Introduction to Java

Java is one of the most popular and widely used programming language and platform. A platform is an environment that helps to develop and run programs written in any programming language.

Java is fast, reliable and secure. From desktop to web applications, scientific supercomputers to gaming consoles, cell phones to the Internet, Java is used in every nook and corner.

Java is easy to learn and its syntax is simple and easy to understand. It is based on C++ (so easier for programmers who know C++). Java has removed many confusing and rarely-used features e.g. explicit pointers, operator overloading etc. Java also takes care of memory management and for that, it provides an automatic garbage collector. This collects the unused objects automatically.

JAVA was developed by James Gosling at **Sun Microsystems**_Inc in the year **1995**, later acquired by Oracle Corporation. It is a simple programming language. Java makes writing, compiling, and debugging programming easy. It helps to create reusable code and modular programs. Java is a class-based, object-oriented programming language and is designed to have as few implementation dependencies as possible. A general-purpose programming language made for developers to *write once run anywhere* that is compiled Java code can run on all platforms that support Java. Java applications are compiled to byte code that can run on any Java Virtual Machine. The syntax of Java is similar to c/c++.

History: Java's history is very interesting. It is a programming language created in 1991. James Gosling, Mike Sheridan, and Patrick Naughton, a team of Sun engineers known as the **Green team** initiated the Java language in 1991. **Sun Microsystems** released its first public implementation in 1996 as **Java 1.0**. It provides no-cost -run-times on popular platforms. Java1.0 compiler was re-written in Java by Arthur Van Hoff to strictly comply with its specifications. With the arrival of Java 2, new versions had multiple configurations built for different types of platforms.

In 1997, Sun Microsystems approached the ISO standards body and later formalized Java, but it soon withdrew from the process. At one time, Sun made most of its Java implementations available without charge, despite their proprietary software status. Sun generated revenue from Java through the selling of licenses for specialized products such as the Java Enterprise System.

On November 13, 2006, Sun released much of its Java virtual machine as free, open-source software. On May 8, 2007, Sun finished the process, making all of its JVM's core code available under open-source distribution terms.

The principles for creating java were simple, robust, secured, high performance, portable, multi-threaded, interpreted, dynamic, etc. In 1995 Java was developed by **James Gosling**, who is known as the Father of Java. Currently, Java is used in mobile devices, internet programming, games, e-business, etc.

Java programming language is named JAVA. Why?

After the name OAK, the team decided to give a new name to it and the suggested words were Silk, Jolt, revolutionary, DNA, dynamic, etc. These all names were easy to spell and fun to say, but they all wanted the name to reflect the essence of technology. In accordance with James Gosling, **Java** the among the top names along with **Silk**, and since java was a unique name so most of them preferred it.

Java is the name of an **island** in Indonesia where the first coffee(named java coffee) was produced. And this name was chosen by James Gosling while having coffee near his office. Note that Java is just a name, not an acronym.

Java Terminology

Before learning Java, one must be familiar with these common terms of Java.

1. **Java Virtual Machine(JVM):** This is generally referred to as JVM. There are three execution phases of a program. They are written, compile and run the program.

- Writing a program is done by a java programmer like you and me.
- The compilation is done by the **JAVAC** compiler which is a primary Java compiler included in the Java development kit (JDK). It takes the Java program as input and generates bytecode as output.
- In the Running phase of a program, **JVM** executes the bytecode generated by the compiler.

Now, we understood that the function of Java Virtual Machine is to execute the bytecode produced by the compiler. Every Operating System has a different JVM but the output they produce after the execution of bytecode is the same across all the operating systems. This is why Java is known as a **platform-independent language.**

2. **Bytecode in** the **Development process:** As discussed, the Javac compiler of JDK compiles the java source code into bytecode so that it can be executed by JVM. It is saved as **.class** file by the compiler. To view the bytecode, a disassembler like javap can be used.

3. **Java Development Kit(JDK):** While we were using the term JDK when we learn about bytecode and JVM. So, as the name suggests, it is a complete Java development kit that includes everything including compiler, Java Runtime Environment (JRE), java debuggers, java docs, etc. For the

program to execute in java, we need to install JDK on our computer in order to create, compile and run the java program.

4. Java Runtime Environment (JRE): JDK includes JRE. JRE installation on our computers allows the java program to run, however, we cannot compile it. JRE includes a browser, JVM, applet supports, and plugins. For running the java program, a computer needs JRE.

5. Garbage Collector: In Java, programmers can't delete the objects. To delete or recollect that memory JVM has a program called Garbage Collector. Garbage Collectors can recollect the objects that are not referenced. So Java makes the life of a programmer easy by handling memory management. However, programmers should be careful about their code whether they are using objects that have been used for a long time. Because Garbage cannot recover the memory of objects being referenced.

6. ClassPath: The classpath is the file path where the java runtime and Java compiler look for **.class** files to load. By default, JDK provides many libraries. If you want to include external libraries they should be added to the classpath.

Primary/Main Features of Java

1. Platform Independent: Compiler converts source code to bytecode and then the JVM executes the bytecode generated by the compiler. This bytecode can run on any platform be it Windows, Linux, or macOS which means if we compile a program on Windows, then we can run it on Linux and vice versa. Each operating system has a different JVM, but the output produced by all the OS is the same after the execution of bytecode. That is why we call java a platform-independent language.

2. Object-Oriented Programming Language: Organizing the program in the terms of collection of objects is a way of object-oriented programming, each of which represents an instance of the class.

The four main concepts of Object-Oriented programming are:

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

3. Simple: Java is one of the simple languages as it does not have complex features like pointers, operator overloading, multiple inheritances, and Explicit memory allocation.

4. Robust: Java language is robust which means reliable. It is developed in such a way that it puts a lot of effort into checking errors as early as possible, that is why the java compiler is able to detect even those errors that are not easy to detect by another programming language. The main features of java that make it robust are garbage collection, Exception Handling, and memory allocation.

5. Secure: In java, we don't have pointers, so we cannot access out-of-bound arrays i.e it shows **ArrayIndexOutOfBound Exception** if we try to do so. That's why several security flaws like stack corruption or buffer overflow are impossible to exploit in Java. Also java programs run in an environment that is independent of the os(operating system) environment which makes java programs more secure .

6. Distributed: We can create distributed applications using the java programming language. Remote Method Invocation and Enterprise Java Beans are used for creating distributed applications in java. The java programs can be easily distributed on one or more systems that are connected to each other through an internet connection.

7. Multithreading: Java supports multithreading. It is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of the CPU.

8. Portable: As we know, java code written on one machine can be run on another machine. The platform-independent feature of java in which its platform-independent bytecode can be taken to any platform for execution makes java portable.

9. High Performance: Java architecture is defined in such a way that it reduces overhead during the runtime and at some time java uses Just In Time (JIT) compiler where the compiler compiles code on-demand basics where it only compiles those methods that are called making applications to execute faster.

10. Dynamic flexibility: Java being completely object-oriented gives us the flexibility to add classes, new methods to existing classes and even create new classes through sub-classes. Java even supports functions written in other languages such as C, C++ which are referred to as native methods.

11. Sandbox Execution: Java programs run in a separate space that allows user to execute their applications without affecting the underlying system with help of a bytecode verifier. Bytecode verifier also provides additional security as its role is to check the code for any violation of access.

12. Write Once Run Anywhere: As discussed above java application generates a '.class' file which corresponds to our applications(program) but contains code in binary format. It provides ease t architecture-neutral ease as bytecode is not dependent on any machine architecture. It is the primary reason java is used in the enterprising IT industry globally worldwide.

13. Power of compilation and interpretation: Most languages are designed with purpose either they are compiled language or they are interpreted language. But java integrates arising enormous power as Java compiler compiles the source code to bytecode and JVM executes this bytecode to machine OS-dependent executable code.

Other things:-

1. ***import java.io.*:*** This means all the classes of io package can be imported. Java io package provides a set of input and output streams for reading and writing data to files or other input or output sources.
2. ***class:*** The class contains the data and methods to be used in the program. Methods define the behavior of the class. Class **GFG** has only one method Main in JAVA.
3. ***static void Main(): static*** keyword tells us that this method is accessible without instantiating the class.
4. ***void:*** keywords tell that this method will not return anything. The **main()** method is the entry point of our application.

5. **System.in:** This is the **standard input stream** that is used to read characters from the keyboard or any other standard input device.

6. **System.out:** This is the **standard output stream** that is used to produce the result of a program on an output device like the computer screen.

7. **println():** This method in Java is also used to display text on the console. It prints the text on the console and the cursor moves to the start of the next line at the console. The next printing takes place from the next line.

8. String []args: This is the argument passed to the main function which is an array of strings with the array name args. One can choose their own flexible name but this name is used by many developers.

Similarities and Difference between Java and C++

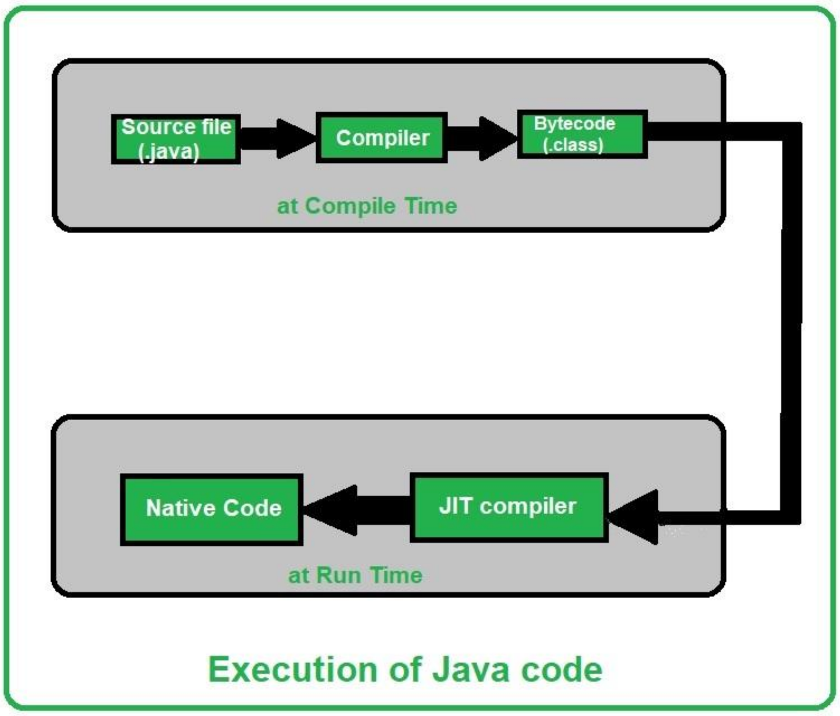
Nowadays Java and C++ programming languages are vastly used in competitive coding. Due to some awesome features, these two programming languages are widely used in industries as well. C++ is a widely popular language among coders for its efficiency, high speed, and dynamic memory utilization. Java is widely used in the IT industry, It is incomparable to any other programming language in terms of software development. Let us go through the various points to compare these popular coding languages:

Similarities between Java and C++

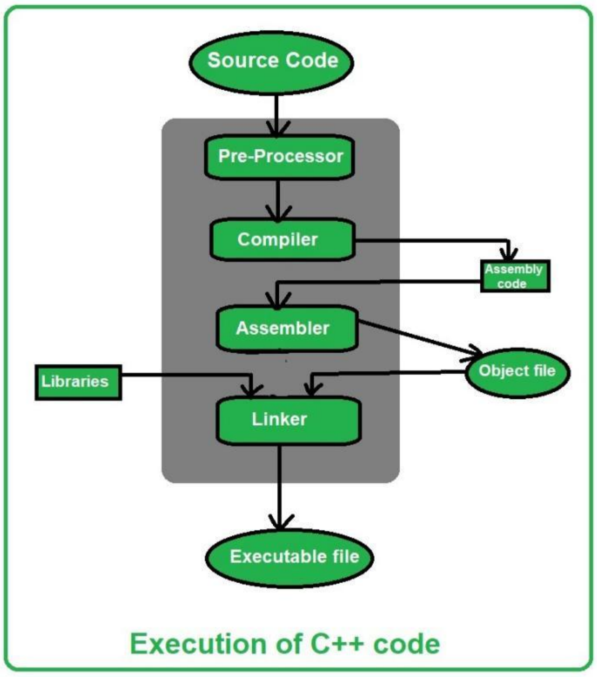
1. Execution: At compile-time, Java source code or **.java** file is converted into bytecode or **.class** file. At runtime, JVM (Java Virtual Machine) will load the **.class** file and will convert it to machine code with the help of an interpreter. After compilation of method calls (using the Just-In-Time (JIT) compiler), JVM will execute the optimized code. So Java is both compiled as well as an interpreted language. On the other hand, C++ executes the code by using only a compiler. The C++ compiler compiles and converts the source code into the machine code. That's why C++ is faster than Java but not platform-independent.

Below is the illustration of how Java and C++ codes are executed:

The execution of a Java code is as follows:



Execution of a C++ Code



2. Features: C++ and Java both have several [Object Oriented programming]() features which provide many useful programming functionalities. Some features are supported by one and some by the other. Even though both languages use the concept of OOPs, neither can be termed 100% object-oriented languages. Java uses primitive data types and thus cannot be termed as 100% Object-Oriented Language. C++ uses some data types similar to primitive ones and can implement methods without using any data type. And thus, it is also deprived of the 100% Object-Oriented title.
Below is the table which shows the features supported and not supported by both the programming languages:

Features	C++	Java
Abstraction	Yes	Yes
Encapsulation	Yes	Yes
Single Inheritance	Yes	Yes
Multiple Inheritance	Yes	No
Polymorphism	Yes	Yes
Static Binding	Yes	Yes
Dynamic Binding	Yes	Yes
Operator Overloading	Yes	No
Header Files	Yes	No
Pointers	Yes	No
Global Variables	Yes	No
Template Class	Yes	No
Interference and Packages	No	Yes
API	No	Yes

Applications: *Both C++ and Java have vast areas of application. Below are the applications of both languages:*

- ***Applications of C++ Programming language*:**

1. *Suitable for Developing large software (like passenger reservation systems).*
2. *MySQL is written in C++.*
3. *For fast execution, C++ is majorly used in Game Development.*
4. *Google Chromium browser, file system, and cluster data processing are all written in C++.*
5. *Adobe Premiere, Photoshop, and Illustrator; these popular applications are scripted in C++.*
6. *Advanced Computations and Graphics- real-time physical simulations, high-performance image processing.*
7. *C++ is also used in many advanced types of medical equipment like MRI machines, etc.*

- ***Applications of Java Programming language*:**

1. *Desktop GUI Applications development.*
2. *Android and Mobile application development.*
3. *Applications of Java are in embedded technologies like SIM cards, disk players, TV, etc.*
4. *Java EE (Enterprise Edition) provides an API and runtime environment for running large enterprise software.*
5. *Network Applications and Web services like Internet connection, Web App Development.*

Environment: *C++ is a Platform dependent while Java is a platform-independent programming language. We have to write and run C++ code on the same platform. Java has the* ***WORA (Write Once and Run Everywhere)*** *feature by which we can write our code in one platform once and we can run the code anywhere.*

Differences between Java and C++ are as follows:

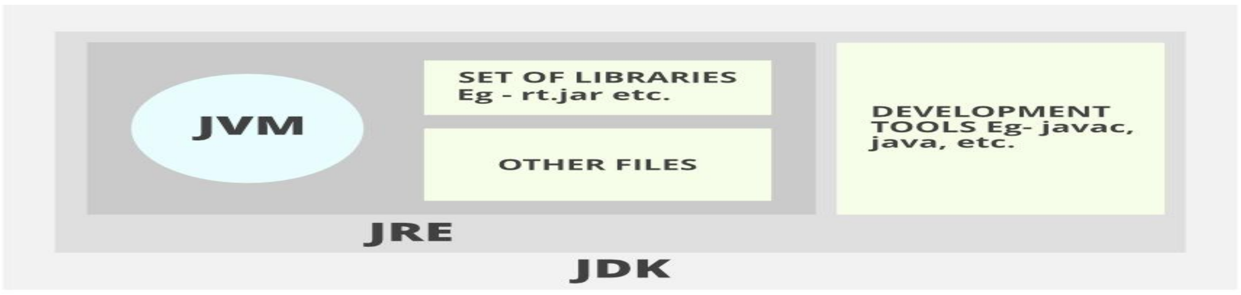
Parameters	Java	C++
Founder	Java was developed by James Gosling at Sun Microsystems.	C++ was developed by Bjarne Stroustrup at Bell Labs in 1979 as an extension of the C language.
First Release	On May 23, 1995	In October 1985
Stable Release	Java SE 18 released on 22 March 2022	C++20 released on 15th December 2020
Official Website	oracle.com/java	isocpp.org
Influenced By:	Java was Influenced by Ada 83, Pascal, C++, C#, etc. languages.	C++ was Influenced by Influenced by Ada, ALGOL 68, C, ML, Simula, Smalltalk, etc. languages.
Influenced to:	Java was influenced to develop BeanShell, C#, Clojure, Groovy, Hack, J#, Kotlin, PHP, Python, Scala, etc. languages.	C++ was influenced to develop C99, Java, JS++, Lua, Perl, PHP, Python, Rust, Seed7, etc. languages.
Platform Dependency	Platform independent, Java bytecode works on any operating system.	Platform dependent, should be compiled for different platforms.
Portability	It can run in any OS hence it is portable.	C++ is platform-dependent. Hence it is not portable.

Parameters	Java	C++
Compilation	Java is both Compiled and Interpreted Language.	C++ is a Compiled Language.
Memory Management	Memory Management is System Controlled.	Memory Management in C++ is Manual.
Virtual Keyword	It doesn’t have Virtual Keyword.	It has Virtual keywords.
Multiple Inheritance	It supports only single inheritance. Multiple inheritances are achieved partially using interfaces.	It supports both single and multiple Inheritance.
Overloading	It supports only method overloading and doesn’t allow operator overloading.	It supports both method and operator overloading.
Pointers	It has limited support for pointers.	It strongly supports pointers.
Libraries	It doesn’t support direct native library calls but only Java Native Interfaces.	It supports direct system library calls, making it suitable for system-level programming.
Libraries	Libraries have a wide range of classes for various high-level services.	C++ libraries have comparatively low-level functionalities.
Documentation Comment	It supports documentation comments (e.g., /**.. */) for source code.	It doesn’t support documentation comments for source code.
Thread Support	Java provides built-in support for multithreading.	C++ doesn’t have built-in support for threads, depends on third-party threading libraries.
Type	Java is only an object-oriented programming language.	C++ is both a procedural and an object-oriented programming language.
Input-Output mechanism	Java uses the (System class): **System.in** for input	C++ uses **cin** for input and **cout** for an output

Parameters	Java	C++
	and **System.out** for output.	operation.
goto Keyword	Java doesn't support goto Keyword	C++ supports goto keyword.
Structures and Unions	Java doesn't support Structures and Unions.	C++ supports Structures and Unions.
Parameter Passing	Java supports only the Pass by Value technique.	C++ supports both Pass by Value and pass by reference.
Global Scope	It supports no global scope.	It supports both global scope and namespace scope.
Object Management	Automatic object management with garbage collection.	It supports manual object management using new and delete.
Call by Value and Call by reference	Java supports only call by value.	C++ both supports call by value and call by reference.
Hardware	Java is not so interactive with hardware.	C++ is nearer to hardware.

Setting up the environment in Java

*Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, etc. Java applications are typically compiled to [bytecode] that can run on any Java virtual machine (JVM) regardless of computer architecture. The latest version is **Java 18**. Below are the environment settings for both Linux and Windows. JVM, JRE, and JDK three are all platform-dependent because the configuration of each Operating System is different. But, Java is platform-independent. Few things must be clear before setting up the environment which can better be perceived from the below image provided as follows:*


JVM
SET OF LIBRARIES Eg - rt.jar etc.
OTHER FILES
DEVELOPMENT TOOLS Eg- javac, java, etc.
JRE
JDK

- **JDK**(Java Development Kit): JDK is intended for software developers and includes development tools such as the Java compiler, Javadoc, Jar, and a debugger.
- **JRE**(Java Runtime Environment): JRE contains the parts of the Java libraries required to run Java programs and is intended for end-users. JRE can be viewed as a subset of JDK.
- **JVM:** JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides a runtime environment in which java bytecode can be executed. JVMs are available for many hardware and software platforms.

Java Basic Syntax

Java program is a collection of objects, and these objects communicate through method calls to each other to work together. Here is a brief discussion on the Classes and Objects, Method, Instance variables, syntax, and semantics of Java.

Basic terminologies in Java

1. **Class:** The class is a blueprint (plan) of the instance of a class (object). It can be defined as a template that describes the data and behavior associated with its instance.

- Example: Blueprint of the house is class.

2. **Object**: The object is an instance of a class. It is an entity that has behavior and state.

- Example: A car is an object whose **states** are: brand, color, and number plate.
- **Behavior:** Running on the road.

3. **Method**: The behavior of an object is the method.

- **Example**: The fuel indicator indicates the amount of fuel left in the car.

4. **Instance variables**: Every object has its own unique set of instance variables. The state of an object is generally created by the values that are assigned to these instance variables.

Example: Steps to compile and run a java program in a console

```
javac GFG.java
java GFG
```

5. public static void main(String [] args)

The method main() is the main entry point into a Java program; this is where the processing starts. Also allowed is the signature **public static void main(String… args)**.

6. Method Names

i. All the method names should start with a lowercase letter.

ii. If several words are used to form the name of the method, then each first letter of the inner word should be in Uppercase. Underscores are allowed, but not recommended. Also allowed are digits and currency symbols.

```
public void employeeRecords() // valid syntax
public void EmployeeRecords() // valid syntax, but discouraged
```

7. Identifiers in java

Identifiers are the names of local variables, instance and class variables, and labels, but also the names for classes, packages, modules and methods. All Unicode characters are valid, not just the ASCII subset.

i. All identifiers can begin with a letter, a currency symbol or an underscore (_). According to the convention, a letter should be lower case for variables.

ii. The first character of identifiers can be followed by any combination of letters, digits, currency symbols and the underscore. The underscore is not recommended for the names of variables. Constants (static final attributes and enums) should be in all Uppercase letters.

iii. Most importantly identifiers are case-sensitive.

iv. A keyword cannot be used as an identifier since it is a reserved word and has some special meaning.

```
Legal identifiers: MinNumber, total, ak74, hello_world, $amount, _under_value
Illegal identifiers: 74ak, -amount
```

8. White spaces in Java

A line containing only white spaces, possibly with the comment, is known as a blank line, and the Java compiler totally ignores it.

9. Access Modifiers: These modifiers control the scope of class and methods.

- **Access Modifiers**: default, public, protected, private
- **Non-access Modifiers**: final, abstract, strictfp.

10. Understanding Access Modifiers:

Access Modifier	Within Class	Within Package	Outside Package by subclass only	Outside Package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

11. Java Keywords

Keywords or Reserved words are the words in a language that are used for some internal process or represent some predefined actions. These words are therefore not allowed to use as variable names or objects.

abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void
volatile	while		

Java Platforms / Editions

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) Java EE (Java Enterprise Edition)

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

3) Java ME (Java Micro Edition)

It is a micro platform that is dedicated to mobile applications.

4) JavaFX

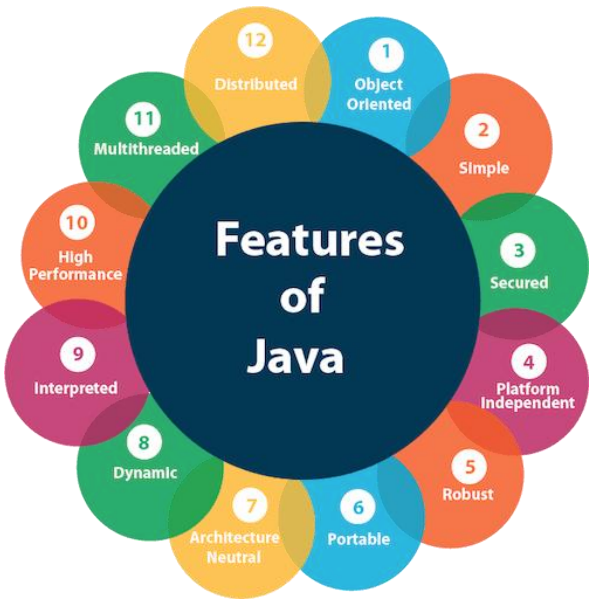
It is used to develop rich internet applications. It uses a lightweight user interface API.

Features of Java

The primary objective of Java programming

language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.

A list of the most important features of the Java language is given below.



1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

Differences between JDK, JRE and JVM

__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.

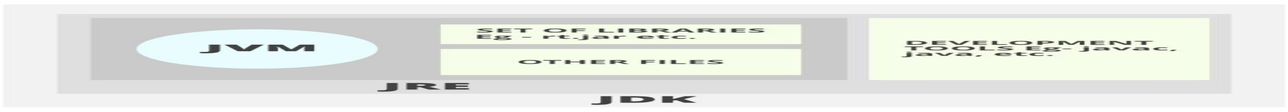
Now we need an environment to make a run of our program. Henceforth, __JRE__ stands for "__Java Runtime Environment__" and may also be written as "__Java RTE.__" The Java Runtime Environment provides the minimum requirements for executing a Java application; it consists of the Java Virtual Machine (JVM), core classes, and supporting files.

Now let us discuss __JVM__, which stands out for java virtual machines. It is as follows:

- *A __specification__ where the working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Sun and other companies.*

- An **implementation** is a computer program that meets the requirements of the JVM specification.
- **Runtime Instance** Whenever you write a java command on the command prompt to run the java class, an instance of JVM is created.

Before proceeding to the differences between JDK, JRE, and JVM, let us discuss them in brief first and interrelate them with the image below proposed.


JVM
SET OF LIBRARIES Eg - rt.jar etc.
OTHER FILES
JRE
DEVELOPMENT TOOLS Eg- javac, java, etc.
JDK

Don't get confused as we are going to discuss all of them one by one.

1. **JDK** (Java Development Kit) is a Kit that provides the environment to **develop and execute(run)** the Java program. JDK is a kit(or package) that includes two things

- Development Tools(to provide an environment to develop your java programs)
- JRE (to execute your java program).

2. **JRE** (Java Runtime Environment) is an installation package that provides an environment to **only run(not develop)** the java program(or application)onto your machine. JRE is only used by those who only want to run Java programs that are end-users of your system.

3. JVM (Java Virtual Machine) is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line, hence it is also known as an *interpreter*.

Now let us discuss the components of JRE in order to understand its importance of it and perceive how it actually works. For this let us discuss components.

The components of JRE are as follows:

1. **Deployment technologies**, including deployment, Java Web Start, and Java Plug-in.
2. **User interface toolkits**, including *Abstract Window Toolkit (AWT), Swing, Java 2D, Accessibility, Image I/O, Print Service, Sound, drag, and drop (DnD),* and *input methods.*
3. **Integration libraries**, including *Interface Definition Language (IDL), Java Database Connectivity (JDBC), Java Naming and Directory Interface (JNDI), Remote Method Invocation (RMI), Remote Method Invocation Over Internet Inter-Orb Protocol (RMI-IIOP),* and *scripting.*
4. **Other base libraries**, including *international support, input/output (I/O), extension mechanism, Beans, Java Management Extensions (JMX), Java Native Interface (JNI), Math, Networking, Override Mechanism, Security, Serialization,* and *Java for XML Processing (XML JAXP).*
5. **Lang and util base libraries**, including *lang and util, management, versioning, zip, instrument, reflection, Collections, Concurrency Utilities, Java Archive (JAR), Logging, Preferences API, Ref Objects,* and *Regular Expressions.*
6. **Java Virtual Machine (JVM)**, including *Java HotSpot Client* and *Server Virtual Machines.*

Variables in Java

Variable in Java is a data container that saves the data values during Java program execution. Every variable is assigned a data type that designates the type and quantity of value it can hold. A variable is a memory location name for the data.

A variable is a name given to a memory location. It is the basic unit of storage in a program.

- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location. All the operations done on the variable affect that memory location.
- In Java, all variables must be declared before use.

Types of Variables in Java

Now let us discuss different types of variables which are listed as follows:

1. Local Variables
2. Instance Variables
3. Static Variables

Let us discuss the traits of every type of variable listed here in detail.

1. Local Variables

A variable defined within a block or method or constructor is called a local variable.

- These variables are created when the block is entered, or the function is called and destroyed after exiting from the block or when the call returns from the function.
- The scope of these variables exists only within the block in which the variables are declared, i.e., we can access these variables only within that block.
- Initialization of the local variable is mandatory before using it in the defined scope.

- Java

```
/*package whatever //do not write package name
here */
// Contributed by Shubham Jain
import java.io.*;

class GFG {
    public static void main(String[] args)
    {
        int var = 10; // Declared a Local
Variable
        // This variable is local to this main
method only
        System.out.println("Local Variable: " +
var);
    }
}
```

Output

```
Local Variable: 10
```

2. Instance Variables

Instance variables are non-static variables and are declared in a class outside of any method, constructor, or block.

- As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
- Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier, then the default access specifier will be used.
- Initialization of an instance variable is not mandatory. Its default value is 0.
- Instance variables can be accessed only by creating objects.

- Java

```
/*package whatever //do not write package name here */

import java.io.*;

class GFG {

    public String geek; // Declared Instance Variable

    public GFG()
    { // Default Constructor

        this.geek = "Shubham Jain"; // initializing
Instance Variable
    }
//Main Method
    public static void main(String[] args)
    {

        // Object Creation
        GFG name = new GFG();
        // Displaying O/P
        System.out.println("Geek name is: " +
name.geek);
    }
}
```

Output

```
Geek name is: Shubham Jain
```

3. Static Variables

Static variables are also known as class variables.

- These variables are declared similarly as instance variables. The difference is that static variables are declared using the static keyword within a class outside of any method, constructor or block.
- Unlike instance variables, we can only have one copy of a static variable per class, irrespective of how many objects we create.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.
- Initialization of a static variable is not mandatory. Its default value is 0.

- *If we access a static variable like an instance variable (through an object), the compiler will show a warning message, which won't halt the program. The compiler will replace the object name with the class name automatically.*
- *If we access a static variable without the class name, the compiler will automatically append the class name.*

- Java

```
/*package whatever //do not write package name here */

import java.io.*;

class GFG {

  public static String geek = "Shubham Jain";          //Declared
static variable

    public static void main (String[] args) {

      //geek variable can be accessed without object creation
      //Displaying O/P
      //GFG.geek --> using the static variable
        System.out.println("Geek Name is : "+GFG.geek);
    }
}
```

Output

```
Geek Name is : Shubham Jain
```

Differences between the Instance variables and the Static variables

Now let us discuss the differences between the Instance variables and the Static variables:

- *Each object will have its own copy of an instance variable, whereas we can only have one copy of a static variable per class, irrespective of how many objects we create.*
- *Changes made in an instance variable using one object will not be reflected in other objects as each object has its own copy of the instance variable. In the case of a static variable, changes will be reflected in other objects as static variables are common to all objects of a class.*
- *We can acces*

Operators in Java

Operator in Java

is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,

- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Arrays in Java

An array in Java *is a group of like-typed variables referred to by a common name. Arrays in Java work differently than they do in C/C++. Following are some important points about Java arrays.*

One-Dimensional Arrays:

Example:

```
int intArray[];    //declaring array
intArray = new int[20];  // allocating memory to array
```

OR

```
int[] intArray = new int[20]; // combining both statements in one
```

Array Literal

In a situation where the size of the array and variables of the array are already known, array literals can be used.

```
 int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };
 // Declaring array literal
```

Accessing Java Array Elements using for Loop

Each element in the array is accessed via its index. The index begins with 0 and ends at (total array size)-1. All the elements of array can be accessed using Java for Loop.

```
 // accessing the elements of the specified array
for (int i = 0; i < arr.length; i++)
  System.out.println("Element at index " + i +
                             " : "+ arr[i]);
```

Time Complexity: *O(n)*
Auxiliary Space *: O(1)*

Arrays of Objects

An array of objects is created like an array of primitive type data items in the following way.

```
Student[] arr = new Student[7]; //student is a user-defined class
```

Multidimensional Arrays:

Multidimensional arrays are **arrays of arrays** with each element of the array holding the reference of other arrays. These are also known as Jagged Arrays. A multidimensional array is created by appending one set of square brackets ([]) per dimension.

```
int[][] intArray = new int[10][20]; //a 2D array or matrix
int[][][] intArray = new int[10][20][10]; //a 3D array
```

Strings in Java

String literal

```
String s = "GeeksforGeeks";
```

Using *new* keyword

```
String s = new String ("GeeksforGeeks");
```

StringBuffer is a peer class of **String** that provides much of the functionality of strings. The string represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.

Syntax:

```
StringBuffer s = new StringBuffer("GeeksforGeeks");
```

StringBuilder in Java represents a mutable sequence of characters. Since the String Class in Java creates an immutable sequence of characters, the StringBuilder class provides an alternate to String Class, as it creates a mutable sequence of characters.

Syntax:

```
StringBuilder str = new StringBuilder();
str.append("GFG");
```

StringTokenizer class in Java is used to break a string into tokens.

Example:

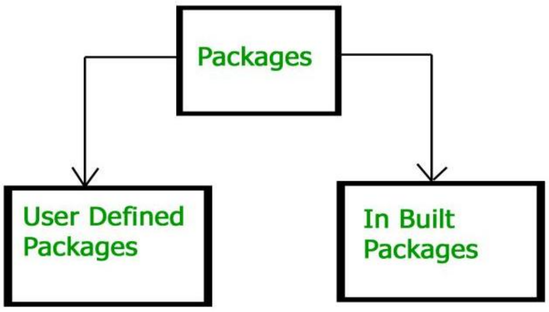
A StringTokenizer object internally maintains a current position within the string to be tokenized. Some operations advance this current position past the characters processed. A token is returned by taking a substring of the string that was used to create the StringTokenizer object.

Java's Selection statements: or Control Statements :

- if
- if-else
- nested-if
- if-else-if
- switch-case
- jump – break, continue, return

Packages In Java

A java package is a group of similar types of classes, interfaces and sub-packages.
Package in java can be categorized in two form, built-in package and user-defined package.
There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
Here, we will have the detailed learning of creating and using user-defined packages.
Types of packages:



Built-in Packages

These packages consist of a large number of classes which are a part of Java **API**.Some of the commonly used built-in packages are:

1) **java.lang**: Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.
2) **java.io**: Contains classed for supporting input / output operations.
3) **java.util**: Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
4) **java.applet**: Contains classes for creating Applets.
5) **java.awt**: Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).
6) **java.net**: Contain classes for supporting networking operations.

User-defined packages

These are the packages that are defined by the user. First we create a directory **myPackage** (name should be same as the name of the package). Then create the **MyClass** inside the directory with the first statement being the **package names**.

```
// Name of the package must be same as the directory
// under which this file is saved
package myPackage;
public class MyClass
{   public void getNames(String s)
    {    System.out.println(s);  }}
```

Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not the method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship**.

Marker interface in Java

It is an empty interface (no field or methods). Examples of marker interface are Serializable, Cloneable and Remote interface. All these interfaces are empty interfaces.

1.Cloneable interface : Cloneable interface is present in java.lang package. There is a method clone() in Object class. A class that implements the Cloneable interface indicates that it is legal for clone() method to make a field-for-field copy of instances of that class.

```
// Java program to illustrate Cloneable interface
import java.lang.Cloneable;
// By implementing Cloneable interface
// we make sure that instances of class A
// can be cloned.
class A implements Cloneable
{
        int i;
        String s;
        // A class constructor
        public A(int i,String s)
        {
                this.i = i;
                this.s = s;
        }

        // Overriding clone() method
        // by simply calling Object class
        // clone() method.
        @Override
        protected Object clone()
                throws CloneNotSupportedException
        {
                return super.clone();
        }
}
public class Test
{
        public static void main(String[] args)
                throws CloneNotSupportedException
        {
                A a = new A(20, "GeeksForGeeks");
                // cloning 'a' and holding
                // new cloned object reference in b
                // down-casting as clone() return
type is Object
                A b = (A)a.clone();
                System.out.println(b.i);
                System.out.println(b.s);}}
```

Output:

```
20
GeeksForGeeks
```

2.Serializable interface : Serializable interface is present in java.io package. It is used to make an object eligible for saving its state into a file. This is called Serialization.
Classes that do not implement this interface will not have any of their state serialized or deserialized. All subtypes of a serializable class are themselves serializable.

```
// Java program to illustrate Serializable
interface
import java.io.*;

// By implementing Serializable interface
// we make sure that state of instances of class
A
// can be saved in a file.
class A implements Serializable
{
        int i;
        String s;

        // A class constructor
        public A(int i,String s)
        {
                this.i = i;
                this.s = s;
        }
}

public class Test
{
        public static void main(String[] args)
        throws IOException,
ClassNotFoundException
        {
                A a = new A(20,"GeeksForGeeks");

                // Serializing 'a'
                FileOutputStream fos = new
FileOutputStream("xyz.txt");
                ObjectOutputStream oos = new
ObjectOutputStream(fos);
                oos.writeObject(a);

                // De-serializing 'a'
                FileInputStream fis = new
FileInputStream("xyz.txt");
                ObjectInputStream ois = new
ObjectInputStream(fis);
                A b = (A)ois.readObject();//down-
casting object

                System.out.println(b.i+" "+b.s);

                // closing streams
                oos.close();
                ois.close();
        }
}
```

```
20 GeeksForGeeks
```

3.Remote interface : Remote interface is present in java.rmi package. A remote object is an object which is stored at one machine and accessed from another machine. So, to make an object a remote object, we need to flag it with Remote interface. Here, Remote interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine.

Exceptions in Java

Exception Handling in Java is one of the effective means to handle the runtime errors so that the regular flow of the application can be preserved. Java Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Ex:-exit program,ignore,deal with execution handling and continue.

Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions. Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.(during the execution like hang the phone,responding the apps.)

Major reasons why an exception Occurs

- Invalid user input

- Device failure
- Loss of network connection
- Physical limitations (out of disk memory)
- Code errors
- Opening an unavailable file

Errors represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc. Errors are usually beyond the control of the programmer, and we should not try to handle errors.

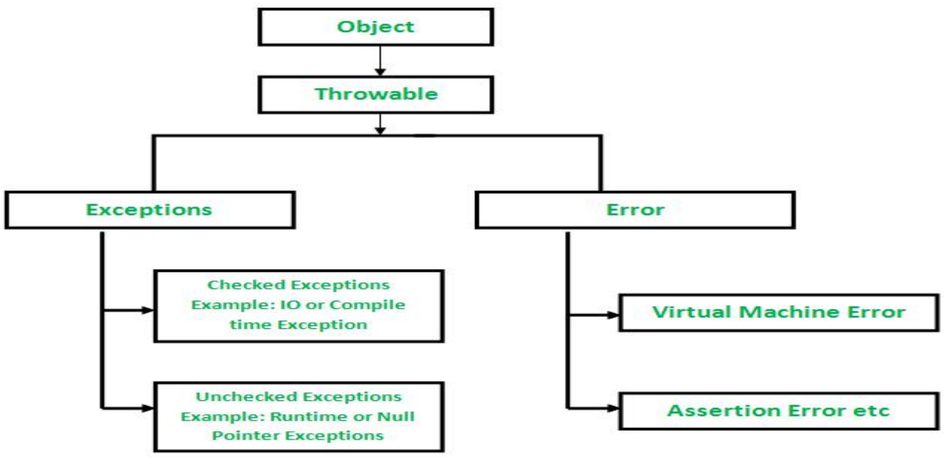
- **Complied time error**:- Syntax checking
- **Runtime error**:- Missing input file,dividing with zero,perform operation.

Let us discuss the most important part which is the **differences between Error and Exception** that is as follows:

- **Error**: An Error indicates a serious problem that a reasonable application should not try to catch.
- **Exception**: Exception indicates conditions that a reasonable application might try to catch.

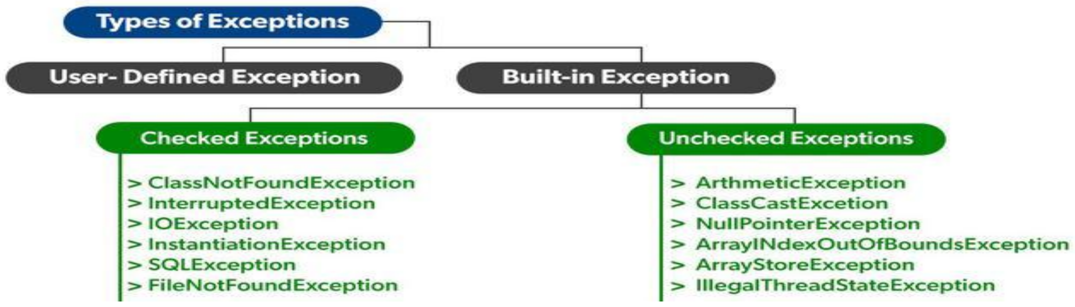
Exception Hierarchy

All exception and error types are subclasses of class **Throwable**, which is the base class of the hierarchy. One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception. Another branch, **Error** is used by the Java run-time system(JVM) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.


Object
Throwable
Exceptions
Error
Checked Exceptions Example: IO or Compile time Exception
Unchecked Exceptions Example: Runtime or Null Pointer Exceptions
Virtual Machine Error
Assertion Error etc

Types of Exceptions

Java defines several types of exceptions that relate to its various class libraries. Java also allows users to define their own exceptions.



Exceptions can be categorized in two ways:

1. **Built-in Exceptions**
   - Checked Exception
   - Unchecked Exception
2. **User-Defined Exceptions**

Let us discuss the above-defined listed exception that is as follows:

A. Built-in Exceptions:

Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations.

- **Checked Exceptions(Complied):** Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.

- **Unchecked Exceptions(during execution):** The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error.

B. User-Defined Exceptions:

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions, which are called 'user-defined Exceptions'.

The ***advantages of Exception Handling in Java*** are as follows:

1. Provision to Complete Program Execution
2. Easy Identification of Program Code and Error-Handling Code
3. Propagation of Errors
4. Meaningful Error Reporting
5. Identifying Error Types

Try, catch, throw and throws in Java

What is an Exception?

An exception is an "unwanted or unexpected event", which occurs during the execution of the program i.e, at run-time, that disrupts the normal flow of the program's instructions. When an exception occurs, the execution of the program gets terminated.

Why does an Exception occur?

An exception can occur due to several reasons like a Network connection problem, Bad input provided by a user, Opening a non-existing file in your program, etc

Blocks & Keywords used for exception handling

1. try: The try block contains a set of statements where an exception can occur.

```
try
{
    // statement(s) that might cause exception
}
```

2. **catch**: The catch block is used to handle the uncertain condition of a try block. A try block is always followed by a catch block, which handles the exception that occurs in the associated try block.

```
catch
{
   // statement(s) that handle an exception
   // examples, closing a connection, closing
   // file, exiting the process after writing
   // details to a log file.
}
```

3. **throw**: The throw keyword is used to transfer control from the try block to the catch block.

4. **throws**: The throws keyword is used for exception handling without try & catch block. It specifies the exceptions that a method can throw to the caller and does not handle itself.

5. **finally**: It is executed after the catch block. We use it to put some common code (to be executed irrespective of whether an exception has occurred or not) when there are multiple catch blocks.

```
// Java program to demonstrate working of try,
// catch and finally
class Division {
        public static void main(String[] args)
        {
        int a = 10, b = 5, c = 5, result;
                try {
                        result = a / (b - c);
                                System.out.println("result"
+ result);
                }
                catch (ArithmeticException e) {

        System.out.println("Exception
caught:Division by zero");
                }
                finally {
                    System.out.println("I am in
final block");}}}
```

Output:

```
Exception caught:Division by zero
I am in final block
```

An example of throws keyword:

```
// Java program to demonstrate working of
throws
class ThrowsExecp {
        // This method throws an exception
        // to be handled
        // by caller or caller
        // of caller and so on.
        static     void     fun()     throws
IllegalAccessException
        {
                System.out.println("Inside fun(). ");
                throw                          new
IllegalAccessException("demo");
        }

// This is a caller function
public static void main(String args[])
{
        try {
                fun();
        }
        catch (IllegalAccessException e) {
                System.out.println("caught
in main.");     }}}
```

Output:

Inside fun().

caught in main.

Multithreading in Java

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process. **Multithreading in Java** is a process of executing multiple threads simultaneously. A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

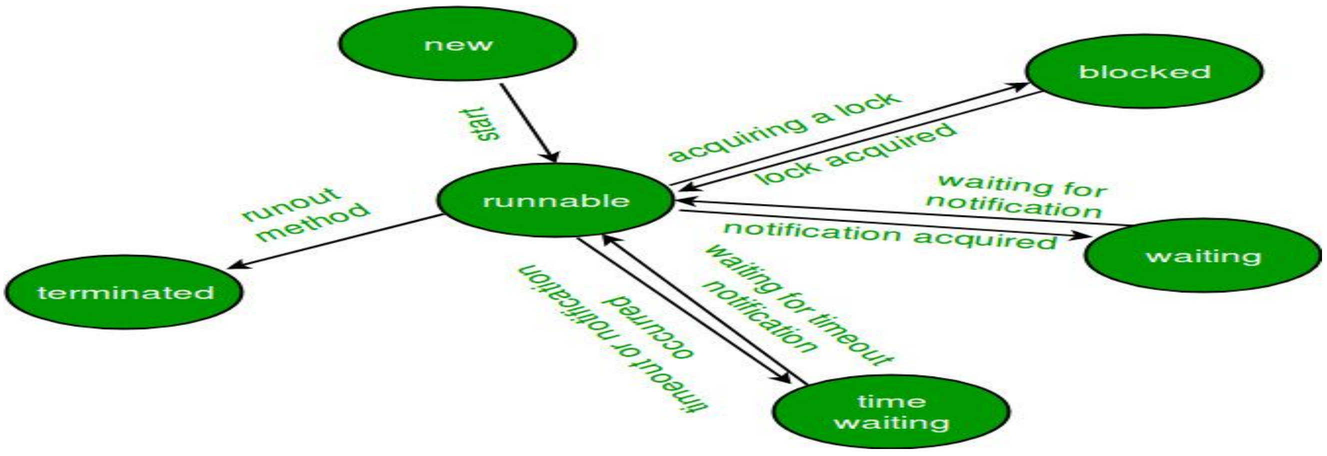
Lifecycle and States of a Thread in Java

A thread in Java at any point of time exists in any one of the following states. A thread lies only in one of the shown states at any instant:

1. *New*
2. *Runnable*
3. *Blocked*
4. *Waiting*

5. Timed Waiting
6. Terminated

The diagram shown below represents various states of a thread at any instant in time.



Life Cycle of a thread

1. **New Thread:** When a new thread is created, it is in the new state. The thread has not yet started to run when the thread is in this state. When a thread lies in the new state, its code is yet to be run and hasn't started to execute.
2. **Runnable State:** A thread that is ready to run is moved to a runnable state. In this state, a thread might actually be running or it might be ready to run at any instant of time. It is the responsibility of the thread scheduler to give the thread, time to run.
   A multi-threaded program allocates a fixed amount of time to each individual thread. Each and every thread runs for a short while and then pauses and relinquishes the CPU to another thread so that other threads can get a chance to run. When this happens, all such threads that are ready to run, waiting for the CPU and the currently running thread lie in a runnable state.
3. **Blocked/Waiting state:** When a thread is temporarily inactive, then it's in one of the following states:

- Blocked
- Waiting

4. **Timed Waiting:** A thread lies in a timed waiting state when it calls a method with a time-out parameter. A thread lies in this state until the timeout is completed or until a notification is received. For example, when a thread calls sleep or a conditional wait, it is moved to a timed waiting state.
5. **Terminated State:** A thread terminates because of either of the following reasons:

- Because it exits normally. This happens when the code of the thread has been entirely executed by the program.
- Because there occurred some unusual erroneous event, like segmentation fault or an unhandled exception.

Synchronization in Java

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.
Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results.

Types of synchronization:

There are two types of synchronization that are as follows:

1. Process synchronization
2. Thread synchronization

- **Daemon Thread in Java**

Daemon thread in Java is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.
There are many java daemon threads running automatically e.g. gc, finalizer etc.
You can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads etc.
In background like music player,clock etc.

#Java I/O Tutorial-

Java I/O (Input and Output) is used to process the input and produce the output.
Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.
We can perform file handling in Java by Java I/O API.
Stream
A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.
In Java, 3 streams are created for us automatically. All these streams are attached with the console.
1) System.out: standard output stream
2) System.in: standard input stream
3) System.err: standard error stream

Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.

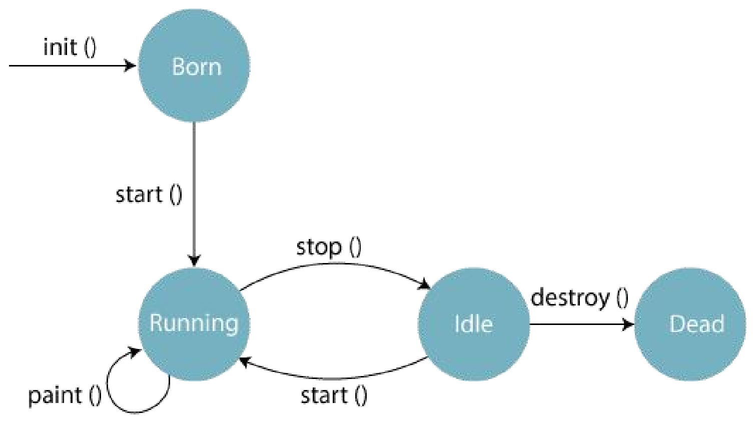
Applet Life Cycle in Java

In Java, an applet is a special type of program embedded in the web page to generate dynamic content. Applet is a class in Java.

The applet life cycle can be defined as the process of how the object is created, started, stopped, and destroyed during the entire execution of its application. It basically has five core methods namely init(), start(), stop(), paint() and destroy().These methods are invoked by the browser to execute.

Along with the browser, the applet also works on the client side, thus having less processing time.

Methods of Applet Life Cycle



There are five methods of an applet life cycle, and they are:

- **init():** The init() method is the first method to run that initializes the applet. It can be invoked only once at the time of initialization. The web browser creates the initialized objects, i.e., the web browser (after checking the security settings) runs the init() method within the applet.
- **start():** The start() method contains the actual code of the applet and starts the applet. It is invoked immediately after the init() method is invoked. Every time the browser is loaded or refreshed, the start() method is invoked. It is also invoked whenever the applet is maximized, restored, or moving from one tab to another in the browser. It is in an inactive state until the init() method is invoked.
- **stop():** The stop() method stops the execution of the applet. The stop () method is invoked whenever the applet is stopped, minimized, or moving from one tab to another in the browser, the stop() method is invoked. When we go back to that page, the start() method is invoked again.
- **destroy():** The destroy() method destroys the applet after its work is done. It is invoked when the applet window is closed or when the tab containing the webpage is closed. It removes the applet object from memory and is executed only once. We cannot start the applet once it is destroyed.
- **paint():** The paint() method belongs to the Graphics class in Java. It is used to draw shapes like circle, square, trapezium, etc., in the applet. It is executed after the start() method and when the browser or applet windows are resized.

Sequence of method execution when an applet is executed:

1. init()
2. start()
3. paint()

Sequence of method execution when an applet is executed:

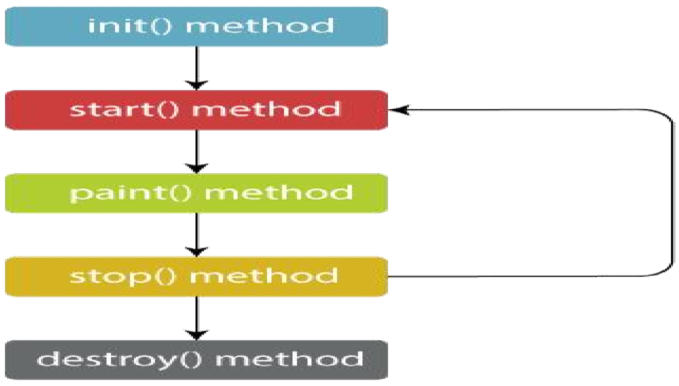
1. stop()
2. destroy()

Applet Life Cycle Working

- The Java plug-in software is responsible for managing the life cycle of an applet.
- An applet is a Java application executed in any web browser and works on the client-side. It doesn't have the main() method because it runs in the browser. It is thus created to be placed on an HTML page.
- The init(), start(), stop() and destroy() methods belongs to the **applet.Applet** class.
- The paint() method belongs to the **awt.Component** class.
- In Java, if we want to make a class an Applet class, we need to extend the **Applet**
- Whenever we create an applet, we are creating the instance of the existing Applet class. And thus, we can use all the methods of that class.

Flow of Applet Life Cycle:

These methods are invoked by the browser automatically. There is no need to call them explicitly.



The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Java Networking :- *Java Networking is a concept of connecting two or more computing devices together so that we can share resources. Java socket programming provides facility to share data between different computing devices.*

Advantage of Java Networking

1. *Sharing resources*
2. *Centralize software management*

The java.net package supports two protocols,
TCP: Transmission Control Protocol provides reliable communication between the sender and receiver. TCP is used along with the Internet Protocol referred as TCP/IP.
UDP: User Datagram Protocol provides a connection-less protocol service by allowing packet of data to be transferred along two or more nodes

Java Networking Terminology

The widely used Java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet
- SMTP
- POP etc.

3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

4) MAC Address

MAC (Media Access Control) address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC address.

For example, an ethernet card may have a **MAC** address of 00:0d:83::b1:c0:8e.

5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

6) Socket

Java Socket programming is used for communication between the applications running on different JRE.

Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

A socket is an endpoint between two way communications.

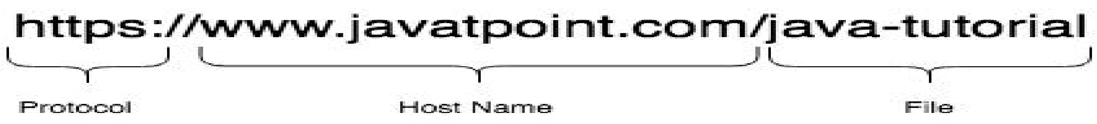
java.net package

The java.net package can be divided into two sections:

1. **A Low-Level API:** It deals with the abstractions of addresses i.e. networking identifiers, Sockets i.e. bidirectional data communication mechanism and Interfaces i.e. network interfaces.
2. **A High Level API:** It deals with the abstraction of URIs i.e. Universal Resource Identifier, URLs i.e. Universal Resource Locator, and Connections i.e. connections to the resource pointed by URLs.

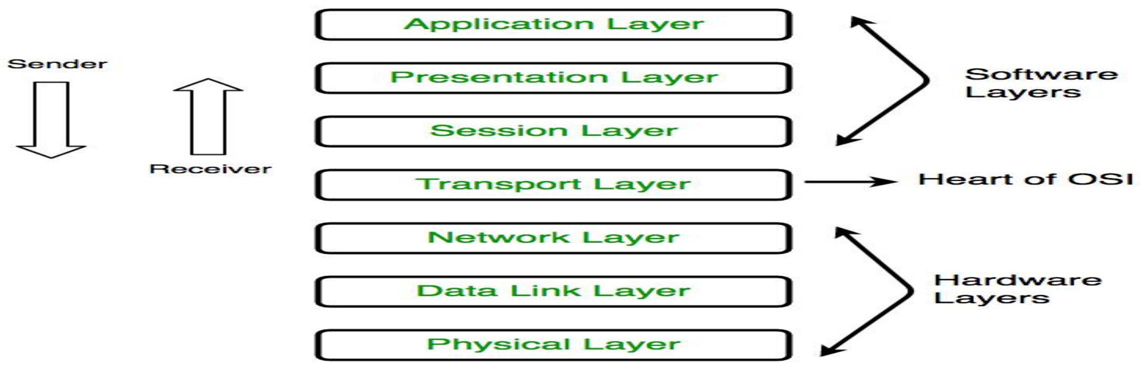
Java URL

The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example



A URL contains many information:

1. **Protocol:** In this case, http is the protocol.
2. **Server name or IP Address:** In this case, www.javatpoint.com is the server name.
3. **Port Number:** It is an optional attribute. If we write http//ww.javatpoint.com:80/sonoojaiswal/ , 80 is the port number. If port number is not mentioned in the URL, it returns -1.
4. **File Name or directory name:** In this case, index.jsp is the file name.



Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

*An **event** can be defined as changing the state of an object or behavior by performing actions.*

Actions can be a button click, cursor movement, keypress through keyboard or page scrolling, etc.

The **java.awt.event** package can be used to provide various event classes.

Java Event classes and Listener interfaces

Event Class	Listener Interface	Description
ActionEvent	ActionListener	An event that indicates that a component-defined action occurred like a button click or selecting an item from the menu-item list.
AdjustmentEvent	AdjustmentListener	The adjustment event is emitted by an Adjustable object like Scrollbar.
ComponentEvent	ComponentListener	An event that indicates that a component moved, the size changed or changed its visibility.
ContainerEvent	ContainerListener	When a component is added to a container (or) removed from it, then this event is generated by a container object.
FocusEvent	FocusListener	These are focus-related events, which include focus, focusin, focusout, and blur.
ItemEvent	ItemListener	An event that indicates whether an item was selected or not.
KeyEvent	KeyListener	An event that occurs due to a sequence of keypresses on the keyboard.
MouseEvent	MouseListener & MouseMotionListener	The events that occur due to the user interaction with the mouse (Pointing Device).
MouseWheelEvent	MouseWheelListener	An event that specifies that the mouse wheel was rotated in a component.
TextEvent	TextListener	An event that occurs when an object's text changes.
WindowEvent	WindowListener	An event which indicates whether a window has changed its status or not.

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

Different interfaces consists of different methods which are specified below.

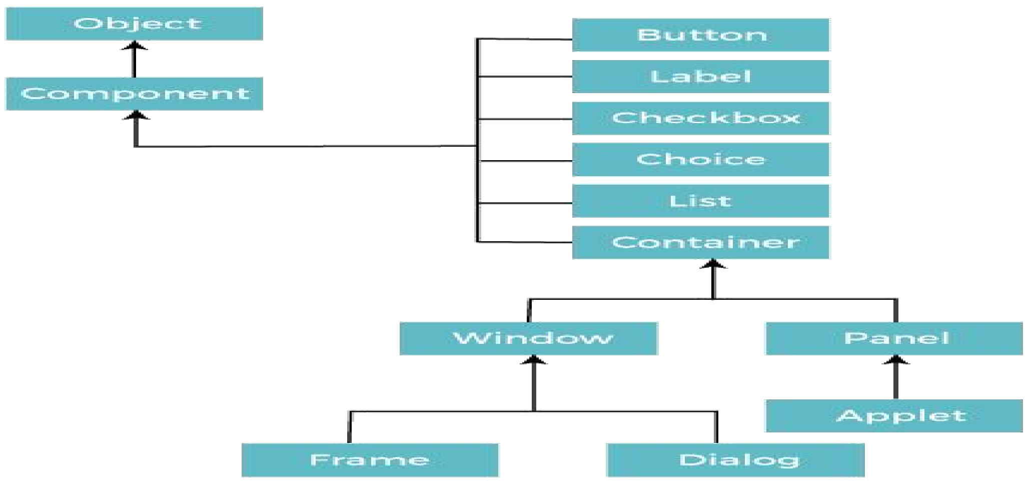
Listener Interface	Methods
ActionListener	• actionPerformed()
AdjustmentListener	• adjustmentValueChanged( )
ComponentListener	• componentResized() • componentShown() • componentMoved() • componentHidden()
ContainerListener	• componentAdded() • componentRemoved()

Listener Interface	Methods
FocusListener	• focusGained() • focusLost()
ItemListener	• itemStateChanged()
KeyListener	• keyTyped() • keyPressed() • keyReleased()
MouseListener	• mousePressed() • mouseClicked() • mouseEntered() • mouseExited() • mouseReleased()
MouseMotionListener	• mouseMoved() • mouseDragged()
MouseWheelListener	• mouseWheelMoved()
TextListener	• textChanged()
WindowListener	• windowActivated() • windowDeactivated() • windowOpened() • windowClosed() • windowClosing() • windowIconified() • windowDeiconified()

AWT

AWT *stands for* **Abstract Window Toolkit**.*It is a platform-dependent API to develop GUI (Graphical User Interface) or window-based applications in Java. It was developed by Sun Microsystem In 1995. It is heavy-weight in use because it is generated by the system's host operating system. It contains a large number of classes and methods, which are used for creating and managing GUI.*

Java AWT Hierarchy



Characteristics

- *It is a set of native user interface components.*
- *It is very robust in nature.*
- *It includes various editing tools like graphics tool and imaging tools.*
- *It uses native window-system controls.*
- *It provides functionality to include shapes, colors and font classes.*

Advantages

- *It takes very less memory for development of GUI and executing programs.*
- *It is highly stable as it rarely crashes.*
- *It is dependent on operating system so performance is high.*
- *It is easy to use for beginners due to its easy interface.*

Disadvantages

- *The buttons of AWT does not support pictures.*
- *It is heavyweight in nature.*
- *Two very important components trees and tables are not present.*
- *Extensibility is not possible as it is platform dependent*

Components

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as **Frame, Dialog** and **Panel**.

It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

Note: A container itself is a component (see the above diagram), therefore we can add a container inside container.

Types of containers:

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

Panel

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

Frame

The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

Useful Methods of Component Class

Method	Description
public void add(Component c)	Inserts a component on this component.
public void setSize(int width,int height)	Sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	Defines the layout manager for the component.
public void setVisible(boolean status)	Changes the visibility of the component, by default false.

Java AWT Example

To create simple AWT example, you need a frame. There are two ways to create a GUI using Frame in AWT.

1. By extending Frame class (**inheritance**)
2. By creating the object of Frame class (**association**)

Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. The **Java LayoutManagers** facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout

4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

Java BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only. It is the default layout of a frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Java GridLayout

The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

Java FlowLayout

The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.

Fields of FlowLayout class

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Java BoxLayout

The **Java BoxLayout class** is used to arrange the components either vertically or horizontally. For this purpose, the BoxLayout class provides four constants. They are as follows:

Note: The BoxLayout class is found in javax.swing package.

Constructor of BoxLayout class

1. **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

Java CardLayout

The **Java CardLayout** class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout Class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Java GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.

The components may not be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of the constraints object, we arrange the component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine the component's size. GridBagLayout components are also arranged in the rectangular grid but can have many different sizes and can occupy multiple rows or columns.

Constructor

GridBagLayout(): The parameterless constructor is used to create a grid bag layout manager.

Class

1. Class is a set of object which shares common characteristics/ behavior and common properties/ attributes.

2. Class is not a real world entity. It is just a template or blueprint or prototype from which objects are created.

3. Class does not occupy memory.
4. Class is a group of variables of different data types and group of methods.

A class in java can contain:

• data member
• method
• constructor
• nested class and
• interface

Object

It is a basic unit of Object-Oriented Programming and represents real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

1. **State**: It is represented by attributes of an object. It also reflects the properties of an object.
2. **Behavior**: It is represented by methods of an object. It also reflects the response of an object with other objects.
3. **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

Abstract Class in Java

An **abstract** class in Java is one that is declared with the abstract keyword. It may have both abstract and non-abstract methods(methods with bodies). An **abstract** is a **java modifier** applicable for **classes** and **methods** in java but ***not for Variables***.

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**. If we have to perform only one operation, having same name of the methods increases the readability of the program.

Advantages of Method Overloading

- Method overloading improves the Readability and reusability of the program.
- Method overloading reduces the complexity of the program.
- Using method overloading, programmers can perform a task efficiently and effectively.
- Using method overloading, it is possible to access methods performing related functions with slightly different arguments and types.
- Objects of a class can also be initialized in different ways using the constructors.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

static Keyword in Java

The **static keyword** in Java is mainly used for memory management. The static keyword in Java is used to share the same variable or method of a given class. The users can apply static keywords with variables, methods, blocks, and nested classes. The static keyword belongs to the class than an instance of the class. The static keyword is used for a constant variable or a method that is the same for every instance of a class. It has execute first of all even to main function(Extra add line by Kunal).

The *static* keyword is a non-access modifier in Java that is applicable for the following:

1. Blocks
2. Variables
3. Methods
4. Classes

Super Keyword in Java

The **super** keyword in java is a reference variable that is used to refer to parent class objects. An understanding of Inheritance and Polymorphism is needed in order to understand the super keyword. Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

this keyword in Java

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.

Usage of Java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

final Keyword in Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

2) Java final method

If you make any method as final, you cannot override it.

3) Java final class

If you make any class as final, you cannot extend it.