Q1. Explain JRE, JDK, and JVM

JVM (Java Virtual Machine)

- Executes Java bytecode.
- Provides a platform-independent runtime environment.

JRE (Java Runtime Environment):

- Runs Java applications.
- Includes the JVM and libraries for execution.

JDK (Java Development Kit):

- Develops and compiles Java applications.
- Includes the JRE and development tools.

Q2. Write the structure of a Java program and how to execute it.

Structure of a Java Program:

```
public class ClassName {
public static void main(String[] args) {
// Program statements
}
// Additional methods (optional)
}
```

Execution Steps:

1. Write Java code and save it with a `.java` extension.
2. Compile: `javac YourProgram.java` (creates a `.class` file).
3. Run: `java YourProgram`. The `main` method is executed.

2. `while` Loop:** Repeats code while a condition is true.
3. `do-while` Loop:** Similar to `while` but guaranteed to execute at least once.
4. `break`:** Exits loop or `switch`.
5. `continue`:** Skips rest of loop and moves to the next iteration.
6. `return`:** Exits a method and can return a value.

These statements control the program flow based on conditions and enable repetitive execution of code.

Q.6 Define a class in general and in Java's context.

In General:

- A class is a blueprint or template for creating objects in programming.
- It defines attributes (data) and methods (operations) that objects of the class will have.
- Objects are instances of a class and can interact with each other.

In Java's Context:

- In Java, a class is a fundamental unit of object-oriented programming.
- It includes fields (attributes), methods, a constructor, and modifiers for encapsulation.
- Objects are created from a class, and they encapsulate data and behavior.

Q7. Define constructor and types of constructor.

Constructor Definition:

- A constructor is a special method in a class that is automatically called when an object is created.
- It initializes the object's attributes and prepares it for use.

Types of Constructors:

1. Default Constructor:

Q3. Difference between C++ and Java. Explain types of data types

Difference between C++ and Java:

1. **Memory Management:**
- C++: Manual memory management.
- Java: Automatic garbage collection.

2. Platform Dependency:
- C++: Platform-dependent.
- Java: Platform-independent (JVM).

3. OOP:
- C++: Supports procedural and OOP.
- Java: Primarily OOP-focused.

4. Pointers:
- C++: Supports pointers.
- Java: No pointers.

5. Multiple Inheritance:
- C++: Supports multiple inheritance.
- Java: Uses interfaces.

6. Exception Handling:
- C++: `try`, `catch`, `throw`.
- Java: `try`, `catch`, `throw`, `finally`.

7. Header Files:
- C++: Uses header files.
- Java: No header files.

8. Operator Overloading:

- Has no parameters.
- Automatically provided by Java if no constructor is defined.
- Initializes fields to default values.

2.Parameterized Constructor:
- Takes parameters for initializing object attributes.
- Allows customization of object properties during creation.

3. Copy Constructor:
- Constructs an object by copying the attributes of another object.
- Creates a new object with the same values as an existing one.

Q7. Define finalize() method in Java.

The `finalize()` method in Java is a part of the `Object` class. It is called by the garbage collector before reclaiming an object to allow for cleanup and resource releasing. However, it's not recommended for critical resource management due to uncertainties in its invocation. As of Java 9, `finalize()` is deprecated, and alternative resource management mechanisms like `try-with-resources` are preferred.

Q8. Define inheritance and types of inheritance.

Inheritance:

Definition: Allows a class to inherit properties and behaviors.

Types:

1. Single (one superclass)
2. Multiple (via interfaces in Java)
3. Multilevel (chain of inheritance)
4. Hierarchical (multiple classes from one superclass)
5. Hybrid (combination of different types)

- C++: Supports operator overloading.
- Java: Doesn't support.

9. Global Functions:
- C++: Allows global functions.
- Java: No global functions.

10. Compilation/Execution:
- C++: Direct machine code.
- Java: Bytecode, interpreted or JIT-compiled by JVM.

Types of Data Types:
1. Primitive:
- Integers, floating-point, characters, boolean.

2. Derived:
- Arrays, classes/objects.

3. Enumerated
- Enum for named constant values.

4. Void Type:
- void for methods with no return.
5. User-Defined:
- Structures (C++), Classes (C++ and Java).

Q4. Explain operators in Java. What are all operators available in Java?
Certainly! Here's a shorter overview of Java operators:

1. Arithmetic Operators:
- `+`, `-`, `*`, `/`, `%`

2. **Relational Operators:**
- `==`, `!=`, `>`, `<`, `>=`, `<=`

3. Logical Operators:
- `&&`, `||`, `!`

4. Assignment Operators:
- `=`, `+=`, `-=`, `*=`, `/=`, `%=`

5. Increment and Decrement Operators:
- `++`, `--`

6. Bitwise Operators:
- `&`, `|`, `^`, `~`, `<<`, `>>`, `>>>`

7. Conditional Operator:
- `? :`

These operators are used for arithmetic, comparison, logical, assignment, bitwise, and conditional operations in Java.

Q5. Explain conditional and control statements.

Conditional Statements:
1. **`if`:** Executes code if a condition is true.
2. **`if-else`:** Executes one block if true, another if false.
3. **`if-else if-else`:** Checks conditions sequentially.
4. **`switch`:** Evaluates variable against cases.

Control Statements:
1. `for` Loop:** Repeats code for a specified number of times.

Q9. Define wrapper class and Vectors. Difference between them?

Wrapper Class:

- Represents primitive data types as objects.
- Exam

ples: `Integer`, `Double`.

- Used for conversion and compatibility in collections.

Vector:

- Dynamic array in Java Collections.
- Thread-safe but considered legacy.
- Largely replaced by modern alternatives like `ArrayList`.

Q10. What is Java Interface? Can we achieve multiple inheritance with the help of an interface? Give an example

Java Interface:

- A collection of abstract methods and constants.
- Defines a contract for classes to implement.

Multiple Inheritance with Interface:

- Yes, achieved through interfaces.
- A class can implement multiple interfaces.
- Example: `class FlyingFish implements Flyable, Swimmable`.