1. What is Java?

- **Answer**: Java is a high-level, object-oriented programming language developed by Sun Microsystems (now owned by Oracle). It is platform-independent, meaning Java programs can run on any platform that supports Java without modification. The slogan "Write Once, Run Anywhere" describes its portability across systems.

2. What are the features of Java?

- **Answer**:
  - **Platform Independent**: Java code is compiled into bytecode, which can be executed on any device with a JVM (Java Virtual Machine).
  - **Object-Oriented**: Java is based on the OOP principles like inheritance, polymorphism, encapsulation, and abstraction.
  - **Multithreaded**: Java provides built-in support for multithreading, enabling efficient CPU usage.
  - **Secure**: Java provides a secure environment using the bytecode verifier and the sandbox model.
  - **Robust**: Java has strong memory management, exception handling, and type checking, which reduce errors.
  - **Distributed Computing**: Java makes it easier to develop networked and distributed applications.

3. What is the JVM?

- **Answer**: The **Java Virtual Machine (JVM)** is the engine that runs Java bytecode. It provides platform independence by converting the bytecode into machine code specific to the system it is running on. JVM is responsible for memory management, garbage collection, and providing a runtime environment for Java programs.

4. What is the difference between JDK, JRE, and JVM?

- **Answer**:
  - **JDK (Java Development Kit)**: It is a full-featured software development kit that includes the JRE, compilers, debuggers, and other tools necessary for Java development.
  - **JRE (Java Runtime Environment)**: It provides libraries and the JVM needed to run Java applications. It does not contain development tools.
  - **JVM (Java Virtual Machine)**: It is part of the JRE and is responsible for executing Java bytecode. JVM makes Java platform-independent.

5. What are the primitive data types in Java?

- **Answer**: Java has 8 primitive data types:
  - **byte** (8-bit)
  - **short** (16-bit)
  - **int** (32-bit)
  - **long** (64-bit)
  - **float** (32-bit, single-precision)
  - **double** (64-bit, double-precision)
  - **char** (16-bit, Unicode character)
  - **boolean** (true or false)

6. What is the difference between == and .equals() in Java?

- **Answer**:
  - ==: Compares the memory addresses (references) of two objects. It checks if two variables point to the same object in memory.
  - **.equals()**: Compares the actual content of two objects. It is defined in the Object class and can be overridden for custom comparison logic.

7. What is a constructor in Java?

- **Answer**: A **constructor** is a special method that is automatically called when an object of a class is created. It initializes the newly created object. A constructor has the same name as the class and does not have a return type.
  - **Types of constructors**:
    - **Default Constructor**: A constructor that takes no arguments and provides default values to the object.
    - **Parameterized Constructor**: A constructor that takes arguments to initialize an object with specific values.

8. What is inheritance in Java?

- **Answer**: **Inheritance** is a mechanism in Java where one class (subclass/child) inherits the properties and behaviors (fields and methods) of another class (superclass/parent). This promotes code reusability and method overriding. Inheritance is implemented using the extends keyword.

  Example:

```
class Animal {
   void sound() {
      System.out.println("Animal makes sound");
   }
}
```

```
class Dog extends Animal {
   void sound() {
      System.out.println("Dog barks");
   }
}
```

9. What is polymorphism in Java?

- **Answer**: **Polymorphism** allows one entity (method, class) to take many forms. In Java, it is achieved through:
  - **Method Overloading** (Compile-time Polymorphism): Multiple methods with the same name but different parameters in a class.
  - **Method Overriding** (Runtime Polymorphism): Subclass provides a specific implementation of a method that is already defined in the superclass.

10. What is the difference between String, StringBuilder, and StringBuffer?

- **Answer**:
  - **String**: Immutable. Once a string object is created, its value cannot be changed.
  - **StringBuilder**: Mutable, but not thread-safe. Suitable for single-threaded environments.
  - **StringBuffer**: Mutable and thread-safe. Suitable for multi-threaded environments where string manipulation is needed.

11. What is exception handling in Java?

- **Answer**: **Exception handling** in Java is a mechanism to handle runtime errors, allowing the program to continue its execution. It is done using try, catch, finally, and throw/throws.
  - **try**: Contains code that may throw an exception.
  - **catch**: Catches and handles the exception.
  - **finally**: A block of code that always executes after the try and catch blocks, even if an exception occurs.
  - **throw**: Used to explicitly throw an exception.
  - **throws**: Declares an exception that might be thrown by a method.

12. What is an abstract class in Java?

- **Answer**: An **abstract class** in Java is a class that cannot be instantiated on its own and must be subclassed. It can have abstract methods (methods without implementation) and non-abstract methods (methods with implementation). The abstract methods must be implemented by subclasses.

13. What is the difference between ArrayList and LinkedList?

- **Answer**:
  - **ArrayList**: Resizes dynamically and provides fast random access with constant-time retrieval. It uses a contiguous block of memory.
  - **LinkedList**: A doubly-linked list with slower random access, but provides faster insertion and removal of elements, especially in the middle.

14. What is the purpose of the super keyword in Java?

- **Answer**: The **super** keyword refers to the superclass of the current object. It is used to:
  - Call a superclass constructor.
  - Access superclass methods or fields that are overridden in the subclass.

  Example:

```
class Animal {
   void sound() {
      System.out.println("Animal makes sound");
   }
}

class Dog extends Animal {
   void sound() {
      super.sound(); // Call Animal's sound method
      System.out.println("Dog barks");
   }
}
```

15. What is a final keyword in Java?

- **Answer**: The final keyword can be used in several contexts:
  - **Final variable**: The value cannot be changed once initialized.
  - **Final method**: The method cannot be overridden in subclasses.
  - **Final class**: The class cannot be subclassed (inherited).

16. What is garbage collection in Java?

- **Answer**: **Garbage collection** in Java is the automatic process of reclaiming memory from objects that are no longer in use. It is managed by the JVM to prevent memory leaks and free up memory for new objects. The most common garbage collector in Java is the **Mark-and-Sweep** algorithm.

17. What is multithreading in Java?

- **Answer**: **Multithreading** in Java allows the execution of multiple threads (independent units of execution) simultaneously. It enables better CPU utilization and performance, especially in tasks like parallel processing. Java provides the Thread class and Runnable interface to implement multithreading.

18. What is the Java API?

- **Answer**: The **Java API (Application Programming Interface)** is a collection of pre-written classes, libraries, and interfaces that provide various functionalities like data manipulation, network communication, file I/O, database connectivity, and more. It is a part of the Java Standard Library and is used to build Java applications. Common Java APIs include java.util, java.io, java.sql, and more.

19. What is the use of `java.sql` package in Java?

- **Answer**: The java.sql package provides the API for database connectivity and interaction. It includes classes and interfaces such as Connection, Statement, PreparedStatement, ResultSet, and DriverManager for working with relational databases like MySQL, Oracle, and others.

20. **JAVA-MYSQL Connection Points?**

- **DriverManager**: This is a part of the java.sql package and manages a list of database drivers. It attempts to select an appropriate driver from the set of registered drivers based on the database URL.
- **getConnection(url, username, password)**: This method of DriverManager is used to establish a connection to the database. It takes three arguments:

1. **url**: The connection URL, which specifies the location of the database. It also includes the type of database (MySQL, PostgreSQL, etc.) and other connection details (hostname, port number, database name).
2. **username**: The username used to authenticate with the database.
3. **password**: The password associated with the username to authenticate and establish the connection.

21. What is the JDBC?

JDBC (Java Database Connectivity) is a standard API (Application Programming Interface) in Java that allows Java applications to interact with databases. It provides a set of interfaces and classes that enable Java programs to connect to, query, update, and manage data in relational databases such as MySQL, PostgreSQL, Oracle, SQL Server, etc.

Here's a breakdown of key points about JDBC:

Purpose of JDBC:

- **Database Connection:** JDBC allows Java applications to connect to a database to retrieve or modify data.
- **Data Retrieval and Manipulation:** It provides methods to run SQL queries (like SELECT, INSERT, UPDATE, DELETE) and retrieve results from a database.
- **Error Handling:** It also provides tools for handling database-related exceptions and errors.

How JDBC Works:

JDBC acts as a bridge between Java applications and a database. Here's how it generally works:

- A **JDBC driver** is used to establish a connection between your Java program and a database.
- Once connected, you can use **SQL statements** to interact with the database.
- The result is returned as a **ResultSet**, which can be processed in your Java application.

JDBC Components:

- **DriverManager:** Manages a list of database drivers. It attempts to connect to the appropriate database driver based on the connection URL.
- **Connection:** This interface provides methods to establish and manage a database connection.
- **Statement:** This interface is used to execute SQL queries against the database.
- **PreparedStatement:** A subclass of Statement that is used for executing precompiled SQL queries with parameters, making it more efficient and secure.
- **ResultSet:** Represents the result of a query, containing rows and columns from the database.
- **SQLException:** A class that handles errors and exceptions that occur when interacting with a database.

22. What is Database Driver?

A **database driver** is a software component that allows an application to interact with a database management system (DBMS). It acts as a bridge between the application (such as a Java program) and the database, enabling the application to send SQL queries and retrieve data from the database.

Key Functions of a Database Driver:

1. **Translates Database Requests**: The driver translates the application's requests into database-specific commands.
2. **Establishes Connections**: It establishes and manages the connection between the application and the database.
3. **Sends Queries and Receives Data**: The driver sends SQL queries from the application to the database, and retrieves the results (or errors) back to the application.
4. **Manages Communication**: The driver handles the low-level communication and data formatting between the application and the database, ensuring the correct exchange of data.

How it Works: The Type-4 driver translates JDBC calls directly into the database's native protocol (such as MySQL's native protocol) and communicates with the database over the network.

Example: com.mysql.cj.jdbc.Driver (for MySQL),

(**cj** stands for **Connector/J**, which is the MySQL JDBC driver for Java.).

Simple JAVA MYSQL CONNECTION

```
import java.sql.*;

public class SimpleJDBCConnection {

   public static void main(String[] args) {

       String url = "jdbc:mysql://localhost:3306/cjits";
       String username = "root";
       String password = "";
       Connection conn = null;

       try {
          conn = DriverManager.getConnection(url, username, password);
          System.out.println("Connection successful!");

       } catch (SQLException e) {

          System.out.println("Error: " + e.getMessage());
       } finally {
          try {
             if (conn != null) {
                 conn.close();
                System.out.println("Connection closed.");
             }
          } catch (SQLException e) {
             System.out.println("Error while closing the connection: " +
e.getMessage());
          }
       }
   }
}
```