OOPJ
Assignment-3

1. **Explain the components of the JDK**.

   Following are the components of JDK:

   a. Java Compiler: It converts Java source code into platform-independent bytecode, which can be executed by the JVM.
   b. Development Tools: It is used for developing applications.
   c. JRE (Java Runtime Environment): It is responsible for running Java applications. It consists of JVM and Core Java libraries.

      i. JVM (Java Virtual Machine): It executes the compiled Java bytecode.

      ii. Core Java Libraries: Provides essential libraries such as collections, networking, I/O, etc.

2. **Differentiate between JDK, JVM, and JRE.**

   JDK: JDK stands for Java Development Kit which provides the environment to develop and execute Java programs. JDK is a package that includes Development Tools to provide an environment to develop your Java programs and JRE to execute Java programs or applications.

   JRE: JRE stands for Java Runtime Environment, it is an installation package that provides an environment to run the Java program or application on any machine.

   JVM: JVM also known as Java Virtual Machine is a part of JRE. JVM is a type of interpreter responsible for converting bytecode into machine-readable code. JVM itself is platform dependent but it interprets the bytecode which is the platform-independent reason why Java is platform-independent.

3. **What is the role of the JVM in Java? How does the JVM execute Java code?**

   JVM is responsible for converting bytecode to machine-specific code and is necessary in both JDK and JRE. It is platform-dependent and performs many functions, including memory management and security.

   JVM plays a crucial role in executing Java code by transforming the platform-independent bytecode into machine-specific instructions.

   a. Java source code (.java) is compiled into bytecode (.class).
   b. Class loader loads the bytecode into the JVM memory.
   c. Bytecode verification ensures the code is safe and conforms to JVM standards.

d. Execution engine either interprets or JIT-compiles the bytecode into native machine instructions.
e. Garbage collection automatically manages memory by removing unused objects.
f. The JVM facilitates execution through thread management, memory management, and native interface.

4. **Explain the memory management system of the JVM.**

Two major concept in memory management are:

a. Memory Structure: Following are parts of memory structure
   - Heap: Stores objects and instances created during runtime.
   - Stack: Stores method calls and local variables for each thread.
   - Method Area: Holds class-level information, including bytecode, method data, constants, and static variables.
   - PC Register: Keeps track of the current instruction being executed for each thread.
   - Native method area: Supports execution of native (non-Java) code via the Java Native Interface (JNI).

b. Garbage Collector: It is responsible for automatically reclaiming memory by removing objects that are no longer in use. This helps prevent memory leaks and ensures efficient use of memory.

5. What is the JIT compiler and its role in the JVM? What is the bytecode and why is it important for Java?

The JIT Compiler is a component of the JVM that improves the performance of Java applications by compiling bytecode into native machine code at runtime. JIT Compiler uses cache memory to store compiled native code. When the same code is executed repeatedly, the JVM can quickly reuse the compiled machine code from the cache instead of recompiling it, which speeds up execution.

Bytecode is an intermediate representation of Java code. When Java source code (written in .java files) is compiled by the Java compiler (javac), it is converted into bytecode, which is stored in .class files. Bytecode is executed by the Java Virtual Machine (JVM).

Importance of Bytecode in Java:

- Platform Independence:Bytecode allows Java applications to be platform-independent. A Java program compiled into bytecode can run on any platform with a compatible JVM.
- Security: Bytecode runs within the JVM, which enforces various security checks to ensure that the code does not perform unsafe operations or access restricted resources.
- Portability: Java programs are compiled once into bytecode, which can then be distributed and executed on any device with a JVM.

6. Describe the architecture of the JVM.

a. Class Loader Subsystem: Loads Java classes into memory.

   Components: Bootstrap Class Loader, Extension Class Loader, and System Class Loader.

b. Runtime Data Areas: Manages memory during the execution of Java applications.

   Components: Includes the Heap (for objects), Stack (for method calls and local variables), Method Area (for class metadata), and Program Counter (PC) Register.

c. Execution Engine: Executes Java bytecode.

   Components: Includes the Interpreter (which interprets bytecode) and the Just-In-Time (JIT) Compiler (which compiles bytecode into native machine code for improved performance).

7. How does Java achieve platform independence through the JVM?

Java achieves platform independence through the JVM (Java Virtual Machine) by using the following process:

- Bytecode compilation: Java source code is compiled into an intermediate form called bytecode, which is platform-independent. This bytecode is stored in .class files.
- JVM: The JVM provides a platform-specific implementation that can execute Java bytecode. Each platform (Windows, Linux, macOS) has its own JVM implementation, but they all adhere to the same specification. JVM provides an abstraction layer between the Java bytecode and the underlying hardware and operating system. By doing this, the JVM ensures that Java programs behave consistently regardless of the environment in which they are running. This abstraction allows Java applications to be portable and run on any platform with a compatible JVM.
- Standard Libraries: Java provides a standard set of libraries (Java API) that are consistent across platforms. These libraries offer a uniform set of functionalities, such as I/O operations, networking, and data structures, which work the same way regardless of the underlying platform.

8. What is the significance of the class loader in Java? What is the process of garbage collection in Java?

Class Loader in Java is a critical component of the Java Virtual Machine (JVM) responsible for dynamically loading, linking, and initializing classes and interfaces at runtime.

Garbage Collection in Java is the process of automatically identifying and reclaiming memory occupied by objects that are no longer in use, freeing up resources and preventing memory leaks.

9. What are the four access modifiers in Java, and how do they differ from each other?

a. public: It allows access from any other class or package. Members (fields, methods, classes) marked as public are accessible everywhere in the application.
b. protected: It allows access within the same package and also from subclasses (even if they are in different packages). It restricts access to subclasses and classes within the same package.
c. Package level private (default): It allows access only within the same package. They are not visible outside the package.
d. Private: It restricts access to the class itself. Members marked as `private` are accessible only within the class they are defined in. They are not accessible from outside the class, including subclasses.

10. What is the difference between public, protected, and default access modifiers?

public: It allows access from any other class or package. Members (fields, methods, classes) marked as public are accessible everywhere in the application.

protected: It allows access within the same package and also from subclasses (even if they are in different packages). It restricts access to subclasses and classes within the same package.

default: It allows access only within the same package. They are not visible outside the package.

11. Can you override a method with a different access modifier in a subclass? For example, can a protected method in a superclass be overridden with a private method in a subclass? Explain.

The access modifier of the overriding method must be the same as or more accessible than the method in the superclass. This means that if a method in the superclass is protected, the overriding method in the subclass can be protected or public, but it cannot be private or package level private (default).

12. What is the difference between protected and default (package-private) access?

protected: It allows access within the same package and also from subclasses (even if they are in different packages). It restricts access to subclasses and classes within the same package.

default: It allows access only within the same package. They are not visible outside the package.

13. Is it possible to make a class private in Java? If yes, where can it be done, and what are the limitations?

Yes, it is possible to make a class private in java. For this we have to use private access modifier.

Private: It restricts access to the class itself. Members marked as `private` are accessible only within the class they are defined in. They are not accessible from outside the class, including subclasses.

14. Can a top-level class in Java be declared as protected or private? Why or why not?

In Java, a top-level class cannot be declared as protected or private. The access modifiers applicable to top-level classes are `public` and package level private (default), but not protected or private.

15. What happens if you declare a variable or method as private in a class and try to access it from another class within the same package?

If a variable or method is declared as private then ina class then it can be accessed from within the same class only.

16. Explain the concept of "package-private" or "default" access. How does it affect the visibility of class members?

protected: It allows access within the same package and also from subclasses (even if they are in different packages). It restricts access to subclasses and classes within the same package.

default: It allows access only within the same package. They are not visible outside the package.