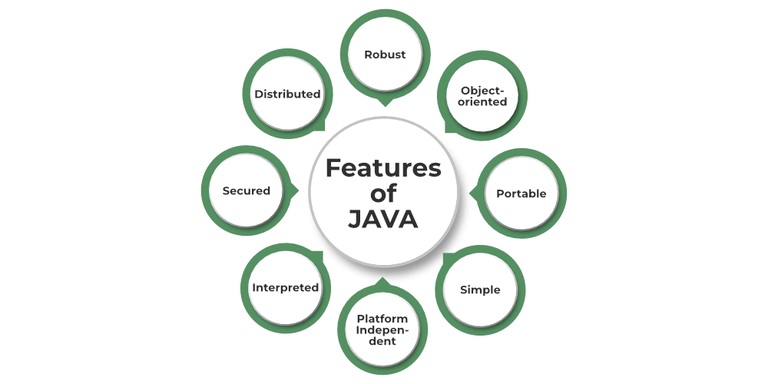
**1. Is Java Platform Independent if then how?**

Yes, Java is Platform Independent. The javac compiler produces bytecode (.class file), which is not tied to specific software or hardware. Although the Java Virtual Machine (JVM) is platform-dependent, the bytecode can run on any system with a JVM, making Java platform-independent.

**2. What are the top Java Features?**

Java is one the most famous and most used language in the real world, there are many features in Java that makes it better than any other language some of them are mentioned below:



**Simple syntax**

**Platform independence**

**Interpreted and compiler-based**

**Robust (garbage collection, exception handling)**

**Object-oriented**

**Secure (code-sharing without revealing)**

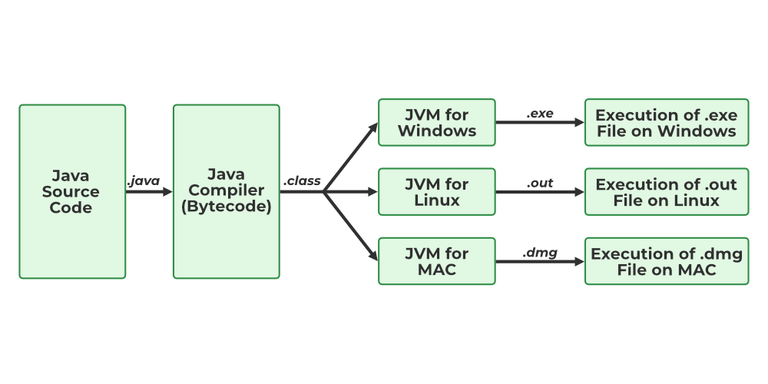
**High performance**

**Dynamic (supports dynamic loading)**

**Distributed (access files across machines)**

**Multithreaded**

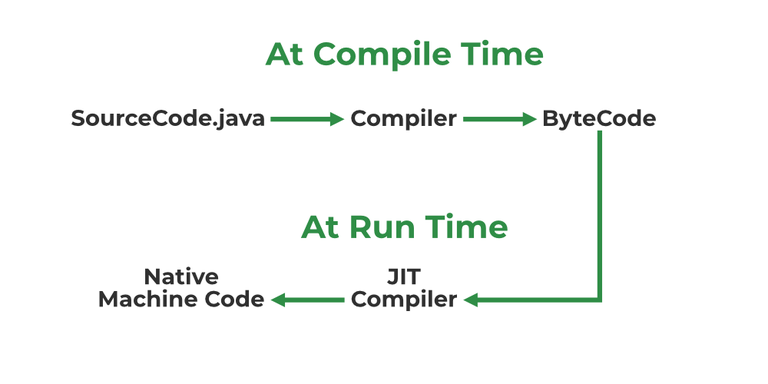
**3. What is JVM?**



JVM stands for Java Virtual Machine it is a Java interpreter. It is responsible for loading, verifying, and executing the bytecode created in Java.

Although it is platform dependent which means the software of JVM is different for different Operating Systems it plays a vital role in making Java platform Independent

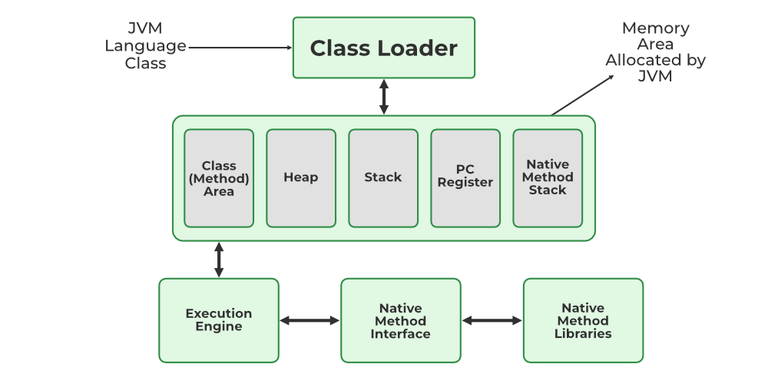
**4. What is JIT?**



JIT stands for (Just-in-Time) compiler is a part of JRE(Java Runtime Environment), it is used for better performance of the Java applications during run-time. The use of JIT is mentioned in step by step process mentioned below:

1. Source code is compiled with **javac** compiler to form bytecode
2. Bytecode is further passed on to JVM
3. JIT is a part of JVM, JIT is responsible for compiling bytecode into native machine code at run time.
4. 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.
5. As JVM calls the compiled code that increases the performance and speed of the execution.

**5. What are Memory storages available with JVM?**



1. Class(Method) Area: stores class-level data of every class such as the runtime constant pool, field, and method data, and the code for methods.
2. Heap: Objects are created or objects are stored. It is used to allocate memory to objects during run time.
3. Stack: stores data and partial results which will be needed while returning value for method and performing dynamic linking
4. Program Counter Register: stores the address of the Java virtual machine instruction currently being executed.
5. Native Method Stack: stores all the native methods used in the application.

**JAVA FEATURES**

* **Simple**: Java is quite simple to understand and the syntax
* **Platform Independent:** Java is platform independent means we can run the same program in any software and hardware and will get the same result.
* **Interpreted**: Java is interpreted as well as a compiler-based language.
* **Robust**: features like Garbage collection, exception handling, etc that make the language robust.
* **Object-Oriented**: Java is an object-oriented language that supports the concepts of class,  objects, four pillars of OOPS, etc.
* **Secured**: As we can directly share an application with the user without sharing the actual program makes Java a secure language.
* **High Performance:**  faster than other traditional interpreted programming languages.
* **Dynamic**: supports dynamic loading of classes and interfaces.Distributed: feature of Java makes us able to access files by calling the methods from any machine connected.
* **Multithreaded**: deal with multiple tasks at once by defining multiple threads
* **Architecture Neutral**: it is not dependent on the architecture.

### 6. What is a classloader?

Classloader is the part of JRE(Java Runtime Environment), during the execution of the bytecode or created .class file classloader is responsible for dynamically loading the java classes and interfaces to JVM(Java Virtual Machine). Because of classloaders Java run time system does not need to know about files and file systems.

### 

### 7. Difference between JVM, JRE, and JDK.

**JVM**: JVM also known as Java Virtual Machine is a part of JRE. JVM is a type of interpreter responsible for converting bytecode into machine-readable code. JVM itself is platform dependent but it interprets the bytecode which is the platform-independent reason why Java is platform-independent.

**JRE**: JRE stands for Java Runtime Environment, it is an installation package that provides an environment to run the Java program or application on any machine.

**JDK**: JDK stands for Java Development Kit which provides the environment to develop and execute Java programs. JDK is a package that includes two things Development Tools to provide an environment to develop your Java programs and, JRE to execute Java programs or applications.

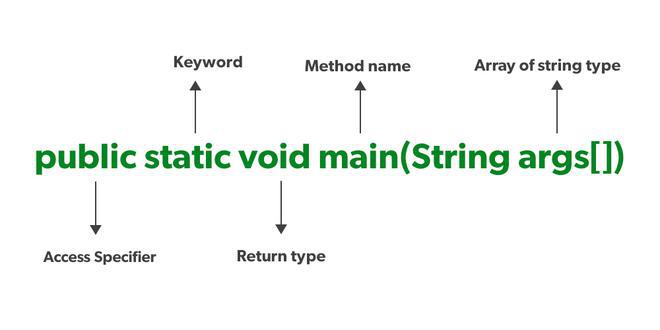
### https://media.geeksforgeeks.org/wp-content/uploads/20210218150010/JDK.png

### 8. What are the differences between Java and C++?

| **Basis** | **C++** | **Java** |
| --- | --- | --- |
| **Platform** | C++ is Platform Dependent | Java is Platform Independent |
| **Application** | C++ is mainly used for System Programming | Java is Mainly used for Application Programming |
| **Hardware** | C++ is nearer to hardware | Java is not so interactive with hardware |
| **Not Supporting** | Functionality supported in Java but not in C++ are:   * thread support * documentation comment * unsigned right shift(>>>) | Functionality supported in C++ but not in Java are:   * goto * Pointers * Call by reference * Structures and Unions * Multiple Inheritance * Virtual Functions |
| **OOPS** | C++ is an object-oriented language. It is not a single root hierarchy . | Java is also an object-oriented language. It is a single root hierarchy as everything gets derived from a single class (java.lang.Object). |
| **Inheritance Tree** | C++ always creates a new inheritance tree. | Java uses a Single inheritance tree as classes in Java are the child of object classes in Java. |

### 

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



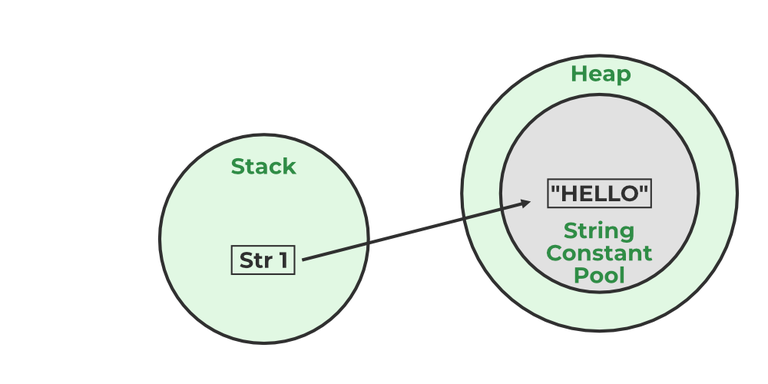
Unlike any other programming language like C, C++, etc. In Java, we declared the main function as a public static void main (String args[]). The meanings of the terms are mentioned below:

1. **public**: the public is the access modifier responsible for mentioning who can access the element or the method and what is the limit.  It is responsible for making the main function globally available. It is made public so that JVM can invoke it from outside the class as it is not present in the current class.
2. **static**: static is a keyword used so that we can use the element without initiating the class so to avoid the unnecessary allocation of the memory.
3. **void**: void is a keyword and is used to specify that a method doesn’t return anything. As the main function doesn’t return anything we use void.
4. **main**: main represents that the function declared is the main function. It helps JVM to identify that the declared function is the main function.
5. **String args[]**: It stores Java command-line arguments and is an array of type java.lang.String class.

### 

### 10. What is Java String Pool?

A Java String Pool is a place in heap memory where all the strings defined in the program are stored. A separate place in a stack is there where the variable storing the string is stored. Whenever we create a new string object, JVM checks for the presence of the object in the String pool, If String is available in the pool, the same object reference is shared with the variable, else a new object is created.



**Example:**

String str1="Hello";

// "Hello" will be stored in String Pool

// str1 will be stored in stack memory

**11. What will happen if we declare don’t declare the main as static?**

We can declare the main method without using static and without getting any errors. But, the main method will not be treated as the entry point to the application or the program.

**12. What are Packages in Java?**

Packages in Java can be defined as the grouping of related types of classes, interfaces, etc providing access to protection and namespace management.

**13. Why Packages are used?**

Packages are used in Java in order to prevent naming conflicts, control access, and make searching/locating and usage of classes, interfaces, etc easier.

**14. What are the advantages of Packages in Java?**

* Packages avoid name clashes.
* The Package provides easier access control.
* We can also have the hidden classes that are not visible outside and are used by the package.
* It is easier to locate the related classes.

**15. How many types of packages are there in Java?**

There are two types of packages in Java

* User-defined packages
* Build In packages

**16. Explain different data types in Java.**

There are 2 types of data types in Java as mentioned below:

1. **Primitive Data Type**
2. **Non-Primitive Data Type or Object Data type**

**Primitive Data Type:** Primitive data are single values with no special capabilities. There are 8 primitive data types:

* **boolean**: stores value true or false
* **byte**: stores an 8-bit signed two’s complement integer
* **char**: stores a single 16-bit Unicode character
* **short**: stores a 16-bit signed two’s complement integer
* **int**: stores a 32-bit signed two’s complement integer
* **long**: stores a 64-bit two’s complement integer
* **float**: stores a single-precision 32-bit IEEE 754 floating-point
* **double**: stores a double-precision 64-bit IEEE 754 floating-point

**Non-Primitive Data Type:** Reference Data types will contain a memory address of the variable’s values because it is not able to directly store the values in the memory. Types of Non-Primitive are mentioned below:

* Strings
* Array
* Class
* Object
* Interface

**17. When a byte datatype is used?**

A byte is an 8-bit signed two-complement integer. The minimum value supported by bytes is -128 and 127 is the maximum value. It is used in conditions where we need to save memory and the limit of numbers needed is between -128 to 127.

**18. Can we declare Pointer in Java?**

No, Java doesn’t provide the support of Pointer. As Java needed to be more secure because which feature of the pointer is not provided in Java.

**19. What is the default value of byte datatype in Java?**

The default value of the byte datatype in Java is 0.

**20. What is the default value of float and double datatype in Java?**

The default value of the float is 0.0f and of double is 0.0d in Java.

**21. What is the Wrapper class in Java?**

Wrapper, in general, is referred to a larger entity that encapsulates a smaller entity. Here in Java, the wrapper class is an object class that encapsulates the primitive data types.

The primitive data types are the ones from which further data types could be created. For example, integers can further lead to the construction of long, byte, short, etc. On the other hand, the string cannot, hence it is not primitive.

Getting back to the wrapper class, Java contains 8 wrapper classes. They are Boolean, Byte, Short, Integer, Character, Long, Float, and Double. Further, custom wrapper classes can also be created in Java which is similar to the concept of Structure in the C programming language. We create our own wrapper class with the required data types.

**22. Why do we need wrapper classes?**

The wrapper class is an object class that encapsulates the primitive data types, and we need them for the following reasons:

* Wrapper classes are final and immutable
* Provides methods like valueOf(), parseInt(), etc.
* It provides the feature of autoboxing and unboxing.

**23. Differentiate between instance and local variables.**

| **Instance Variable** | **Local Variable** |
| --- | --- |
| Declared outside the method, directly invoked by the method. | Declared within the method. |
| Has a default value. | No default value |
| It can be used throughout the class. | The scope is limited to the method. |

**24. What are the default values and size assigned to variables and instances in Java?**

In Java When we haven’t initialized the instance variables then the compiler initializes them with default values. The default values for instances and variables depend on their data types. Some common types of default data types are:

The default value for numeric types (byte, short, int, long, float, and double) is 0.

The default value for the boolean type is false.

The default value for object types (classes, interfaces, and arrays) is null.

The null character, “u0000, ” is the default value for the char type.

| **Data Type** | **Size** |
| --- | --- |
| Byte | 1 byte |
| Short | 2 bytes |
| Int | 4 bytes |
| Long | 8 bytes |
| Float | 4 bytes |
| Double | 8 bytes |
| Boolean | 1 bit |
| Char | 1 byte |

| **Data Type** | **Default Values** |
| --- | --- |
| Byte | 0 |
| Short | 0 |
| Int | 0 |
| Long | 0 |
| Float | 0.0 |
| Double | 0.0 |
| Boolean | false |
| Char | \u0000′ or null |

**25. What is a Class Variable?**

In Java, a class variable (also known as a static variable) is a variable that is declared within a class but outside of any method, constructor, or block. Class variables are declared with the static keyword, and they are shared by all instances (objects) of the class as well as by the class itself. No matter how many objects are derived from a class, each class variable would only exist once.

class GFG {

public static int ctr = 0;

public GFG() { ctr++; }

public static void main(String[] args)

{

GFG obj1 = new GFG();

GFG obj2 = new GFG();

GFG obj3 = new GFG();

System.out.println("Number of objects created are "

+ GFG.ctr);

}

}

**Output**

Number of objects created are 3

**26. What is the default value stored in Local Variables?**

There is no default value stored with local variables. Also, primitive variables and objects don’t have any default values.

**27. Explain the difference between instance variable and a class variable.**

Instance Variable: A class variable without a static modifier known as an instance variable is typically shared by all instances of the class. These variables can have distinct values among several objects. The contents of an instance variable are completely independent of one object instance from another because they are related to a specific object instance of the class.

import java.io.\*;

class GFG {

private String name;

public void setName(String name) { this.name = name; }

public String getName() { return name; }

public static void main(String[] args)

{

GFG obj = new GFG();

obj.setName("John");

System.out.println("Name " + obj.getName());

}

}

Output

Name John

Class Variable: Class Variable variable can be declared anywhere at the class level using the keyword static. These variables can only have one value when applied to various objects. These variables can be shared by all class members since they are not connected to any specific object of the class.

import java.io.\*;

class GFG {

// class variable

private static final double PI = 3.14159;

private double radius;

public GFG(double radius) { this.radius = radius; }

public double getArea() { return PI \* radius \* radius; }

public static void main(String[] args)

{

GFG obj = new GFG(5.0);

System.out.println("Area of circle: "

+ obj.getArea());

}

}

Output

Area of circle: 78.53975

**28. What is a static variable?**

The static keyword is used to share the same variable or method of a given class. Static variables are the variables that once declared then a single copy of the variable is created and shared among all objects at the class level.

**29. What is the difference between System.out, System.err, and System.in?**

**System.out** – It is a PrintStream that is used for writing characters or can be said it can output the data we want to write on the Command Line Interface console/terminal.

**System.out.println("This is how we throw error with System.out");**

**System.err** – It is used to display error messages.

**System.err.println("This is how we throw error with System.err");**

Although, System.err have many similarities both of them have quite a lot of difference also, let us check them.

| **System.out** | **System.err** |
| --- | --- |
| It will print to the standard out of the system. | It will print to the standard error. |
| It is mostly used to display results on the console. | It is mostly used to output error texts. |
| It gives output on the console with the default(black) color. | It also gives output on the console but most of the IDEs give it a red color to differentiate. |

**System.in** – It is an InputStream used to read input from the terminal Window. We can’t use the System.in directly so we use Scanner class for taking input with the system.in.

**Scanner sc = new Scanner(System.in);**

**30. What do you understand by an IO stream?**

**Java IO stream**

Java brings various Streams with its I/O package that helps the user to perform all the input-output operations. These streams support all types of objects, data types, characters, files, etc to fully execute the I/O operations.

**31. What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?**

The key difference between them is that byte stream data is read and written by input/output stream classes. Characters are handled by the Reader and Writer classes. In contrast to Reader/Writer classes, which accept character arrays as parameters, input/output stream class methods accept byte arrays. In comparison to input/output streams, the Reader/Writer classes are more efficient, handle all Unicode characters, and are useful for internalization. Use Reader/Writer classes instead of binary data, such as pictures, unless you do so.

**32. What are the super most classes for all the streams?**

All the stream classes can be divided into two types of classes that are ByteStream classes and CharacterStream Classes. The ByteStream classes are further divided into InputStream classes and OutputStream classes. CharacterStream classes are also divided into Reader classes and Writer classes. The SuperMost classes for all the InputStream classes is java.io.InputStream and for all the output stream classes is java.io.OutPutStream. Similarly, for all the reader classes, the super-most class is java.io.Reader, and for all the writer classes, it is java.io.Writer.

**33. What are the FileInputStream and FileOutputStream?**

To read and write data, Java offers I/O Streams. A Stream represents an input source or an output destination, which could be a file, an i/o device, another program, etc. FileInputStream in Java is used to read data from a file as a stream of bytes. It is mostly used for reading binary data such as images, audio files, or serialized objects.

File file = new File("path\_of\_the\_file");

FileInputStream inputStream = new FileInputStream(file);

In Java, the FileOutputStream function is used to write data byte by byte into a given file or file descriptor. Usually, raw byte data, such as pictures, is written into a file using FileOutputStream.

File file = new File("path\_of\_the\_file");

FileOutputStream outputStream = new FileOutputStream(file);

**34. What is the purpose of using BufferedInputStream and BufferedOutputStream classes?**

When we are working with the files or stream then to increase the Input/Output performance of the program we need to use the BufferedInputStream and BufferedOutputStream classes. These both classes provide the capability of buffering which means that the data will be stored in a buffer before writing to a file or reading it from a stream. It also reduces the number of times our OS needs to interact with the network or the disk. Buffering allows programs to write a big amount of data instead of writing it in small chunks. This also reduces the overhead of accessing the network or the disk.

BufferedInputStream(InputStream inp);

// used to create the bufferinput stream and save the arguments.

BufferedOutputStream(OutputStream output);

// used to create a new buffer with the default size.