**1. What is Java?**

Java is a cross-platform object-oriented programming language that was released by Sun Microsystems in the year 1995. Today, Java is needed to run various applications such as games, social media applications, audio and video applications, etc.

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

A Java package organizes Java classes into namespaces, providing a unique namespace for each type it contains. Classes in the same package can access each other's package-private and protected members.

There are two types of packages

1)default: .lang, util etc…..

2)user defuned: folders which we made for saves the programs.

**3. Explain JDK, JRE and JVM?**

JDK is a software development kit whereas JRE is a software bundle that allows Java program to run, whereas JVM is an environment for executing bytecode. The full form of JDK is Java Development Kit, while the full form of JRE is Java Runtime Environment, while the full form of JVM is Java Virtual Machine.

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

public static void main (String [] args)in Java main is a method which is the entry point of any java program main is public because it is called from outside..main method always static bcoz main () is internally called without creating object...it is void bcoz it doesn't return any value.

**5. What are the differences between C++ and Java?**C++ and Java are most commonly used programming languages. Java has a strong influence of C++ as it was developed after C++ and both support OOP (Object Oriented Programming) paradigms. The crucial difference which differentiates both of the programming languages is that C++ is platform dependent while Java is platform independent. Java source code is converted into bytecode when compiled. At runtime, the interpreter executes this bytecode and gives output. Mostly Java is an interpreted language and therefore platform independent. On the other hand, C++ uses a compiler to compile and run the source code. It converts source code into machine level language; hence C++ is platform dependent.

**6. Why Java is platform independent?**

**Ans:**

Java is platform-independent because the same java program can run on any operating system. If you write a code in Java, then the program will be sent to the compiler for compilation. The compiler creates a . class file that is readable for JVM(Java Virtual Machine).

2) For every operating system separate JVM is available which is capable to read the . class file or byte code. An important point to be noted is that while JAVA is platform-independent language, the JVM is platform-dependent. Different JVM is designed for different OS and byte code is able to run on different OS

**7. What are wrapper classes in Java?**

**Ans:**

1) Java Object Oriented Programming. A Wrapper class is a class which contains the primitive data types (int, char, short, byte, etc). In other words, wrapper classes provide a way to use primitive data types (int, char, short, byte, etc) as objects. These wrapper classes come under java.

2) Wrapper classes are used to convert any data type into an object. The primitive data types are not objects; they do not belong to any class; they are defined in the language itself. Sometimes, it is required to convert data types into objects in Java language.

3) e.g. Integer, Byte, Long, Float, Double, Character, Boolean,Short

**8. Why pointers are not used in Java?**

**Ans**

1) 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.

2)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.

3) 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.

**9. List some features of Java?**

**Ans:**

Following are the features of Java:

1. Object Oriented

In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

1. Platform Independent

Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform-independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

1. Simple

Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

1. Secure

With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

1. Architecture-neutral

Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

1. Portable

Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. The compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

1. Robust

Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.

1. Multithreaded

With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

1. Interpreted

Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

1. High Performance

With the use of Just-In-Time compilers, Java enables high performance.

1. Distributed

Java is designed for the distributed environment of the internet.

1. Dynamic

Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects at run-time.

**10.Why is Java Architectural Neutral**

**Ans:**

1)Java's object-oriented model enables programmers to benefit from its large set of existing classes. An attractive feature of Java is that it is architecture neutral. This means that Java code needs to be compiled only once, after which it can run on many CPUs.

2) For architecture neutral the compiler will generate an architecture-neutral object file meaning that compiled Java code (bytecode) can run on many processors given the presence of a Java runtime. For portable it means there are are no implementation-dependent aspects of the specification.

**11**.**How Java enabled High Performance?**

**Ans:**

Java enabled High performance by introducing JIT- Just In Time compiler , JIT helps the compiler to compile the code On demand basis i.e which ever method is called only that method block will get compiled making compilation fast n time-efficient. This makes the java delivering high performance.

# **12. Why java is considered as dynamic?**

**Ans :**

The fact that Java loads class files at runtime is one thing that makes it dynamic, java is also considered as a Dynamic programming language cause of Bytecode[a class file . When you write a source code in one platform, then the same code can be executed in other platforms as long as it has the JDK Installed .

**13. What is java virtual machine and how it is considered in context of Java Platform independent features?**

**Ans :**

A Java virtual machine (JVM) is a virtual machine that enables a computer to run Java programs as well as programs written in other languages that are also compiled to Java bytecode. The JVM reference implementation is developed by the OpenJDK project as open source code and includes a JIT compiler called HotSpot.

Java is platform independent. Because the Java compiler converts the source code to bytecode, which is Intermidiate Language. Bytecode can be executed on any platform (OS) using JVM( Java Virtual Machine).

**14.List two Java IDE’s?**

**Ans:**

1. NetBeans: Once owned by Sun Microsystems, and now by Oracle, NetBeans is one of the best IDEs from the past decade. NetBeans is completely modularized, and all the IDE functions come in packages called modules.  It offers easy integration with version control software NetBeans is a cross platform IDE, supporting Windows, Mac and Linux.

2. Eclipse: Eclipse is one of the most popular IDEs out there, and not for just Java, but also for C++ and PHP. Eclipse is an open source tool and has a great community of developers. It also features a huge library of plugins, all made by users. Eclipse latest big release came last year with Eclipse Luna, which offered support for Java 8. Eclipse is available for Windows, Mac, Linux and OSX.

**15.Why Java is called “Platform”?**

**Ans:**

When the Java program runs in a particular machine it is sent to java compiler, which converts this code into intermediate code called bytecode. JVM recognizes the platform it is on and converts the bytecodes into native machine code. Hence java is called platform independent language.

**16. Is Java Pure-Object oriented Language?**

**Ans**

No, Java is not pure object-oriented programming language because it supports primitive data types like int, char, Boolean, long, short, float, double and everything is not an object in Java.

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

**Ans** :

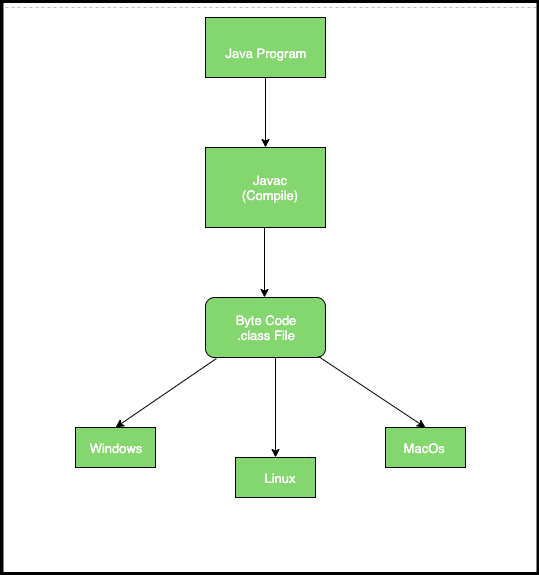
We have learned java version 1.8 or java 8. Some of the new features added to it are as follow: -

* Lambda Expressions − a new language feature allowing us to treat actions as objects
* Method References − enable us to define Lambda Expressions by referring to methods directly using their names
* *Optional* − special wrapper class used for expressing optionality
* Functional Interface – an interface with maximum one abstract method; implementation can be provided using a Lambda Expression
* Default methods − give us the ability to add full implementations in interfaces besides abstract methods
* Nashorn, JavaScript Engine − Java-based engine for executing and evaluating JavaScript code
* *Stream* API − a special iterator class that allows us to process collections of objects in a functional manner
* Date API − an improved, immutable JodaTime-inspired Date API.

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

**ANS**

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.**

**Ans:**

|  |  |
| --- | --- |
| path | classpath |
| path variable is used to set the path for all Java software tools like javac.exe, java.exe, javadoc.exe, and so on. | classpath variable is used to set the path for java classes. |

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

**Ans**

**main():** It is a default signature which is predefined in the JVM. It is called by JVM to execute a program line by line and end the execution after completion of this method. We can also overload the main() method.

**Qu21. DIfference between JDK and JRE**

key JDK JRE

DEFINITION JDK(Java Development Kit) is used JRE(Java Runtime Environment) is the develop

Java applications. JDK also of JVM(Java Virtual Machine) and it is specially

contains numerous development tools designed to execute Java programs.

like compilers, debuggers, etc.

FUNCTIONALITY It is mainly used for the execution of It is mainly used for creating an environment

code and its main functionality is for code execution.

development.

DEPENDENCY It is platform-dependent. It is also platform-dependent like JDK.

OF PLATFORM

TYPE OF TOOLS Since JDK is responsible for the On the other hand, JRE is not responsible for

development purpose, therefore it development purposes so it doesn’t contain such tools

contains tools which are required as the compiler, debugger, etc. Instead, it contains class

for development and debugging purpose. a libraries and supporting files required for the purpose

of execution of the program.

IMPLEM- JDK = JRE + other development tools. JRE = JVM + other class libraries.

ENTATION

**Qu22 What is JVM? what it does?**

JVM, i.e., Java Virtual Machine.

--> Mostly in other Programming Languages, compiler produce code for a particular system but Java compiler produce Bytecode for a Java Virtual Machine.

--> When we compile a Java program, then bytecode is generated. Bytecode is the source code that can be used to run on any platform.

--> Bytecode is an intermediary language between Java source and the host system.

--> It is the medium which compiles Java code to bytecode which gets interpreted on a different machine and hence it makes it Platform/Operating system independent.

The JVM performs following operation:

--> Loads the code

--> Verifies the code

--> Executes the code

--> Provides the run-time environment

--> Provides the memory area

--> Register set

--> Provides a garbage collection heap

--> Reports the fatal errors

--> Provides a class file format

**Q23 Why JVM is called as virtual machine?**

The JVM is called a virtual machine because the JVM definition defines an abstract machine. This includes registers, stack, etc, and the byte code that Java source is compiled to is practically machine code for this virtual machine. The JVM then interprets or compiles this byte code into native machine instructions.

The difference is essentially that the JVM is a virtualized processor and the other virtual machines are virtualized machines (including video card, network, and other external devices and hardware registers).

**Q24. What are the main components of JVM? Explain the. Or Explain JVM architecture.**



1) Classloader

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader.

There are three built-in classloaders in Java.

1.Bootstrap ClassLoader: This is the first classloader which is the super class of Extension classloader. It loads the rt.jar file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.

2.Extension ClassLoader: This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside $JAVA\_HOME/jre/lib/ext directory.

3.System/Application ClassLoader: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

2) Class(Method) Area

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap

It is the runtime data area in which objects are allocated.

4) Stack

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.Each thread has a private JVM stack, created at the same time as thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register

PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack

It contains all the native methods used in the application.

7) Execution Engine

It contains:

A virtual processor

Interpreter: Read bytecode stream then execute the instructions.

Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the

instruction set of a specific CPU.

8) Java Native Interface

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc.

Java uses JNI framework to send output to the Console or interact with OS libraries.

**Q25 What is difference between Java compiler and JIT**

The basic definition of compiler as "a program that translates the source code to a machine executable code". When compiling a java program, the static compiler that is run using the command javac converts the source code to byte code which are in the form of .class file. These code needs to be interpreted to convert it into machine executable code. Using a compiler, Java source code is converted to Java byte code (.class files). Once this is done, JVM loads the .class files at run time and converts them to a machine understandable code using an interpreter.

JIT compiler is a feature of JVM which when enabled makes the JVM analyze the method calls in byte code and compiles them to more native and efficient code. JIT optimizes the prioritized method calls at this point of time. Once these method calls are compiled, the JVM then executes this optimized code instead of interpreting it which is likely to increase the performance of the execution.

Though both compiler and an interpreter do the same job of converting a high level language to a machine executable code, there are few differences in the way they do it. A compiler is a program that converts the entire code into a equivalent machine code at once. That is a compiler scans the entire program first, translate it to a machine executable code which will then be executed by the process and the corresponding output is generated. Compiler generates errors if any at the end of execution of the program. On the other side, interpreter takes the single instruction of the code, translates it into an intermediate code and then into a machine code, executes it and takes another instruction. Interpreter generates errors as soon as it finds error in any of the sequence of instructions.

Though the time for analyzing the program and converting high level language to a machine executable code is high in compiler, when it comes to overall performance compiler is fast. This is because when an error occurs during the interpretation of the code, it has to reinterpret the previously interpreted code once again which is a tedious process.

JIT is the tool which can transform bytecode to the binary code. javac is the tool which can transform code to the Java bytecode. We can conclude

the relationship: JDK = JRE + javac + jconsole +jvisualvm + demo + document + other

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

**Ans:**

Yes. An Empty.java file is a perfectly valid source file.

Java file contain more than one java classes, provided at the most one of them is a public class.

**26. Is JRE different for different Platforms?**

**Ans:**

Whenever we try to run the code, JVM requires some library set and files for code execution and these files are presented in JRE.

JRE = JVM + set of libraries. ... JRE is also platform dependent. That means we have different JRE versions for different platforms.JRE does not contain tools such as compiler or debugger. The JREs are both platform and architecture-specific.

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

**Ans:**

Whenever we try to run the code, JVM requires some library set and files for code execution and these files are presented in JRE.

JRE = JVM + set of libraries. ... JRE is also platform dependent. That means we have different JRE versions for different platforms.JRE does not contain tools such as compiler or debugger. The JREs are both platform and architecture-specific.

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

**Ans:**

1. C++ uses only compiler, whereas Java uses compiler and interpreter both.

2. C++ supports both operator overloading & method overloading whereas Java only supports method overloading.

3.C++ supports manual object management with the help of new and delete keywords whereas Java has built-in automatic garbage collection.

4. C++ is a platform dependent language. The source code written in C++ needs to be compiled on every platform whereas Java is platform-independent. Once compiled into byte code, it can be executed on any platform.

5. C++ code is not portable. It must be compiled for each platform whereas Java, however, translates the code into byte code. This byte code is portable and can be executed on any platform.

6. Memory management in C++ is manual. We need to allocate/deallocate memory manually using the new/delete operators whereas Java the memory management is system-controlled.

7. In C++, methods and operators can be overloaded. This is static polymorphism whereas In Java, only method overloading is allowed.It does not allow operator overloading.

**29. Who invokes main () function?**

**Ans:**

The main method in the Java language is similar to the main function in C and C++. When the Java interpreter executes an application (by being invoked upon the application's controlling class), it starts by calling the class's main method.

public static void main (String [] args)

{

boolean t=true;

System.out.println("Before return");

if(t) return;

System.out.println("not execute");

}

In the above code when the return is used then it should return to the function which calls the main function. Who exactly calls the main function in java.

**30. What is .class file known as?**

**Ans:**

A **Java class file** is a [file](https://en.wikipedia.org/wiki/Computer_file) (with the .class [filename extension](https://en.wikipedia.org/wiki/Filename_extension)) containing [Java bytecode](https://en.wikipedia.org/wiki/Java_bytecode) that can be executed on the [Java Virtual Machine (JVM)](https://en.wikipedia.org/wiki/Java_Virtual_Machine). A Java class file is usually produced by a [Java compiler](https://en.wikipedia.org/wiki/Java_compiler) from [Java programming language](https://en.wikipedia.org/wiki/Java_(programming_language)) [source files](https://en.wikipedia.org/wiki/Source_file) (.java files) containing Java [classes](https://en.wikipedia.org/wiki/Class_(programming)) (alternatively, other [JVM languages](https://en.wikipedia.org/wiki/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 . ?**

**Ans:**

According to Java standards and common practices, we should declare every class in its own source file. And even if we declare multiple classes in a single source file (.java), still each class will have its own class file after compilation. But the fact is that we can declare more than one class in a single source file with these constraints,

* Each source file should contain only one public class and the name of that public class should be similar to the name of the source file.
* If you are declaring a main method in your source file then main should lie in that public class.

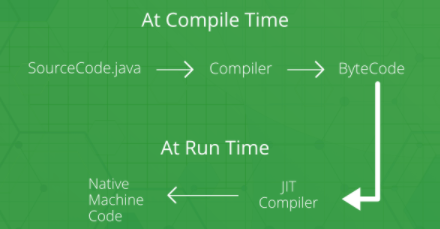
**Rule of public class:**

In the case of a **public class**, we can't use a different **file name**. The filename must have the same **name** as the **public class name** in that **file**, which is the way to tell the JVM that this is an entry point.

**32.What is JIT compiler?**

**Ans:**

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.



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

**Ans:**

**J**ava **V**irtual**M**achine is a program/software which takes Java bytecode (.class files)

and converts the byte code into machine understandable code.

JVM contains a module known as a class loader. A class loader in JVM loads, links and, initializes a program. It−

* Loads the class into the memory.
* Verifies the byte code instructions.
* Allocates memory for the program.

The memory in the JVM is divided into five different parts namely− **Method area−** The method area stores the class code − code of the variables and methods.

**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**.

**Ans:**

A local variable is a variable declared inside a method. A localvariable is only accessible inside the method that declared it. A parameter is a variable that is passed to a method when the method is called.

There are three types of variables in [Java](https://www.javatpoint.com/java-tutorial):

* local variable
* instance variable
* static variable

#### 1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

**Rules for local variable:**

* Local variables cannot use any of the access level since their scope is only inside the method.
* Final is the Only Non Access Modifier that can be applied to a local variable.
* Local variables are not assigned a default value, hence they need to be initialized.

**35. What are the various access specifiers in Java?**

**Ans:**

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **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.
3. **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.
4. **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

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

**Ans:**

A local variable is a variable declared inside a method. A localvariable is only accessible inside the method that declared it. A parameter is a variable that is passed to a method when the method is called.

There are three types of variables in [Java](https://www.javatpoint.com/java-tutorial):

* local variable
* instance variable
* static variable

#### 1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

**Rules for local variable:**

* Local variables cannot use any of the access level since their scope is only inside the method.
* Final is the Only Non Access Modifier that can be applied to a local variable.
* Local variables are not assigned a default value, hence they need to be initialized.

**37. What is native code?**

**Ans:**

Although it is rare, occasionally you may want to call a subroutine that is written in a language other than Java. Typically, such a subroutine exists as executable code for the CPU and environment in which you are working—that is, native code.

**38. Why there is no sizeof operator in java ?**

**Ans:**

Because the size of primitive types is explicitly mandated by the Java language.  Since allocation is done by the new operator depending on its argument thereisno need to specify the amount of memory needed.

In java, you don't work directly with memory, so sizeof is usually not needed

**39. What kinds of programs u can develop using Java ?**

**Ans:**

When considering what you can do with Java, many developers think of building:

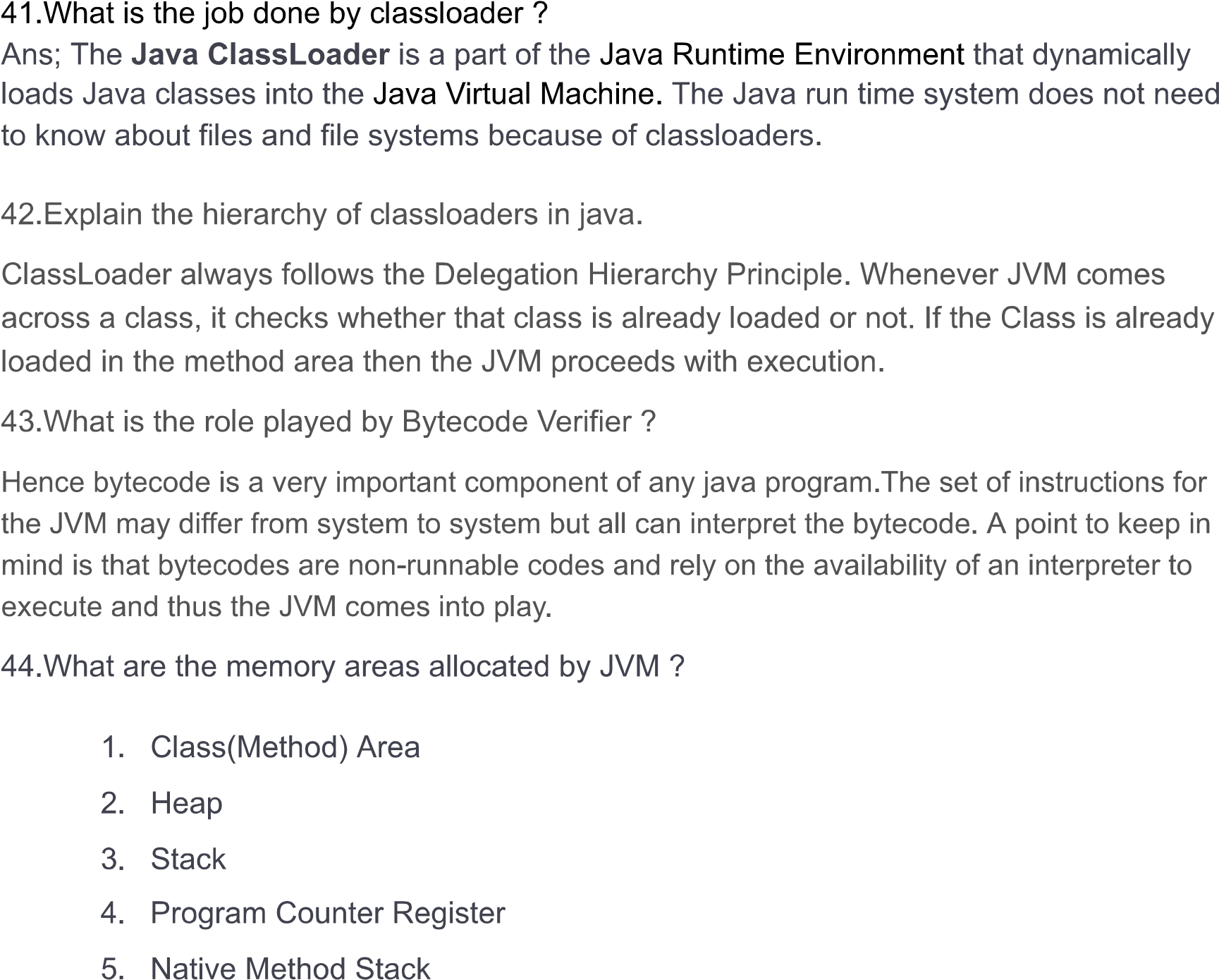
* Application servers.
* Web applications.
* Unit tests.
* Mobile applications.
* Desktop applications.
* Enterprise applications.

**40.You have reference type as a member of class. What is the default value it gets?**

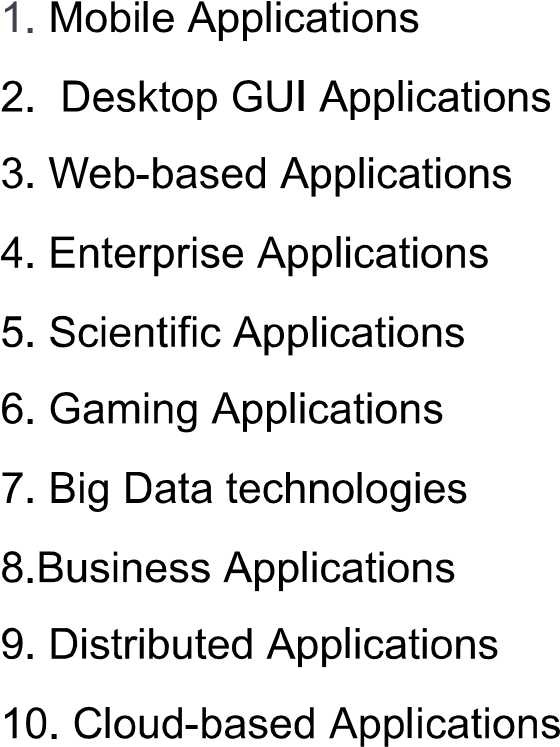
**Ans:**

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.

The default value of a reference type is null. It means that if a reference type is a static class member or an instance field and not assigned an initial value explicitly, it will be initialized automatically and assigned the value of null. Please note that, in the case of an array, this applies to the array itself and to its reference type components.



**Ans:**



**46.When parseInt() method can be used?**

**Ans:**

Convert a string to an integer with the parseInt method of the Java Integer class. The parseInt method is to convert the String to an int and throws a NumberFormatException if the string cannot be converted to an int type.

**47.What is finalized() method ?**

**Ans:**

The java.lang.Object.finalize() is called by the garbage collector on an object when garbage collection determines that there are no more references to the object. A subclass overrides the finalize method to dispose of system resources or to perform other cleanup. Declaration: protected void finalize() Return Value: This method does not return a value. Exception: Throwable − the Exception raised by this method

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

**Ans:**

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. 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 reinterpret 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.

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

**Ans:**

The default value of a reference type is null. It means that if a reference type is a static class member or an instance field and not assigned an initial value explicitly, it will be initialized automatically and assigned the value of null. Please note that, in the case of an array, this applies to the array itself and to its reference type components.

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

**Ans:**

The switch statement is a multi-way branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression. Basically, 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

Qu21. DIfference between JDK and JRE

key JDK JRE

DEFINITION JDK(Java Development Kit) is used JRE(Java Runtime Environment) is the implementation

develop Java applications. JDK also of JVM(Java Virtual Machine) and it is specially

contains numerous development tools designed to execute Java programs.

like compilers, debuggers, etc.

FUNCTIONALITY It is mainly used for the execution of It is mainly used for creating an environment

code and its main functionality is for code execution.

development.

DEPENDENCY It is platform-dependent. It is also platform-dependent like JDK.

OF PLATFORM

TYPE OF TOOLS Since JDK is responsible for the On the other hand, JRE is not responsible for

development purpose, therefore it development purposes so it doesn’t contain such tools

contains tools which are required as the compiler, debugger, etc. Instead, it contains class

for development and debugging purpose. a libraries and supporting files required for the purpose

of execution of the program.

IMPLEMENTATION JDK = JRE + other development tools. JRE = JVM + other class libraries.

Qu22 What is JVM? what it does?

JVM, i.e., Java Virtual Machine.

--> Mostly in other Programming Languages, compiler produce code for a particular system but Java compiler produce Bytecode for a Java Virtual Machine.

--> When we compile a Java program, then bytecode is generated. Bytecode is the source code that can be used to run on any platform.

--> Bytecode is an intermediary language between Java source and the host system.

--> It is the medium which compiles Java code to bytecode which gets interpreted on a different machine and hence it makes it Platform/Operating system independent.

The JVM performs following operation:

--> Loads the code

--> Verifies the code

--> Executes the code

--> Provides the run-time environment

--> Provides the memory area

--> Register set

--> Provides a garbage collection heap

--> Reports the fatal errors

--> Provides a class file format

Q23 Why JVM is called as virtual machine

The JVM is called a virtual machine because the JVM definition defines an abstract machine. This includes registers, stack, etc, and the byte code that Java source is compiled to is practically machine code for this virtual machine. The JVM then interprets or compiles this byte code into native machine instructions.

The difference is essentially that the JVM is a virtualized processor and the other virtual machines are virtualized machines (including video card, network, and other external devices and hardware registers).

Q24. What are the main components of JVM? Explain the. Or Explain JVM architecture.



1) Classloader

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader.

There are three built-in classloaders in Java.

1.Bootstrap ClassLoader: This is the first classloader which is the super class of Extension classloader. It loads the rt.jar file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.

2.Extension ClassLoader: This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside $JAVA\_HOME/jre/lib/ext directory.

3.System/Application ClassLoader: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

2) Class(Method) Area

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap

It is the runtime data area in which objects are allocated.

4) Stack

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.Each thread has a private JVM stack, created at the same time as thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register

PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack

It contains all the native methods used in the application.

7) Execution Engine

It contains:

A virtual processor

Interpreter: Read bytecode stream then execute the instructions.

Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the

instruction set of a specific CPU.

8) Java Native Interface

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc.

Java uses JNI framework to send output to the Console or interact with OS libraries.

Q25 What is difference between Java compiler and JIT

The basic definition of compiler as "a program that translates the source code to a machine executable code". When compiling a java program, the static compiler that is run using the command javac converts the source code to byte code which are in the form of .class file. These code needs to be interpreted to convert it into machine executable code. Using a compiler, Java source code is converted to Java byte code (.class files). Once this is done, JVM loads the .class files at run time and converts them to a machine understandable code using an interpreter.

JIT compiler is a feature of JVM which when enabled makes the JVM analyze the method calls in byte code and compiles them to more native and efficient code. JIT optimizes the prioritized method calls at this point of time. Once these method calls are compiled, the JVM then executes this optimized code instead of interpreting it which is likely to increase the performance of the execution.

Though both compiler and an interpreter do the same job of converting a high level language to a machine executable code, there are few differences in the way they do it. A compiler is a program that converts the entire code into a equivalent machine code at once. That is a compiler scans the entire program first, translate it to a machine executable code which will then be executed by the process and the corresponding output is generated. Compiler generates errors if any at the end of execution of the program. On the other side, interpreter takes the single instruction of the code, translates it into an intermediate code and then into a machine code, executes it and takes another instruction. Interpreter generates errors as soon as it finds error in any of the sequence of instructions.

Though the time for analyzing the program and converting high level language to a machine executable code is high in compiler, when it comes to overall performance compiler is fast. This is because when an error occurs during the interpretation of the code, it has to reinterpret the previously interpreted code once again which is a tedious process.

JIT is the tool which can transform bytecode to the binary code. javac is the tool which can transform code to the Java bytecode. We can conclude

the relationship: JDK = JRE + javac + jconsole +jvisualvm + demo + document + other

djncjkfdk