Date: / /

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

If you declare a variable or method as private in a class, it can not be accessed from any other class, even in same package. private members are only accessible within the class where they are defined.

```
ex- class A {
        private int myvar;

        private void mymethod(){

        }
    }

    class B {

        void method()
        {
         A  a = new A();

         a.myvar =5;  // error not accessible
         a.mymethod(); // error
        }
    }.
```

Date: / /

Q.16. Explain concept of "Package-private" access.

Package private is also known as default acess.
package-private members are accessible only within classes in the same package.

```
ex -> class A {
        int aa ; // Package-private variable
        void method(){ //package-p. method.
                          // code.
        }
      }

   -> class Binsamepackage {
        void method()
        {
          A  a = new A();
            a.aa = 5; // Accessible within
                           same pkg.
            a.method(); //accessible within
                           same pkg.
          }
        }

   -> class Cindiffpkg {
        void method1() {
           A  a = new A();
           a. aa = 5;    } // not accessible,
           a. method();  }   diffn package.
        }
      }.
```

4. Default (package - Private) -
- Accessible only within the same package.

Use - when you want access restricted to classes within the same package offering more encapsulation that protected.

Q.11 Can you override a method with a different access modifier in a subclass?
ex - can a protected method in a superclass be overridden with a private method in a subclass? Explain

- No, you can not override a method in a subclass with more restrictive access modifiers.

In Java, when you override a method the access level can not be reduced but can be made less restrictive.

ex - a protected method in superclass can be overridden with a public method in ~~a~~ subclass.
but can not be overridden with a private method.

Q.12) Is it possible to make a class private in Java? If yes, where it can be done, and what are the limitations.

- Yes, it is possible to make class private in java, but only if it is a nested (inner) class.

Location - A private class can be declared inside another class, known as nested or inner class.

Limitations - A private nested class is accessible only within the outer class
A top-level class (not nested) can not be declared private.

```
ex - class A {
         private class B {

             //

                              }

         }
```

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

- No, a top-level class in Java cannot be declared as protected or private.

Java restricts top-level classes to only two access levels ; public and default
(Package-private)

Date: / /

•) Namespace management -
It creates seperate namespaces for loaded classes, allowing for class isolation and the ability to load classes with the same name but different implementations.

•) Security - class loaders help in enforcing security policies by checking classes for permissions before loading.

- Process of Garbage Collector in java -
garbage collection in java is a automated process of managing memory by reclaiming unused objects.

1. Marking - Identifying objects that are still in use or not.

2. Deleting - Removing objects that are no longer reachable thus reclaiming their memory.

3. Compaction -
live objects are moved together to free memory space.

4. Garbage Collection Algorithm -
•) Serial collector -
Single-threaded and suited for small applications.

Date: / /

•) parallel collector - use multiple threads for better performance.

•) CMS collector - It aims to minimize pause times by doing most of the work concurrently.

Q.9. What are four access modifiers in java, and how they differ from each other.

1) public -
- Accessible from any other class in any package.

Use - when you want a class, method or variable to be visible everywhere.

2) protected -
Accessible within same package and by subclasses in other package.

Use - used to allow subclasses to access members while restricting access from unrelated classes.

3) Private -
Accessible only within the same class.

Use - when you want to encapsulate the data and restrict access from all other classes, including subclasses.

Date: / /

What is bytecode? and its Importance.
Bytecode is the intermidiate representation of Java source code that is produced by the Java compiler (Javac)
and it is the set of instructions that are executed by the JVM.

Importance of bytecode -
1. platform Independent -
Bytecode is the key to Java's platform Independence. It allows Java program to run on any platform that has a compatible JVM without modification.

2. Security -
Bytecode undergoes verification by the JVM before execution.
this verification process helps to prevent unathorized access to resource and ensures that the code does not perform unsafe operation.

3. portability
4. optimizations
5. Developement.
6. Enables dynamic Loading.

Q. 6 Describe JVM Architecture.

Date: / /

Q. 7) How does Java achieve platform independence through the JVM?
- Java achieve platform independence through JVM by -

1. Compilation to bytecode -
Java Source code is compiled into bytecode by the Java compiler. bytecode is platform independent, intermidiate representation that can run on any platform with a compatible JVM.

2. JVM Execution -
The JVM interprets or compile the bytecode into machine code that can be executed on the host system.

3. Write once, run anywhere -
JVM handles the translation of bytecode into platform-specific instructions, the same Java bytecode can run on any platform with a JVM,

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

- Significance of class loader -
i) Dynamic loading -
classes are loaded on demand, which conserves memory and speeds up the application startup.

Date: / /

3. Stack -
It stores method call information, including local variables, paramters and return addresses.
Each thread has its own stack, and the stack memory is released when a method call is completed.

4. Program counter (PC) register -
It keeps track of the address of the currently executing instruction on a thread. and the memory address of next instruction to execute.

5. Native method stack -
It manages memory for native methods (non-java), typically written in C/C++.

Garbage collection (GC) -
The JVM automatically handles memory deallocation using a garbage collector. which reclaims memory used by objects that are no longer referenced by the application.

Date: / /

Q.5. What are JIT compiler and its role in the JVM?

JIT compiler role in JVM -
The JIT (Just-In-Time compilers) is a crucial component of the JVM that improves the performance of java applications by compiling bytecode into native machine code at runtime.

Role of JIT compiler in the JVM.
1. performance optimization -
The jit compiler enhances execution speed by translating frequently executed bytecode into machine code, which the CPU can execute directly.

2. Adaptive optimization -
The JIT compiler monitors the execution of the code and optimizes the most frequently run methods. by compiling them into optimized machine code.

3. garbage collection Integration -
The JIT compiler works closely with the garbage collector to optimise memory usage and manage resources efficiently.

4. Compilation on demand -
The JIT compiler operates 'just-In-time.' meaning it compiles code at the moment it is needed.

Date: / /

③ Security - JVM provides a secure execution environment, using class loader and bytecode verifiees to prevent malicious code from accessing unathorized resources.

② JVM executes Java code -

1. Loading -
The JVM's class loader subsystem loads .class files into memory.
It can load classes from various sources like the local file system, network.

2. Linking - linking phase includes -
•) Verification -
Ensures the bytecode is correct and maintaing security.
•) preparation - allocates memory for static variables and sets default values.
•) Resolution - converts symbolic references in the bytecode into direct references.

3. Initialization -
The JVM initializes classes and objects, running static initializers and constructor.

Date: / /

4. Execution -
The JVM executes the bytecode using an Interpreter or a Just-in-time (JIT) compiler.

•) Interpreter - executes bytecode line by line and translating each instruction into machine code.
•) JIT compiler - compiles bytecode into native machine code, which can then be executed directly by the CPU, significantly improving performance.

5. Runtime - During execution, The JVM manages program resources, performs garbage collection, and handles exceptions.

Q.4 Memory management system of JVM.
1. Method Area -
- It includes class defination, method data, and static variables.
- It stores class-level information, such as class structures, method-area and constant pool information.

2. Heap -
It stores all objects and instance variables, it is the primary area for dynamic memory allocation.
The heap is subject to garbage collector, where unused objects are automatically removed to free up space.

Date: / /

Q.2 Difference between JDK, JVM and JRE.
- JDK - java developement kit.

It includes JRE, JVM, compiler, developement tools, libraries and documentation.
It is used for developing java applications and applets.
and it is required for developing, compiling, and debugging tools java applications.

JRE - Java runtime environment.

It includes JVM, core libraries, and other components necessary for running java applications.
It provides an environment to run java applications.
and it is required if you want to run java programs, not develop them.

JVM -
It is an abstract machine that enables a computer to run java programs by converting java bytecode into machine-specific code.
It includes the class loaders, runtime data areas, execution engines and garbage collector.
A core part of both JDK and JRE; it directly handles the execution of java bytecode on the host machine.

Date: / /

JDK - is a full developement kit that includes the JRE and additional tools for developing java programs.

JRE - is a subset of JDK that provides the environment to run that applications, including the JVM.

JVM - JVM is the engine that runs java bytecode, part of the JRE and it is responsible for making java platform independent.

Q.3. Role of the JVM in java and How JVM executes java code.
① The JVM is a crucial part of the java platform, serving as a engine that runs java applications.
Its primary role is to provide environment where java bytecode can be executed, allowing java applications to be platform-independent. (write once run anywhere)
• key functions -
① platform Independence -
The JVM allows java program to run on any device or operating system.

② Memory management -
It manages memory allocation and deallocation through a process known as garbage collection, which helps to reclaim memory used by objects that are no longer needed.

Assigment 3

Date: / /

Q Explain Components of JDK.
The JDK is a Comprehensive toolkit for java developement. Its main components include -

→ Java compiler (Javac) -
Converts java source code into bytecode.

→ java runtime Environment (JRE) -
JRE includes JVM and standard java libraries, It is used to running java applications.

→ java Virtual machine (JVM) -
Execute java bytecode.

→ Java API's (Application programming Interface) -
A set of libraries and classes that provide various Functionalities, Such as file I/O, networking, and GUI developement.

→ java debugger (jdb) -
A tool For debugging java programs

→ java documentation tool (javadoc) -
generates documentation from java source code comments.

→ java archieve (jar) tool -
This tool is used for packaging related class libraries and resources into a single, compressed file, making it easier to distribute applications.