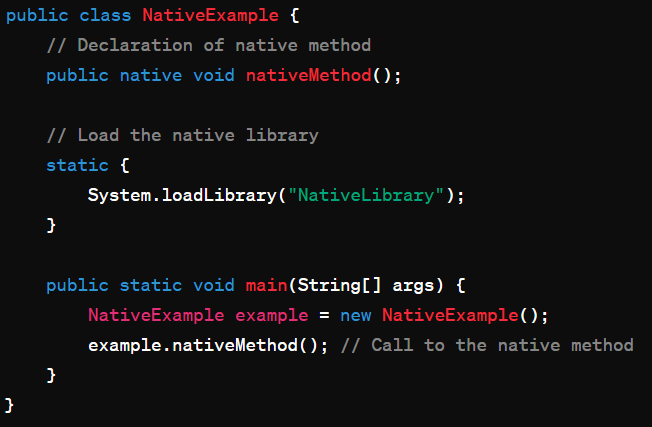
* A constructor in Java is a special type of method that has the same name as the class and is used to initialize objects of that class.
* Constructors do not have a return type, not even **void**. They are called implicitly when an object of the class is created using the **new** keyword.
* Constructors can have parameters, and these parameters can be used to initialize the instance variables of the object being created.
* If no constructor is explicitly defined in a class, Java provides a default constructor (parameterless constructor) with no implementation.

A computer screen shot of text

Description automatically generated

**Native Stack Method:**

* In Java, a "native" method is a method whose implementation is provided by code external to the Java Virtual Machine (JVM). It's written in a language other than Java, such as C or C++.
* Native methods are declared using the **native** keyword and do not have a method body. They typically end with a semicolon instead of curly braces.
* The native method declaration is followed by the implementation of the method in the target language, usually in a separate file.



**Java Static Method:**

* A static method in Java is a method that belongs to the class rather than to any instance of the class.
* Static methods are declared with the **static** keyword, and they can be called directly using the class name without creating an instance of the class.
* Static methods cannot access instance variables or instance methods directly; they can only access other static members of the class.
* Static methods are commonly used for utility methods or helper functions that do not require access to instance-specific data.

A computer screen shot of a code

Description automatically generated

In Java, a static block is a block of code enclosed within braces **{}** and preceded by the **static** keyword. It's used for static initialization of a class.

A static block is executed when the class is loaded into memory by the JVM (Java Virtual Machine). It runs exactly once, before the execution of the main method or the creation of any object of the class.

A computer code on a black background

Description automatically generated

In this example, **square()** is a non-static method of the **MathUtils** class. To call this method, we first create an object **math** of the **MathUtils** class using the **new** keyword. Then, we use the object **math** followed by the dot (**.**) operator to call the **square()** method.

Notice that the **square()** method is no longer declared with the **static** keyword. This means it is associated with instances of the **MathUtils** class rather than the class itself. Therefore, we need to create an instance (object) of the **MathUtils** class in order to call the **square()** method.

A computer screen shot of text

Description automatically generated