CDAC Mumbai PG-DAC AUGUST 24

Assignment No-3.

Q.1 Explain components of the JDK.

→ (1) Java Runtime Environment (JRE):

The JRE provides the libraries & utilites needed to run Java programs. It includes the Java Virtual Machine (JVM), which excutes Java bytecode.

(2) Java Development Tools:

These tools help developers compile, debug, & package Java applications.

- Javac (Java compiler): Compiles Java source code into bytecode.
- Jar (Java Debugger): Create & manage ~~Developer~~ JAR files.
- Jdb (Java Debugger): Debugs Java programs.

(3) Java Libraries:

The JDK includes various libraries for tasks like:

(4) Utility libraries:

Additional libraries for tasks like:

- File & I/O
- Data Structures
- Math & Statistics.

(5) Documentation & Demo Programs:

The JDK includes documentation, tutorials, & demo programs to help developers learn & get started with Java.

② Difference between JDK, JRE, JVM

→ ① JDK (Java Development Kit):

- A software development kit that provides tools & libraries for developing, testing, & running Java applications.
- Includes JRE, JVM, & development tools like Javac, jar, jdb, & javadoc.

② JRE (JAVA Runtime environment):

- A software package that provides the libraries & utilites needed to run Java programs.
- Includes JVM, Java libraries & utilites like java, javaw, & javaus.
- Used by end-users to run Java applications.

③ Java (Java Virtual Machine)

- A virtual machine that executes Java bytecode.
- Provides a platform-independent environment for running Java programs.

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

→ The Java Virtual Machine (JVM) plays a crucial role in the Java by executing Java bytecode & providing a platform - independent environment for running Java programs. Here's how JVM executes Java code:

Steps:

① loading
② verification
③ Preparation
④ Resolution
⑤ Initialization
⑥ Execution

Execution process:

① Bytecode Interpretation:
② Just - In - Time (JIT) Compilation
③ Memory management.
④ Garbage collection
⑤ Security.

* JVM's key features:

① Platform independence
② memory management
③ Security
④ Dynamic loading.

④ Explain the memory management system of the JVM.

→

Memory management process:

① Memory allocation:

JVM allocates memory for objects, arrays, classes as needed.

② Memory Deallocation:

JVM reclaims memory occupied by objects no longer in use (garbej collection).

③ Garbage Collection:

JVM periodically identifiers & reclaims memory occupied by unreachable objects.

④ Memory compaction:

JVM compacts memory to reduce fragmentation & improve performance.

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

→ ① Improve performance by compiling frequently executed code into native machine code.

② Optimize code for the specifice CPU architecture.

③ Reduce interpretation overhead.

Importance of Bytecode:
1. Platform independence:
2. Efficient interpretation
3. Security
4. Dynamic loading
5. compactness:

Bytecode is important for Java:
1. Write once, Run anywhere
2. efficient execution
3. Security
4. Dynamic Nature:

6. Describes the architecture of the JVM.
→ 1. Class loader subsystem
2. Runtime Data areas:
   - Method area
   - Heap
   - Stack
   - Native
   - PC register
3. Execution engine
   - Intepreter
   - Just-in-time (JIT)
4. Native method interface
5. Security manager
6. Garbage collector
7. Thread management
8. Runtime environment
9. Native libraries.

(7) How does Java achive platform independance through the JVM?

→ Java achieves platform independance through the JVM in the following ways:

1. Bytecode
2. JVM interpretation
3. JVM runtime environment
4. Dynamic loading
5. Native method interface
6. Platform-independent Data Types:
7. JVM's platform-independant memory

*Heres a step-by step explanation:

1. Java code is compiled into bytecode (.class files)
2. The JVM loads the bytecode into into memory.
3. The JVM interprets the bytecode, converting it into platform-specific machine code.
4. JVM executes the machine code, using platform-independent libraries & API's.
5. The JVM dynamically loads classes & libraries as needed.
6. The JVM manages memory allocation & deallocation, ensuring platfor independance.

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

→ Class loader:

The class loader in Java is responsible for loading classes & their dependencies into the Java Virtual Machine (JVM). It's significance includes:

1. Dynamic loading:
2. Namespace management
3. Security
4. Flexibity

Garbage Collection:

1. Marking
2. Normal Deletion
3. Compacting:

1. Generational Garbage Collection:
   Divides objects into generations based on their lifespan.
2. Concurrent garbage collection:
   Runs garbage collection concurrently with the application.

Benefits of garbage collection include:

1. Memory safety: Prevents memory leaks & dangling pointers.
2. Reduced memory usage:
   frees up memory
3. Improved Performance:
   Reduces the need for manual memory management.

(10)

(9) What are the four access modifiers in Java & how do they differ from each other.

→ The four access modifiers in Java are:

(1) Public: Accessible from anywhere, both with & outside the class, package, or subclass.

(2) Private:
Accessible only within the same class, not from outside the class or its subclasses.

(3) Protected:
Accessible within the same class, its subclasses, & classes in the same package.

Here, a summary of their differences:

(1) Scope: Public has the widest scope, while private has the narrowest.

(2) Accessibility:
Public is accessible from from anywhere, while private is only accessible within within the same class.

(3) Inheritance:
Protected members are inherited by subclasses; while private numbers are not.

(10) What is the difference between public, protected & default access modifiers?

→

(1) Public:

- Accessible from anywhere (same class, same package, different package, subclass)
- No restrictions on access.

(2) Protected:

- Accessible within the same class & its subclasses.
- Accessible within the same package (but not from outside the package).
- More restrictive than public, less restrictive than default & private.

(3) Default (no modifier):

- Accessible within the same class & same package.
- Not accessible from outside the package or from subclasses.
- More restrictive that protected, less restrictive than private.

(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.

- A public method can do be overridden with a public method.
- A protected method can be omidden with a protected or public with a default or protected method.

However, you cannot overide a method with a less accessible access modifier. for example:

- A public method cannot be overridden with a protected, default, or private method.
- A protected method cannot be overidden with default or private method.

(12) What is the difference bet.n protected & defaults (package-private) access?

→ Protected:

- Accessible within the same class
- Accessible within subclass
- Accessible within the same package.

Default (package - private):

- Accessible within the same class.
- Accessible within the same package
- Not accessible from subclass outside the package

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

→ However,

you can declared a nested class (a class inside another class) as private.

Here's an example:

```
public class OuterClass {
    private class InnerClass {
        // InnerClass is only accessible within
OuterClass
    }
}
```

In this case, InnerClass is only accessible within OuterClass & not from outside.

limitations:

- A private nested class can't be accessed from outside the enclosing class.
- A private nested class can't be used as a type for method parameters or return type.

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

→ Here's why:

- Protected access modifier: A protected class would imply that it's accessible from subclasses, but since a top-level class can't be subclassed from outside its package, this access modifier doesn't make sense.
- Private access modifier: A private class would imply that it's only accessible within the same class, but since a top level class is already a seperate entity, declaring it private would mean it can't accessed at all, which is not usefull.

Here's a summary:

- Public: Accessible from anywhere
- Package: private (default): Accessible within the same package
- Protected: Not allowed for top-level classes.
- Private: Not allowed for top-level classes.

Q.15 What happens if you if declared a variable or method as private in a class a ~~any~~ to classess it from another class within the same package?

→ If you declared a variable or method as private in a class you cannot access it directly from another class, even if it's within the same package.

Private members are only accessible within the same class, not from outside the class, including classess in the same package.

To access the private number, you would need to:

(1) Make the member public or package-private (default access)

(2) provide a public getter or setter method in the original class to access the private member.

(3) Use inheritance (if) & access the member through a protected accessor.

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

→ Package-private access is useful when:

- You want to encapsulation implementation details within a package.
- You want to allow classes within the same package to share resources or functionality.
- You want to hide internal implementation details from external packages.

Effect on visibility:

- Visible to classes within the same packages
- Not visible to classes in other packages.
- Not visible to subclasses in other packages.