**Assignment 3**

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

JDK is an implementation of any one of the given Java Platforms released by Oracle corporation: Standard Edition Java Platform, Enterprise Edition, Micro Edition.

JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) etc. to complete the development of a Java Application.

primary components of JDK

Appletviewer apt extcheck idlj jabswitch java javac Javadoc jar javafxpackager jarsigner javah javap javaws JConsole jdb jhat jinfo jmap jmc jps jrunscript jstack jstat jstatd keytool pack200 Policytool VisualVM wsimport xjc

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

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation: Standard Edition Java Platform, Enterprise Edition Java Platform, Micro Edition Java Platform

**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 also platform-dependent and performs many functions, including memory management and security. In addition, JVM can run programs that are written in other programming languages that have been converted to Java bytecode.

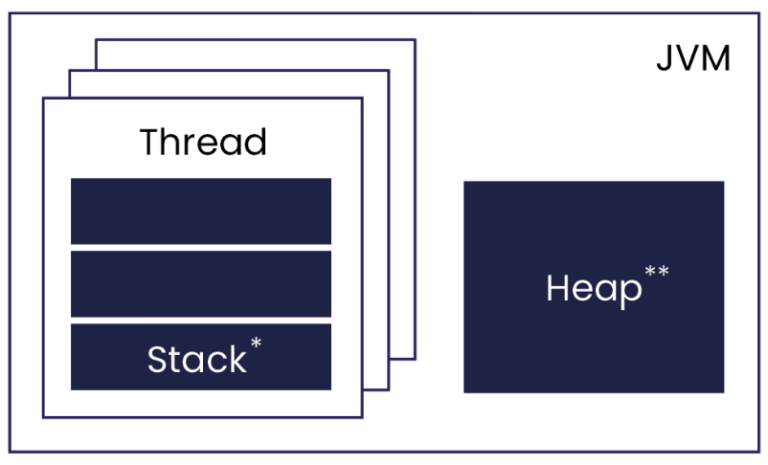
JVM consists of three main components or subsystems:

Class Loader Subsystem is responsible for loading, linking, and initializing a Java class file (that is, “Java file”), otherwise known as dynamic class loading.

Runtime Data Areas contain method areas, PC registers, stack areas, and threads.

Execution Engine contains an interpreter, compiler, and garbage collection area.

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

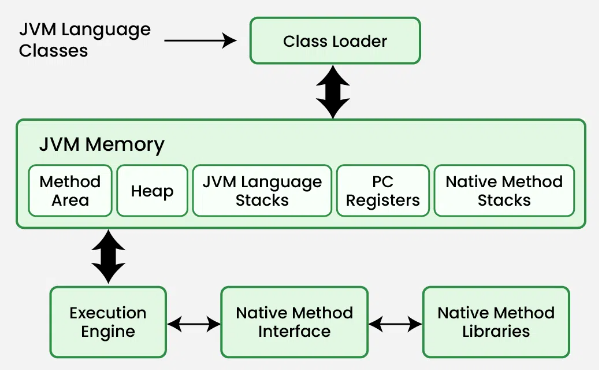
In Java, memory management is an automatic process that is managed by the Java Virtual Machine (JVM), and one that does not need explicit intervention. Java, being a block-structured language, uses a model where its memory is divided into two main types: stack and heap.

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

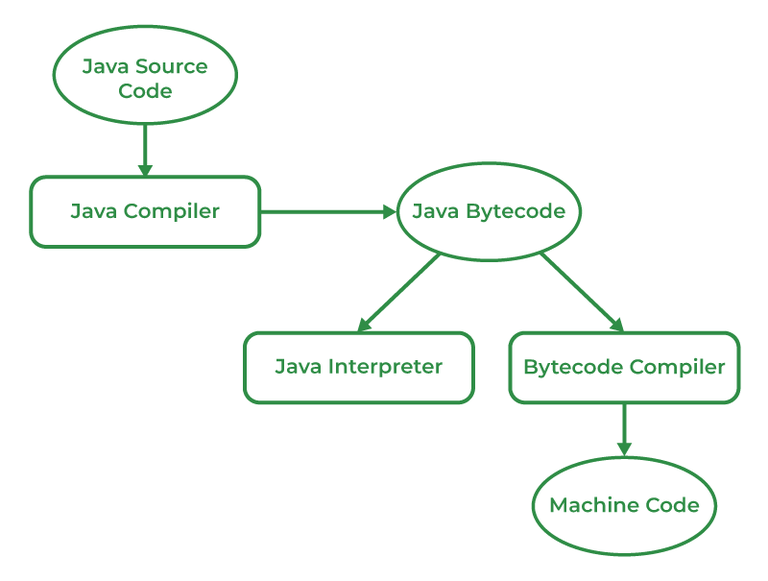
The Just-In-Time (JIT) compiler is a component of the runtime environment that improves the performance of Java™ applications by compiling bytecodes to native machine code at run time. Java programs consists of classes, which contain platform-neutral bytecodes that can be interpreted by a JVM on many different computer architectures. At run time, the JVM loads the class files, determines the semantics of each individual bytecode, and performs the appropriate computation. The additional processor and memory usage during interpretation means that a Java application performs more slowly than a native application. The JIT compiler helps improve the performance of Java programs by compiling bytecodes into native machine code at run time.

**6 Describe the architecture of the JVM.**

JVM(Java Virtual Machine) runs Java applications as a run-time engine. JVM is the one that calls the main method present in a Java code. JVM is a part of JRE(Java Runtime Environment). Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM. When we compile a .java file, .class files(contains byte-code) with the same class names present in .java file are generated by the Java compiler. This .class file goes into various steps when we run it. These steps together describe the whole JVM.



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

Whenever a program is written in JAVA, the java compiles it. The result of the JAVA compiler is the .class file or the bytecode and not the machine’s native code (unlike the C compiler). The bytecode generated is a non-executable code and needs an interpreter to execute on a machine. This interpreter is the JVM and thus the Bytecode is executed by the JVM. And finally, the program runs to give the desired output. In the case of C or C++ (languages that are not platform independent), the compiler generates an executable file (such as a .exe file) which is both OS-dependent and CPU-dependent. When we try to run this executable file on another OS or CPU architecture, it does not run, since it is specifically compiled for the target operating system and hardware architecture, making it incompatible with others.

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

Automatic memory management is implemented through a process known as garbage collection in Java, where manual memory deallocation is not required. Its purpose is to enhance the speed of applications and eliminate memory leaks by releasing memory occupied by unused objects.Java uses an automated garbage collection method. Automatic garbage collection entails heap memory scanning to determine which objects are being used and which are not and removing the latter.A referenced object, also known as an in-use object, signifies that a part of your program still holds a pointer to it. Once an object is no longer being used or referenced by any part of your program, the memory occupied by that object becomes available for reuse. It is not necessary for the programmer to specifically indicate objects for deletion. The JVM houses the garbage collection implementation.The JVM (Java Virtual Machine) keeps track of object references to find inaccessible objects (trash) and frees up memory for useful objects. Programmers can quickly destroy these objects with a garbage collector in Java.

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

Access modifiers in Java are keywords that determine the scope and visibility of classes, methods, variables, and constructors within an application. They are fundamental to object-oriented programming as they help enforce encapsulation, a core principle restricting direct access to some of an object's components. Java provides four main types of access modifiers: `public`, `private`, `protected`, and the default access (no modifier). The `public` modifier allows elements to be accessible from any other class in the application, regardless of the package. `Private` restricts access to the elements only within the class they are declared. `Protected` allows access within the same package or in subclasses, which might be in different packages. Lastly, the default access (no modifier) limits the visibility to classes within the same package. These modifiers help manage how data and methods are accessed and modified, playing a crucial role in safe and maintainable code development.

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

**Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

**Protected:** The access level of a protected modifier is within the package and outside the package through the child class. If you do not make the child class, it cannot be accessed from outside the package.

**Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

**Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

**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.**

Yes, an overridden method can have a different access modifier but it cannot lower the access scope. The following rules for inherited methods are enforced -

Methods declared public in a superclass also must be public in all subclasses.

Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.

Methods declared private are not inherited at all, so there is no rule for them.

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

Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from 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, we can declare a class as private but these classes can be only inner or nested classes. We can’t a top-level class as private because it would be completely useless as nothing would have access to it.

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

No, we cannot declare a top-level class as private or protected. It can be either public or default (no modifier). If it does not have a modifier, it is supposed to have a default access.

**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 you declare a variable or method as private in a class, it means that the variable or method can only be accessed from within the same class where it is declared. This access restriction is enforced regardless of the package the classes belong to.

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

The public access modifiers in Java have no restriction on the instance members, methods, and classes of the particular class. When you fix classes with public access specifiers in Java, you will be able to access them in any package or class. The scope of the public Java access modifiers is extremely flexible compared to other access modifiers. This type of access modifier in java is accessible only within the class.

The private access modifiers in Java are quite restricted. Data members and class methods prefixed with only private access modifiers can only be accessed inside the class. Even other classes inside the same package cannot reach the private methods or members.

Protected access modifiers and instance members can be used in the same package. You will be able to access them in another package using a child class. Therefore, you are required to extend the class containing protected members in another package. You won’t get direct access by creating the class object containing protected functions and members.

Protected access modifiers and instance members can be used in the same package. You will be able to access them in another package using a child class. Therefore, you are required to extend the class containing protected members in another package. You won’t get direct access by creating the class object containing protected functions and members.