Assignment 3

Date:06/05/2021
Submission
Date:08/05/2021

1. What is Java?

Java is a general-purpose, class-based, object-oriented programming language designed for having lesser implementation dependencies. It is a computing platform for application development. Java is fast, secure, and reliable, therefore. It is widely used for developing Java applications in laptops, data centers, game consoles, scientific supercomputers, cell phones, etc.

2. What is a package in Java? List down various advantages of packages.

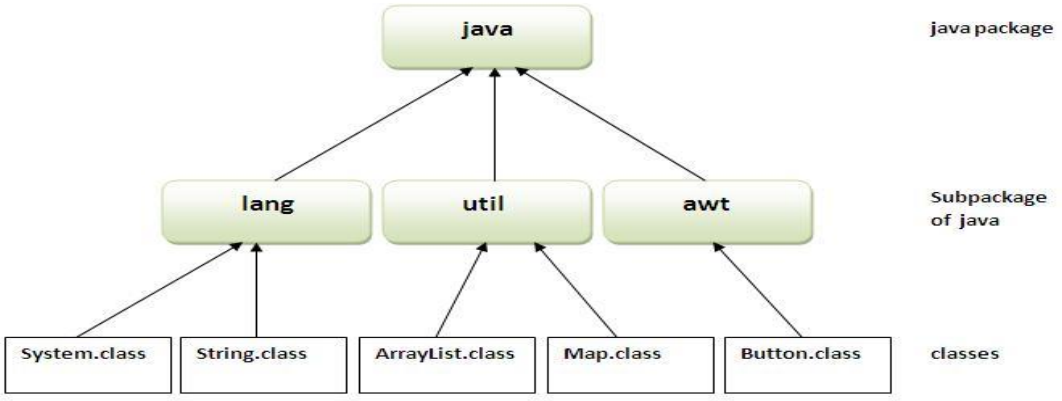
A package is a collection of classes. A package provides the space essentially used to organize classes that are related to each other. You can create multiple packages to organize multiple categories of classes, as per the requirement. Moreover, packages ensure that the non-related classes with the same name do not conflict with each other. A class can belong only to one package

The following syntax is used to define a package:

package <package_name>;

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.



3. Explain JDK, JRE and JVM?

JAVA DEVELOPMENT KIT

The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE),

an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) and other tools needed in Java development.

JAVA RUNTIME ENVIRONMENT
JRE stands for “Java Runtime Environment” and may also be written as “Java RTE.” The Java Runtime Environment provides the minimum requirements for executing a Java application; it consists of the Java Virtual Machine (JVM), core classes, and supporting files.

JAVA VIRTUAL MACHINE
JVM is an engine that provides a runtime environment to drive the Java Code or applications. It converts Java bytecode into machine language. JVM is a part of Java Run Environment (JRE). It cannot be separately downloaded and installed. To install JVM, you need to install JRE. The full form of JVM is Java Virtual Machine.

In many other programming languages, the compiler produces machine code for a specific system. However, Java compiler produces code for a virtual machine which is called as JVM.

4. Explain public static void main(String args[]) in Java.

class keyword is used to declare a class in java.
public keyword is an access modifier which represents visibility. It means it is visible to all.
static is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create an object to invoke the main method. So it saves memory.
void is the return type of the method. It means it doesn't return any value.
main is method name. it represents the starting point of the program.
String[] args is formal parameter of main method. it is used for command line argument.
System.out.println() is used to print statement. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class.

5. What are the differences between C++ and Java?

C++	Java
C++ is platform-dependent.	Java is platform-independent.
C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
C++ supports the goto statement.	Java doesn't support the goto statement.
C++ supports multiple inheritance.	Java doesn't support multiple inheritance

	through class. It can be achieved by interfaces in java.
C++ supports operator overloading.	Java doesn't support operator overloading.
C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
C++ supports structures and unions.	Java doesn't support structures and unions.
C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.

6. Why Java is platform independent?

Platform independence: In Java, when you compile an error-free code, the compiler converts the program to a platform independent code, known as the bytecode. Thereafter, the Java Virtual Machine (JVM) interprets this bytecode into the machine code that can be run on that machine. Converting a Java program into bytecode makes a Java program platform independent because any computer installed with the JVM can interpret and run the bytecode.

7. What are wrapper classes in Java?

Wrapper Classes : Variables that are declared by using the primitive data types are not objects. The primitive data types, such as int and char, are not a part of the object hierarchy. Therefore, in order to use the primitive data types as objects, Java provides wrapper classes. A wrapper class acts like an object wrapper and encapsulates the primitive data types within the class so it can be treated like an object.
The various wrapper classes, such as Boolean, Character, Integer, Short, Long, Double, and Float, are provided by the java.lang package. These classes are useful when you need to manipulate primitive data types as objects.

8. Why pointers are not used in Java?

Some reasons for Java does not support Pointers:
1. Memory access via pointer arithmetic: this is fundamentally unsafe. Java has a robust security model and disallows pointer arithmetic for the same reason. It would be impossible for the Virtual Machine to ensure that code containing pointer arithmetic is safe without expensive runtime checks.

2. Security: By not allowing pointers, Java effectively provides another level of abstraction to the developer. No pointer support make Java more secure because they

point to memory location or used for memory management that loses the security as we use them directly.

3. Passing argument by reference: Passing a reference which allows you to change the value of a variable in the caller's scope. Java doesn't have this, but it's a pretty rare use case and can easily be done in other ways. This is in general equivalent to changing a field in an object scope that both the caller and callee can see.

4. Manual memory management: you can use pointers to manually control and allocate blocks of memory . This is useful for some bigger applications like games, device drivers etc. but for general purpose Object Oriented programming it is simply not worth the effort. Java instead provides very good automatic Garbage Collection (GC) which takes care of memory management.

9. List some features of Java?

Simple: When an object of a class is created, memory needs to be allocated. Later on, when the object is no longer required, the allocated memory to the object should be released. In Java, memory allocation and deallocation happens automatically. The process of automatically deallocating memory is called garbage collection

Object-oriented: Java is an object-oriented language and it incorporates the various characteristics of an object-oriented language, such as inheritance and polymorphism. In Java, the entire program code must be encapsulated inside a class. Even the most basic program in Java must be written within a class. Furthermore, Java uses objects to refer to real world entities, such as a game or a game player..

Distributed: Java can be used for the development of those applications that can be distributed on multiple machines in a network, such as the Internet. These applications can be used by multiple users from multiple machines in a network. For this purpose, Java supports the various Internet protocols, such as Transmission Control Protocol and Internet Protocol (TCP/IP) and Hyper Text Transfer Protocol (HTTP).

Secure: Java has built-in security features that ensure that a Java program gains access to only those resources of a computer that are essential for its execution. This ensures that a Java program executing on a particular machine does not harm it.

Robust: Java provides various features, such as memory management and exception handling, which make it a robust programming language. These features ensure that the Java applications run securely and effectively on a machine and do not crash a machine if minor errors are encountered.

Multithreaded: Java provides the concept of multithreading that allows a program to simultaneously execute multiple tasks by using threads. A thread is the smallest unit of execution in a Java program.

10. Why is Java Architectural Neutral?

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.
In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

11. How Java enabled High Performance?

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.

12. Why Java is considered dynamic?

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.
Java supports dynamic compilation and automatic memory management (garbage collection)

13. What is Java Virtual Machine and how it is considered in context of Java's platform independent feature?

In Java, when you compile an error-free code, the compiler converts the program to a platform independent code, known as the bytecode. Thereafter, the Java Virtual Machine (JVM) interprets this bytecode into the machine code that can be run on that machine. Converting a Java program into bytecode makes a Java program platform independent because any computer installed with the JVM can interpret and run the bytecode.

14. List two Java IDE's?

Eclipse
NetBeans

15. Why Java is called as "Platform" ?

Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

16. Is Java Pure-Object oriented Language ?

NO

Pure Object Oriented Language or Complete Object Oriented Language are Fully Object Oriented Language which supports or have features which treats everything inside program as objects. It doesn't support primitive datatype(like int, char, float, bool, etc.).

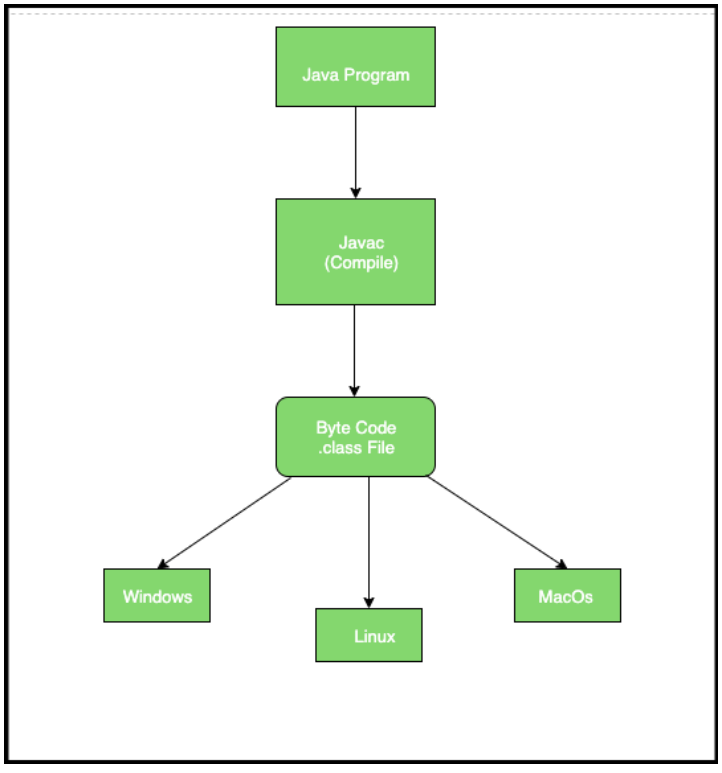
17. Which version of java have u learned? Name some of the new features added to it.

18. What gives Java its 'write once and run anywhere' nature?

Java applications are called **WORA (Write Once Run Anywhere)**. This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM.

In traditional programming languages like C, C++ when programs were compiled, they used to be converted into the code understood by the particular underlying hardware, so If we try to run the same code at another machine with different hardware, which understands different code will cause an error, so you have to re-compile the code to be understood by the new hardware.

In Java, the program is not converted to code directly understood by Hardware, rather it is converted to bytecode(.class file), which is interpreted by JVM, so once compiled it generates bytecode file, which can be run anywhere (any machine) which has JVM(Java Virtual Machine) and hence it gets the nature of Write Once and Run Anywhere.



19. Difference between path and classpath.

PATH − The path environment variable is used to specify the set of directories which contains executional programs. When you try to execute a program from command line, the operating system searches for the specified program in the current directly, if available, executes it. In case the programs are not available in the current directory, operating system verifies in the set of directories specified in the 'PATH ' environment variable.

CLASSPATH − The class path environment variable is used to specify the location of the classes and packages. When we try to import classes and packages other that those that are available with Java Standard Library. JVM verifies the current directly for them, if not available it verifies the set of directories specified in the 'CLASSPATH' environment variable.

20. What is the signature of main function in java ?

public static void main(String args[]);

21. What is the difference between JDK and JRE?

KEY	**JDK**	**JRE**
DEFINITION	JDK(Java Development Kit) is used to develop Java applications. JDK also contains numerous development tools like compilers, debuggers, etc.	JRE(Java Runtime Environment) is the implementation of JVM(Java Virtual Machine) and it is specially designed to execute Java programs.
FUNCTIONALITY	It is mainly used for the execution of code and its main functionality is development.	It is mainly used for creating an environment for code execution.
DEPENDENCY OF PLATFORM	It is platform-dependent.	It is also platform-dependent like JDK.
TYPE OF TOOLS	Since JDK is responsible	On the other hand, JRE is not

	for the development purpose, therefore it contains tools which are required for development and debugging purpose.	responsible for development purposes so it doesn't contain such tools as the compiler, debugger, etc. Instead, it contains class libraries and supporting files required for the purpose of execution of the program.
IMPLEMENTATION OF JDK AND JRE	JDK = JRE + other development tools.	JRE = JVM + other class libraries.

22. What is JVM ? What it does?

JAVA VIRTUAL MACHINE

JVM is an engine that provides a runtime environment to drive the Java Code or applications. It converts Java bytecode into machine language. JVM is a part of Java Run Environment (JRE). It cannot be separately downloaded and installed. To install JVM, you need to install JRE. The full form of JVM is Java Virtual Machine.

What it does

The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

23. Why JVM is called as "virtual machine"?

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.

24. What are the main components of JVM? Explain them. Or Explain JVM Architecture.

The Java architecture defines the components that are essential to carry out the creation and execution of code written in the Java programming language. The various components of the Java architecture are:

- Source file
- Class file
- JVM
- Application Programming Interface (API)

Source File

Java is a robust programming language that enables developers to build a wide variety of applications for varying purposes. In order to write a Java program or an application, you need to write certain code. A Java application contains the source code that enables an application to provide its desired functionalities. This source code is saved in a source file with the extension, .java.

Class File

Once created, a .java file is compiled to generate the .class file. A class file contains the bytecode that is generated by the compiler. Further, this class file can be executed by any machine that supports the Java Runtime Environment (JRE). JVM

The JVM is an application that is responsible for executing Java programs on a computer. It resides in the Random Access Memory (RAM) and is specific for every platform, such as Sun and Macintosh. The bytecode generated during the compilation of a program is the machine language of the JVM. JVM is responsible for executing the bytecode and generating the machine specific code for the machine on which the Java program needs to be executed.

The major components of JVM are:

- Class loader
- Execution engine and Just-In-Time (JIT) compiler
- Bytecode verifier

Class Loader The class loader loads the class files required by a program that needs to be executed. The classes are loaded dynamically when required by the running program.

Execution Engine and JIT Compiler

The Java execution engine acts as an interpreter that interprets the bytecode one line after another and executes it. The line-by-line interpretation is a time-consuming task and reduces the performance of a program. To enhance the performance, the JIT compiler has been introduced in the JVM. The JIT compiler compiles the bytecode into machine executable code, as and when needed, which optimizes the performance of a Java program.

Bytecode Verifier

Before being executed by the JVM, the bytecode is verified by using the bytecode verifier. The bytecode verifier ensures that the bytecode is executed safely and does not pose any security threat to the machine.

API

The Java API is a collection of software components, such as classes, interfaces, and methods, which provides various capabilities, such as GUI, Date and Time, and Calendar. The related classes and interfaces of the Java API are grouped into packages. A package is a container of classes.

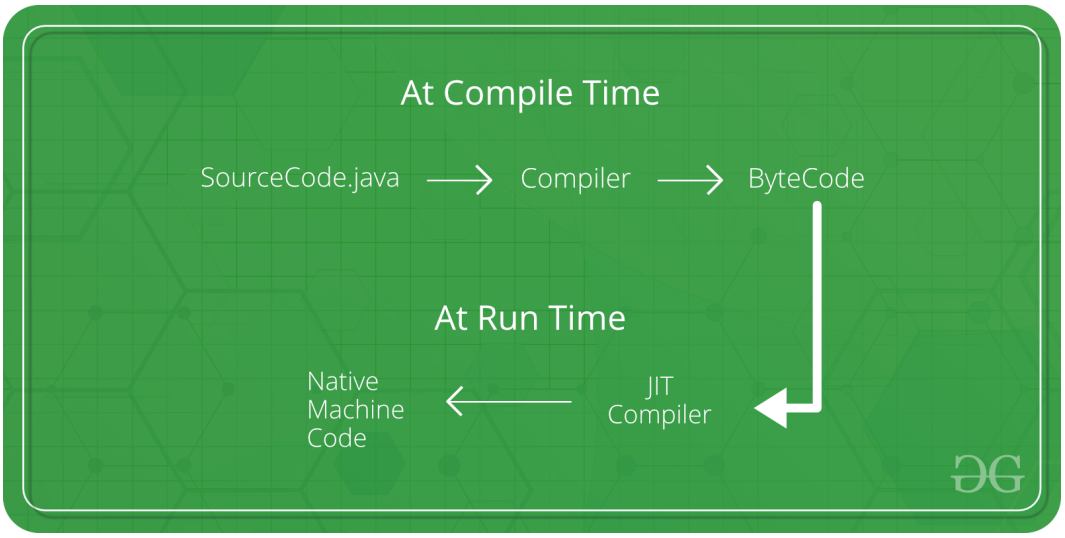
25. What is the difference between Java compiler (javac) and JIT ?

Java JIT Compiler :

Bytecode is one of the most important features of java that aids in cross-platform execution. Way of converting bytecode to native machine language for execution has a huge impact on the speed of it. These Bytecode have to be interpreted or compiled to proper machine instructions depending on the instruction set architecture. Moreover these can be directly executed if the instruction architecture is bytecode based. Interpreting the bytecode affects the speed of execution.

In order to improve performance, JIT compilers interact with the Java Virtual Machine (JVM) at run time and compile suitable bytecode sequences into native machine code. While using a JIT compiler, the hardware is able to execute the native code, as compared to having the JVM interpret the same sequence of bytecode repeatedly and incurring an overhead for the translation process. This subsequently leads to performance gains in the execution speed, unless the compiled methods are executed less frequently.

The JIT compiler is able to perform certain simple optimizations while compiling a series of bytecode to native machine language. Some of these optimizations performed by JIT compilers are data-analysis, reduction of memory accesses by register allocation, translation from stack operations to register operations, elimination of common sub-expressions etc. The greater is the degree of optimization done, the more time a JIT compiler spends in the execution stage. Therefore it cannot afford to do all the



optimizations that a static compiler is capable of, because of the extra overhead added to the execution time and moreover it's view of the program is also restricted.

26. Is Empty .java file name a valid source file name?

Yes, save your java file by .java then compile it by javac .java and run by java your classname

27. Is JRE different for different Platforms ?
YES

28. Difference between C++ and java in terms of object creation.

Java and C++ have a constructor, and they work the same way in Java as they do in C++. But, the way they are called is different.

For example, in Java, a constructor must be called by using the new() operator. There is no other way to explicitly call the constructor while creating an object, but in C++, you can do it without a new operator.

29. Who invokes main() function ?

The main method is executed/invoked by the JVM.

30. What is .class file known as ?

A Java class file is a file (with the .class filename extension) containing Java bytecode that can be executed on the Java Virtual Machine (JVM). A Java class file is usually produced by a Java compiler from Java programming language source files (.java files) containing Java classes (alternatively, other JVM languages can also be used to create class files). If a source file has more than one class, each class is compiled into a separate class file.

31. Can we define more than one public class in a java source code ? what is the rule of public class and file name . ?

No. because in on .java (source file) file only one public class is allowed.

Public classes are accessible anywhere in one package as well as in another package.

Filename of .java file should be same as the class name when we are using IDE.

On command line excecution file name of .java file can be anything.

32. What is JIT compiler?

The Just-In-Time (JIT) compiler is a an essential part of the JRE i.e. Java Runtime Environment, that is responsible for performance optimization of java based applications at run time. Compiler is one of the key aspects in deciding performance of an application for both parties i.e. the end user and the application developer.

Working of JIT Compiler

Java follows object oriented approach, as a result it consists of classes. These constitute of bytecode which are platform neutral and are executed by the JVM across diversified architectures.

At run time, the JVM loads the class files, the semantic of each is determined and appropriate computations are performed. The additional processor and memory usage during interpretation makes a Java application perform slowly as compared to a native application.

The JIT compiler aids in improving the performance of Java programs by compiling bytecode into native machine code at run time.

The JIT compiler is enabled throughout, while it gets activated, when a method is invoked. For a compiled method, the JVM directly calls the compiled code, instead of interpreting it. Theoretically speaking, If compiling did not require any processor time or memory usage, the speed of a native compiler and that of a Java compiler would have been same.
JIT compilation requires processor time and memory usage. When the java virtual

machine first starts up, thousands of methods are invoked. Compiling all these methods can significantly affect startup time, even if the end result is a very good performance optimization.

33. How many types of memory areas are allocated by JVM?

A Java virtual machine is an essential component that performs special types of tasks.

1. Loading of code
2. Verification of code
3. Executing the code
4. Providing a runtime environment for the users

All these functions take different forms of the memory structure. These data structures are:

- Heap
- Stack
- Program Counter Register
- Native Method Stack

Heap memory structure is usually implemented for allocating memory dynamically. Variables assigned with this type of memory structure can be allocated at runtime, but they have slow access to memory.

The stack memory structure is mostly implemented for providing static memory allocation. Programmers could make use of stack if they knew in advance how much memory needs to be allocated for the storage of data.

Program Counter Register: Programs are a set of instructions or orders feed to a computer for performing. These instructions are delivered to the processor by the program written by a human. The program counter register holds the address of the upcoming instructions to be executed.

Native methods form a stack that is primarily implemented to line up with your system calls as well as libraries scripted in different computer languages.

34. What is the rule for local member in java.

35. What are the various access specifiers in Java?

Java provides four types of access specifiers that we can use with classes and other entities.

These are:

1) **Default:** Whenever a specific access level is not specified, then it is assumed to be 'default'. The scope of the default level is within the package.

2) **Public:** This is the most common access level and whenever the public access specifier is used with an entity, that particular entity is accessible throughout from within or outside the class, within or outside the package, etc.

3) **Protected:** The protected access level has a scope that is within the package. A protected entity is also accessible outside the package through inherited class or child class.

4) **Private:** When an entity is private, then this entity cannot be accessed outside the class. A private entity can only be accessible from within the class.

Access Specifier	**Inside Class**	**Inside Package**	**Outside Package subclass**	**Outside package**
Private	Yes	No	No	No
Default	Yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

We can summarize the access modifiers in the following table.

Next, we will discuss each of these access specifiers in detail.

Default Access Specifiers

A default access modifier in Java has no specific keyword. Whenever the access modifier is not specified, then it is assumed to be the default. The entities like classes, methods, and variables can have a default access.

A default class is accessible inside the package but it is not accessible from outside the package i.e. all the classes inside the package in which the default class is defined can access this class.

Similarly a default method or variable is also accessible inside the package in which they are defined and not outside the package.

36. What is the rule for local member in java.

Rules for local members in java:

1) Name of the variable should be unique inside one block/scope of variable.
2) Local members are not accessible outside the block.
3) In class local members, access of members can be modify with the help of access specifiers or modifiers such as public private protected.
4) Name of the local member should be as per the naming convention.it should defined as per the identifier rules.

37. What is native code?

Native code is computer programming (code) that is compiled to run with a particular processor (such as an Intel x86-class processor) and its set of instructions. If the same program is run on a computer with a different processor, software can be provided so that the computer emulates the original processor. In this case, the original program runs in "emulation mode" on the new processor and almost certainly more slowly than in native mode on the original processor. (The program can be rewritten and recompiled so that it runs on the new processor in native mode.)

Native code can also be distinguished from bytecode (sometimes called interpreted code), a form of code that can be said to run in a virtual machine (for example, the Java Virtual Machine). The virtual machine is a program that converts the platform-generalized bytecode into the native code that will run in a specific processor. Microsoft's .NET compilers for its Visual Basic, C#, and JavaScript languages produce bytecode (which Microsoft calls Intermediate Language). Java bytecode and Microsoft's Intermediate Language can be compiled into native code before execution by a just-in-time compiler for faster performance.

38. Why there is no sizeof operator in java ?

Because the size of primitive types is explicitly mandated by the Java language. There is no variance between JVM implementations. Moreover, since allocation is done by the new operator depending on its argument there is no need to specify the amount of memory needed.
It would sure be convenient sometimes to know how much memory an object will take so you could estimate things like max heap size requirements but I suppose the Java Language/Platform designers did not think it was a critical aspect.

41.What is the job done by classloader ?

The **Java ClassLoader** is a part of the **Java Runtime Environment** that dynamically loads Java classes into the **Java Virtual Machine**. The Java run time system does not

need to know about files and file systems because of classloaders.

Java classes aren't loaded into memory all at once, but when required by an application. At this point, the **Java ClassLoader** is called by the **JRE** and these ClassLoaders load classes into memory dynamically.

Types of ClassLoaders in Java

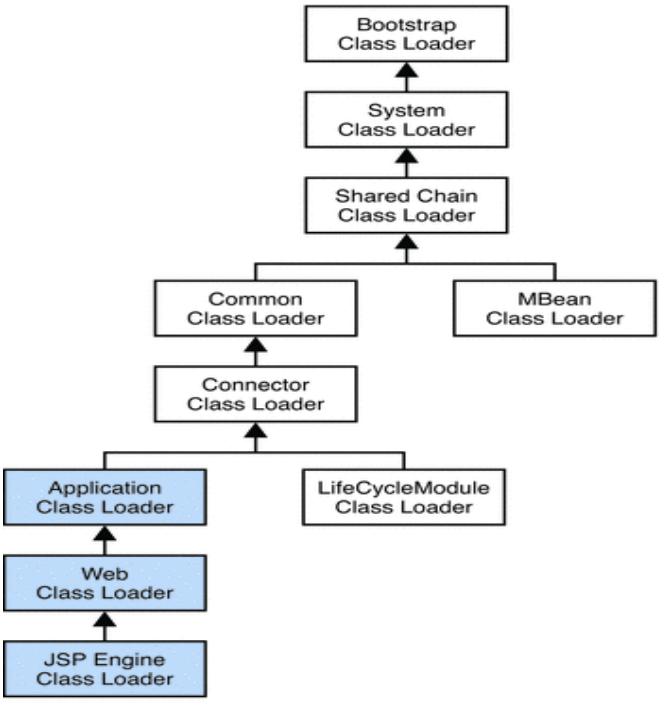
Not all classes are loaded by a single ClassLoader. Depending on the type of class and the path of class, the ClassLoader that loads that particular class is decided. To know the ClassLoader that loads a class the *getClassLoader()* method is used. All classes are loaded based on their names and if any of these classes are not found then it returns a **NoClassDefFoundError** or **ClassNotFoundException**.

A Java Classloader is of **three types**:

1. **BootStrap ClassLoader:** A Bootstrap Classloader is a Machine code which kickstarts the operation when the JVM calls it. It is not a java class. Its job is to load the first pure Java ClassLoader. Bootstrap ClassLoader loads classes from the location ***rt.jar***. Bootstrap ClassLoader doesn't have any parent ClassLoaders. It is also called as the **Primodial ClassLoader**.
2. **Extension ClassLoader:** The Extension ClassLoader is a child of Bootstrap ClassLoader and loads the extensions of core java classes from the respective JDK Extension library. It loads files from ***jre/lib/ext*** directory or any other directory pointed by the system property ***java.ext.dirs***.
3. **System ClassLoader:** An Application ClassLoader is also known as a System ClassLoader. It loads the Application type classes found in the environment variable ***CLASSPATH, -classpath or -cp command line option***. The Application ClassLoader is a child class of Extension ClassLoader.

39. Explain the hierarchy of classloaders in java.

Class loaders in the Application Server runtime follow a delegation hierarchy that is illustrated in the following.



The following table describes the class loaders in the Application Server.

Sun Java System Application Server Class Loaders

Class Loader	**Description**
Bootstrap	The Bootstrap class loader loads the basic runtime classes provided by the JVM, plus any classes from JAR files present in the system extensions directory. It is parent to the System class loader. To add JAR files to the system extensions, directory, see Using the Java Optional Package Mechanism.
System	The System class loader loads Application Server launch classes. It is parent to the Shared Chain class loader. It is created based on the system-classpath attribute of the java-config element in the domain.xml file. In the Admin Console, select the Application Server component, the JVM Settings tab, and the Path Settings tab, then edit the System Classpath field. See Using the System Class Loader and *java-config* in *Sun Java System Application*

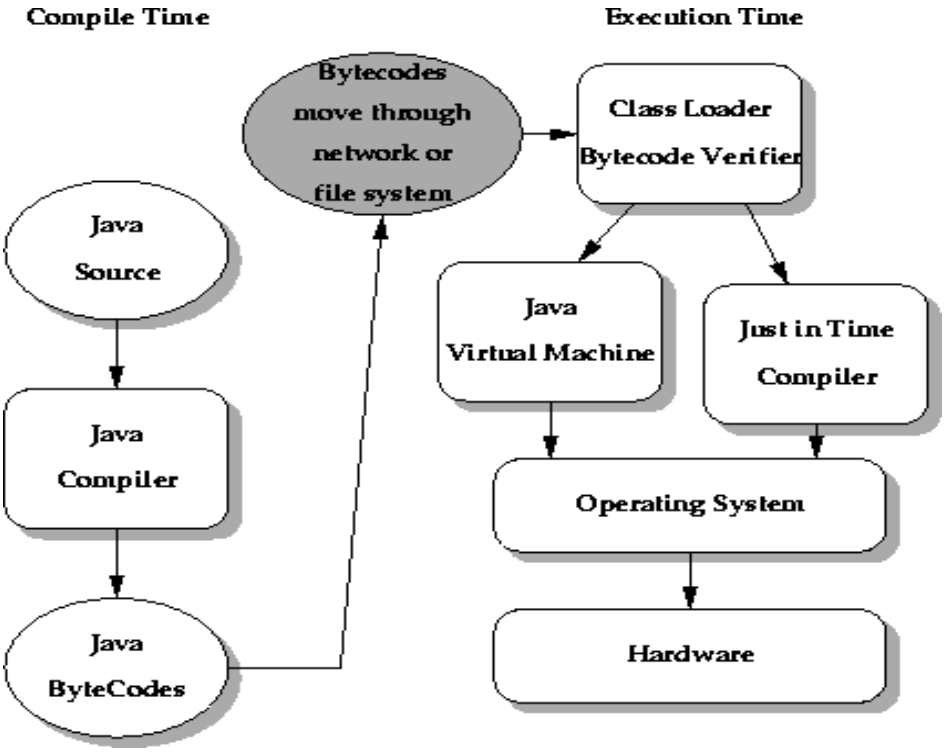
Class Loader	Description
	Server Platform Edition 9 Administration Reference.
Shared Chain	The Shared Chain class loader loads most of the core Application Server classes. It is parent to the MBean class loader and the Common class loader. Classes specified by the classpath-prefix and classpath-suffix attributes of the java-config element in the domain.xml file are added to this class loader. In the Admin Console, select the Application Server component, the JVM Settings tab, and the Path Settings tab, then edit the Classpath Prefix or Classpath Suffix field. The environment classpath is included if env-classpath-ignored="false" is set in the java-config element. Use classpath-prefix to place libraries ahead of Application Server implementation classes in the shared chain. The classpath-prefix is ideal for placing development and diagnostic patches. Use classpath-suffix to place libraries after implementation classes in the shared chain.
MBean	The MBean class loader loads the MBean implementation classes. See MBean Class Loading.
Common	The Common class loader loads classes in the *domain-dir*/lib/classes directory, followed by JAR files in the *domain-dir*/lib directory. It is parent to the Connector class loader. No special classpath settings are required. The existence of these directories is optional; if they do not exist, the Common class loader is not created. See Using the Common Class Loader.
Connector	The Connector class loader is a single class loader instance that loads individually deployed connector modules, which

Class Loader	Description
	are shared across all applications. It is parent to the LifeCycleModule class loader and the Application class loader.
LifeCycleModule	The LifeCycleModule class loader is created once per lifecycle module. Each lifecycle-module element's classpath attribute is used to construct its own class loader. For more information on lifecycle modules, see Chapter 13, Developing Lifecycle Listeners.
Application	The Application class loader loads the classes in a specific enabled individually deployed module or Java EE application. One instance of this class loader is present in each class loader universe; see Class Loader Universes. The Application class loader is created with a list of URLs that point to the locations of the classes it needs to load. It is parent to the Web class loader. The Application class loader loads classes in the following order: 1. Classes specified by the library-directory element in the application.xml deployment descriptor or the –-libraries option during deployment; see Application-Specific Class Loading 2. Classes specified by the application's or module's location attribute in the domain.xml file, determined during deployment 3. Classes in the classpaths of the application's sub-modules 4. Classes in the application's or module's stubs directory The location attribute points to *domain-dir*/applications/j2ee-apps/*app-name* or *domain-dir*/applications/j2ee-modules/*module-name*.

Class Loader	Description
	The stubs directory is *domain-dir*/generated/ejb/j2ee-apps/*app-name* or *domain-dir*/generated/ejb/j2ee-modules/*module-name*.
Web	The Web class loader loads the servlets and other classes in a specific enabled web module or a Java EE application that contains a web module. This class loader is present in each class loader universe that contains a web module; see Class Loader Universes. One instance is created for each web module. The Web class loader is created with a list of URLs that point to the locations of the classes it needs to load. The classes it loads are in WEB-INF/classes or WEB-INF/lib/*.jar. It is parent to the JSP Engine class loader.
JSP Engine	The JSP Engine class loader loads compiled JSP classes of enabled JSP files. This class loader is present in each class loader universe that contains a JSP page; see Class Loader Universes. The JSP Engine class loader is created with a list of URLs that point to the locations of the classes it needs to load.

43.What is the role played by Bytecode Verifier ?

The bytecode verifier traverses the bytecodes, constructs the type state information, and verifies the types of the parameters to all the bytecode instructions.



The illustration shows the flow of data and control from Java language source code through the Java compiler, to the class loader and bytecode verifier and hence on to the Java virtual machine, which contains the interpreter and runtime system. The important issue is that the Java class loader and the bytecode verifier make no assumptions about the primary source of the bytecode stream--the code may have come from the local system, or it may have travelled halfway around the planet. The bytecode verifier acts as a sort of gatekeeper: it ensures that code passed to the Java interpreter is in a fit state to be executed and can run without fear of breaking the Java interpreter. Imported code is not allowed to execute by any means until after it has passed the verifier's tests. Once the verifier is done, a number of important properties are known:

- There are no operand stack overflows or underflows
- The types of the parameters of all bytecode instructions are known to always be correct
- Object field accesses are known to be legal--private, public, or protected

While all this checking appears excruciatingly detailed, by the time the bytecode verifier has done its work, the Java interpreter can proceed, knowing that the code will run securely. Knowing these properties makes the Java interpreter much faster, because it doesn't have to check anything. There are no operand type checks and no stack

overflow checks. The interpreter can thus function at full speed without compromising reliability.

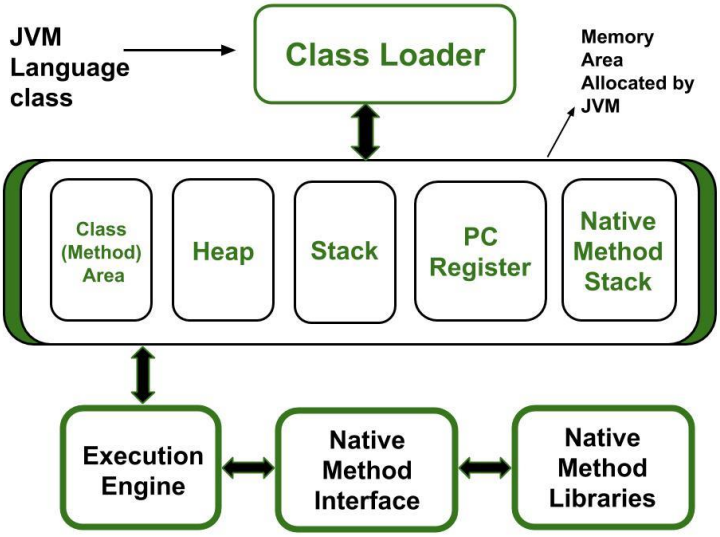
44.What are the memory areas allocated by JVM ?

JVM perform some particular types of operations:

- Loading of code
- Verification of code
- Executing the code
- It provide run-time environment to the users

Types of Memory areas allocated by the JVM:

All these functions take different forms of the memory structure. The memory in the JVM divided into 5 different parts:



Class(Method) Area

- Heap
- Stack
- Program Counter Register
- Native Method Stack

Let’s see about them in brief:

Class Loader:It is a subsystem of JVM which is used to load class files.It is mainly responsible for three activities.

Loading

Linking

Initialization

Class(Method) Area: It stores class level data of every class such as the runtime constant pool, field and method data, the code for methods.

Heap: It is used to allocate memory to objects at run time

Stack:

Each thread has a private JVM stack, created at the same time as thread. It is used to store data and partial results which will be needed while returning value for method and performing dynamic linking.

Java Stack stores frames and a new frame is created each time at every invocation of the method.

A frame is destroyed when its method invocation completes

Program Counter Register: Each JVM thread which carries out the task of a specific method has a program counter register associated with it. The non-native method has a PC which stores the address of the available JVM instruction whereas, in a native method, the value of the program counter is undefined. PC register is capable of storing the return address or a native pointer on some specific platform.

Native method Stacks: Also called as C stacks, native method stacks are not written in Java language. This memory is allocated for each thread when its created And it can be of a fixed or dynamic nature.

45.What kinds of programs u can develop using Java .

Below is the Java applications list:

- Desktop GUI Applications
- Mobile Applications
- Enterprise Applications
- Scientific Applications
- Web-based Applications
- Embedded Systems
- Big Data Technologies

- Distributed Applications
- Cloud-based Applications
- Web servers and Application servers
- Software Tools

46. When parseInt() method can be used?

While operating upon strings, there are times when we need to convert a number represented as a string into an integer type. The method generally used to convert String to Integer in Java is parseInt().
There are two variants of this method:
public static int parseInt(String s) throws NumberFormatException

- This function parses the string argument as a signed decimal integer.

public static int parseInt(String s, int radix) throws NumberFormatException

- This function parses the string argument as a signed integer in the radix specified by the second argument.

47. What is finalized() method ?

The finalize() method of Object class is a method that the Garbage Collector always calls just before the deletion/destroying the object which is eligible for Garbage Collection, so as to perform clean-up activity. Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection or we can say resource de-allocation. Remember it is not a reserved keyword. Once the finalize method completes immediately Garbage Collector destroy that object.

Syntax:

protected void finalize throws Throwable{}
Since Object class contains the finalize method hence finalize method is available for every java class since Object is the superclass of all java classes. Since it is available for every java class hence Garbage Collector can call the finalize method on any java object.

48. Difference between C++ pointer and Java reference.

Java doesn't have pointers; Java has references.
Reference:
A reference is a variable that refers to something else and can be used as an alias for that something else.
Pointer: A pointer is a variable that stores a memory address, for the purpose of acting as an alias to what is stored at that address.
So, a pointer is a reference, but a reference is not necessarily a pointer. Pointers are a particular implementation of the concept of a reference, and the term tends to be used only for languages that give you direct access to the memory address.
some keypoints about pointers and references in context of C/C++ and Java:

- **C/C++ allows pointer arithmetic but Java Pointers (References) not:** The term "pointer" is strongly associated with the C/C++ concept of pointers, which are variables which store memory addresses and can be modified arithmetically to point to arbitrary addresses.

In Java, pointers only exist as an implementation detail for References. A copy of the reference is copied to the stack of a called function, pointing to the same object as the calling function and allowing you to manipulate that object. However you cannot change the object the calling function refers to.

- **Java doesn't support pointer explicitly, But java uses pointer implicitly:** Java use pointers for manipulations of references but these pointers are not available for outside use. Any operations implicitly done by the language are actually NOT visible.
- **Pointers can do arithmetic, References can't**: Memory access via pointer arithmetic is fundamentally unsafe and for safe guarding, Java has a robust security model and disallows pointer arithmetic for this reason. Users cannot manipulate pointers no matter what may ever is the case.
- **Pointing objects:** In C, we can add or subtract address of a pointer to point to things. In Java, a reference points to **one thing only**. You can make a variable hold a different reference, but such c manipulations to pointers are not possible.
- **References are strongly typed:** Type of a reference is much more strictly controlled in Java than the type of a pointer is in C. In C you can have an int* and cast it to a char* and just re-interpret the memory at that location. That re-interpretation doesn't work in Java: you can only interpret the object at the other end of the reference as something that it already is (i.e. you can cast a Object reference to String reference only if the object pointed to is actually a String).
- **Manipulation of pointers can be dangerous:** On one hand, it can be good and flexible to have control over pointers by user but it may also prove to be dangerous. They may turn out to be big source of problems, because if used incorrectly they can easily break assumptions that your code is built around. And it's pretty easy to use them incorrectly.

So overall Java doesn't have pointers (in the C/C++ sense) because it doesn't need them for general purpose OOP programming. Furthermore, adding pointers to Java would undermine security and robustness and make the language more complex.

49. U have reference type as a member of class. What is the default value it gets?

Reference types hold references to objects and provide a means to access those objects stored somewhere in memory. The memory locations are irrelevant to programmers. All reference types are a subclass of type java.lang.Object.

Reference types

Reference type	Brief description
Annotation	Provides a way to associate metadata (data about data) with program elements.
Array	Provides a fixed-size data structure that stores data elements of the same type.
Class	Designed to provide inheritance, polymorphism, and encapsulation. Usually models something in the real world and consists of a set of values that holds data and a set of methods that operates on the data.
Enumeration	A reference for a set of objects that represents a related set of choices.
Interface	Provides a public API and is "implemented" by Java classes.

Default Values

Default values are the values assigned to instance variables in Java, when no initialization value has been explicitly set.

Instance and Local Variable Objects

Instance variables (i.e., those declared at the class level) have a default value of null. null references nothing.

Local variables (i.e., those declared within a method) do not have a default value, not even a value of null. Always initialize local variables because they are not given a default value. Checking an uninitialized local variable object for a value (including a value of null) will result in a compile-time error.

Although object references with a value of null do not refer to any object on the heap, objects set to null can be referenced in code without receiving compile-time or runtime errors:

```
Date dateOfParty = null;

// This will compile

if (dateOfParty == null) {

  ...

}
```

Invoking a method on a reference variable that is null or using the dot operator on the object will result in a java.lang.NullPointerException:

```
private static int MAX_LENGTH = 20;

...

String theme = null;

// Exception thrown, since theme is null

if (theme.length() > MAX_LENGTH) {

  ...

}
```

Arrays

Arrays are always given a default value whether they are declared as instance variables or local variables. Arrays that are declared but not initialized are given a default value of null.

In the following code, the gameList1 array is initialized but not the individual values, meaning that the object references will have a value of null. Objects have to be added to the array:

```
// The declared arrays named gameList1 and

// gameList2 are initialized to null by default

Game[] gameList1;

Game gameList2[];

// The following array has been initialized but

// the object references are still null since
```

```
// the array contains no objects
   gameList1 = new Game[10];
// Add a Game object to the list
// Now the list has one object
   gameList1[0] = new Game();
```

Multidimensional arrays in Java are actually arrays of arrays. They may be initialized with the new operator or by placing their values within braces. Multidimensional arrays may be uniform or nonuniform in shape:

```
// Anonymous array
int twoDimensionalArray[][] = new int[6][6];
twoDimensionalArray[0][0] = 100;
int threeDimensionalArray[][][] = new int[2][2][2];
threeDimensionalArray[0][0][0] = 200;
int varDimensionArray[][] = {{0,0},{1,1,1},{2,2,2,2}};
varDimensionArray[0][0] = 300;
```

Anonymous arrays allow for the creation of a new array of values anywhere in the code base:

```
// Examples using anonymous arrays
int[] luckyNumbers = new int[] {7, 13, 21};
int totalWinnings = sum(new int[] {3000, 4500, 5000});
```

50. What are the expressions allowed in switch block of java ?

The expression can be byte, short, char, and int primitive data types. Beginning with JDK7, it also works with enumerated types (Enums in java), the String class and Wrapper classes.