- **Procedural language Approach**

A procedural language is a sort of computer programming language that has a set of functions, instructions, and statements that must be **executed in a certain order** to accomplish a job or program. In general, procedural language is used to specify the steps that the computer takes to solve a problem.

Computer procedural languages include BASIC, C, FORTRAN, Java, and Pascal, to name a few. To create programs, they use variables, conditional statements, and functions that permit a computer to process and provide the desired output. Problems with this type of programming is that, as programs grows larger complexity increases.

- **Object-Oriented Approach**

- The fundamental idea behind object-oriented languages is to combine both, **data and the functions that operate on that data** into a single unit. Such a unit is called an **object**.
- Java., C#, Python, PHP, Type-Script are the examples of object-oriented languages.
-

- **Procedural Language Vs OOP**

Features	Procedural Oriented Programming (POP)	Object Oriented Programming (OOPS)
Divided into	In POP, program is divided into smaller parts called as functions.	in OOPs , the program is divided into parts known as **objects**.
Importance	In POP, importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOPs, Importance is given to the data rather than procedures or functions because it works as a **real world**.
Approach	POP follows **Top Down approach**.	OOPs follows **Bottom Up approach**.
Access Specifiers	POP does not have any access specifier.	OOPs has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOPs, objects can move and communicate with each other through member functions.
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOPs, data can not move easily from function to function,it can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper way for hiding data so it is **less secure**.	OOPs provides Data Hiding so provides **more security**.
Overloading	In POP, Overloading is not possible.	In OOPs, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	C, VB, FORTRAN, Pascal.	C++, JAVA, VB.NET, C#.NET.

- **Principles of Object-Oriented Languages**

- Class
- Objects
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism

1. Class:

A class is a user-defined data type. Collection of objects is called class. It is a logical entity. It consists of data members and member functions, which can be accessed and used by creating an instance of that class(Object). It represents the set of properties or methods that are common to all objects of one type. A class is like a blueprint for an object.

For Example: Consider the Class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc. So here, Car is the class, and wheels, speed limits, mileage are their properties.

2. Object:

It is a basic unit of Object-Oriented Programming and represents the real-life entities. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated. An object has an identity, state, and behavior. Each object contains data and code to manipulate the data.

For Example: "Dog" is a real-life Object, which has some characteristics like color, Breed, Bark, Sleep, and Eats.

3. Data Abstraction:

Data abstraction is one of the most essential and important features of object-oriented programming. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

For Example: A man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

4. Encapsulation:

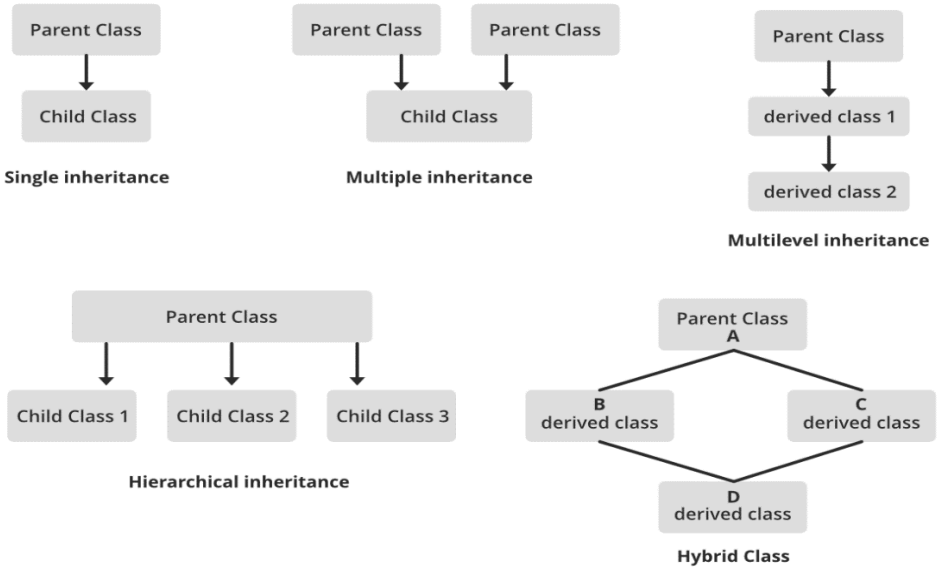
Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. In Encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of their class in which they are declared. As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.

For Example: In a company, there are different sections like the accounts section, finance section, sales section, etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps

records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name "sales section".

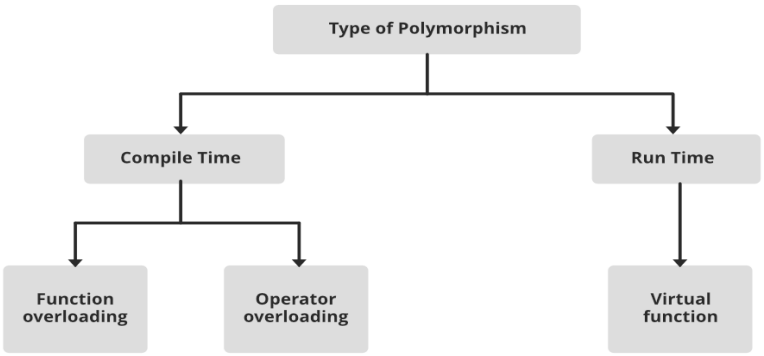
5. Inheritance:

Inheritance is an important pillar of OOP(Object-Oriented Programming). The capability of a class to derive properties and characteristics from another class is called Inheritance. When we write a class, we inherit properties from other classes. So when we create a class, we do not need to write all the properties and functions again and again, as these can be inherited from another class that possesses it. Inheritance allows the user to reuse the code whenever possible and reduce its redundancy.



6. Polymorphism:

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. For example, A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations. This is called polymorphism.



Advantage of OOPs over Procedure-oriented programming language

- OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.
- OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.
- OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

- **What is Java?**

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language. Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team. James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Applications

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

- Desktop Applications such as acrobat reader, media player, antivirus, etc.
- Web Applications such as irctc.co.in, javatpoint.com, etc.
- Enterprise Applications such as banking applications.
- Mobile
- Embedded System
- Smart Card
- Robotics
- Games, etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

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

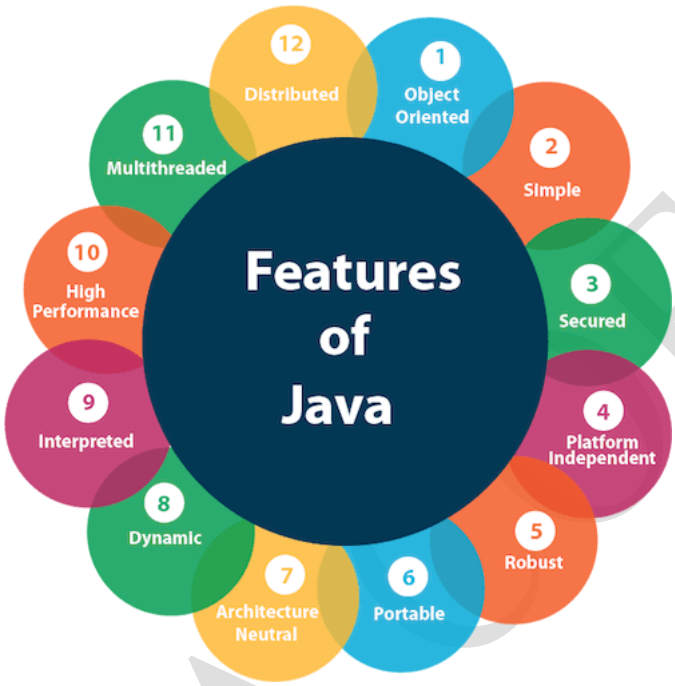
It is used to develop rich internet applications. It uses a lightweight user interface API.

Java Version History

- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan 1996)
- JDK 1.1 (19th Feb 1997)
- J2SE 1.2 (8th Dec 1998)
- J2SE 1.3 (8th May 2000)
- J2SE 1.4 (6th Feb 2002)
- J2SE 5.0 (30th Sep 2004)
- Java SE 6 (11th Dec 2006)
- Java SE 7 (28th July 2011)
- Java SE 8 (18th Mar 2014)
- Java SE 9 (21st Sep 2017)
- Java SE 10 (20th Mar 2018)
- Java SE 11 (September 2018)
- Java SE 12 (March 2019)
- Java SE 13 (September 2019)
- Java SE 14 (Mar 2020)
- Java SE 15 (September 2020)
- Java SE 16 (Mar 2021)
- Java SE 17 (September 2021)
- Java SE 18 (March 2022)
- Java SE 20 (March 2023)

- **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. Java is guaranteed to be Write Once, Run Anywhere.



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

1. **Simple:** Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master. Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:
   Java syntax is based on C++ (so easier for programmers to learn it after C++).
   Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
   There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.
2. **Object-Oriented:** In Java, everything is an Object. Java can be easily extended since it is based on the Object model. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behaviour.
3. **Portable:** Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.
4. **Platform independent:** when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
5. **Secured:** Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:
   - No explicit pointer

- Java Programs run inside a virtual machine sandbox

6. **Robust:** The English mining of Robust is strong. Java is robust because:
   - It uses strong memory management.
   - There is a lack of pointers that avoids security problems.
   - Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
   - There are exception handling and the type checking mechanism in Java. All these points make Java robust.
7. **Architecture neutral:** Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed. Also, Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
8. **Interpreted:** The Java source code first compiled into a binary byte code using Java compiler, then this byte code runs on the JVM (Java Virtual Machine), which is a software based interpreter. So Java is considered as both interpreted and compiled.
9. **High Performance:** Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. Java architecture is defined in such a way that it reduces overhead during the runtime and at sometimes 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. **Multithreaded:** A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.
11. **Distributed:**Java is distributed because it facilitates users to create distributed applications in Java. This feature of Java makes us able to access files by calling the methods from any machine on the internet.
12. **Dynamic:** 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 supports dynamic compilation and automatic memory management (garbage collection).

➢ JAVA ARCHITECTURE AND COMPONENTS

Java Architecture combines the process of compilation and interpretation.

- The code written in Java, is converted into byte codes which is done by the Java Compiler.
- The byte codes, then are converted into machine code by the JVM.
- The Machine code is executed directly by the machine.

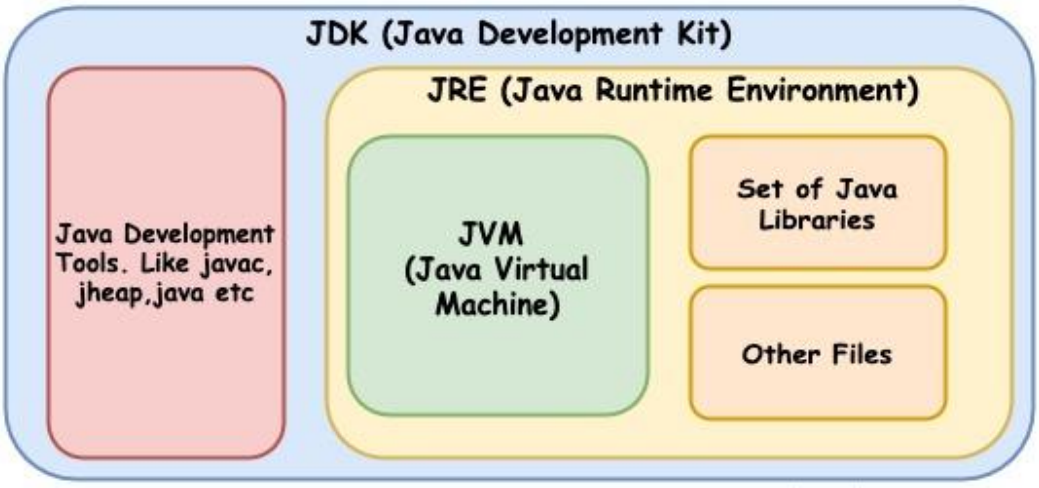
Components of Java Architecture There are three main components of Java language: JVM, JRE, and JDK. Java Virtual Machine, Java Runtime Environment and Java Development Kit respectively.

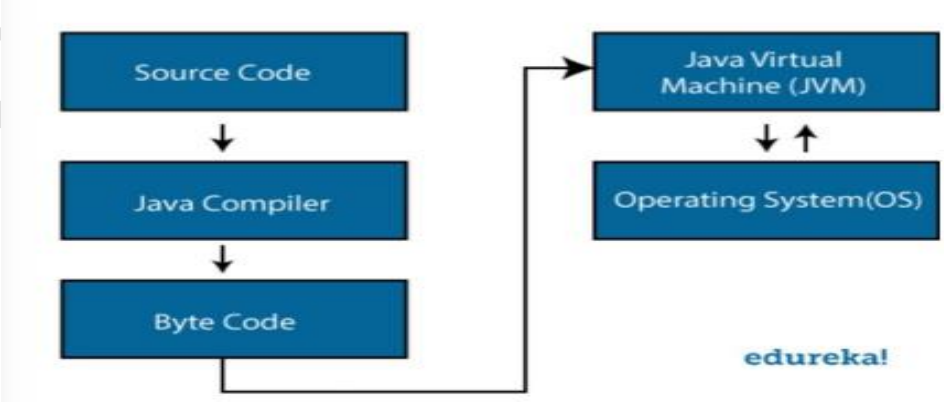
- **JDK (Java Development Kit):** The Java Development Kit (JDK) is one of three core technology packages used in Java programming, along with the JVM (Java Virtual Machine) and the JRE (Java Runtime Environment).

The JDK allows developers to create Java programs that can be executed and run by the JVM and JRE. JDK is an abbreviation for Java Development Kit. It is an environment of software development used for developing applets and Java applications. JDK has a physical existence, and it contains JRE + development tools. One can easily install more than one version of JDK on the same computer. The Java developers can make use of it on macOS, Windows, Linux, and Solaris. JDK assists them in coding and running the Java programs.

The JDK consists of a private JVM (Java Virtual Machine) along with a few other resources, java (a loader/interpreter), like javac (a compiler), Javadoc (a documentation generator), jar (an archiver), etc., for completing the process of Java application development.

- **JRE (Java Runtime Environment):** The JRE software builds a runtime environment in which Java programs can be executed. The JRE is the on-disk system that takes your Java code, combines it with the needed libraries, and starts the JVM to execute it. The JRE contains libraries and software needed by your Java programs to run. JRE is a part of JDK (which we will study later) but can be downloaded separately.
- **JVM (Java Virtual Machine):**It provides a runtime environment for driving Java applications or code. JVM is an abstract machine that converts the Java bytecode into a machine language. It is also capable of running the programs written by programmers in other languages (compiled to the Java bytecode). The JVM is also known as a virtual machine as it does not exist physically.





- **Structure of Java Programs :**

```
//Name of this file will be "Hello.java"
class Hello
```

```
{
   /* Writes the words "Hello Java" on the screen */
   public static void main(String[] args)
   {
      System.out.println("Hello Java");
   }
}
```

Output: Hello Java

- class Hello:

This creates a class called Hello.

All class names must start with a capital letter.

- /* Comments */:

The compiler ignores comment block. Comment can be used anywhere in the program to add info about the program or code block, which will be helpful for developers to understand the existing code in the future easily.

- Braces:

Two curly brackets {...} are used to group all the commands, so it is known that the commands belong to that class or method.

- public static void main:

1. When the main method is declared public, it means that it can also be used by code outside of its class, due to which the main method is declared public.
2. The word static used when we want to access a method without creating its object, as we call the main method, before creating any class objects.
3. The word void indicates that a method does not return a value. main() is declared as void because it does not return a value.
4. main is a method; this is a starting point of a Java program.

- String[] args:
  It is an array where each element of it is a string, which has been named as "args". If your Java program is run through the console, you can pass the input parameter, and main() method takes it as input.
- System.out.println();: This statement is used to print text on the screen as output, where the **system** is a predefined class, and **out** is an object of the PrintWriter class defined in the system. The method **println** prints the text on the screen with a new line. You can also use print() method instead of println() method. All Java statement ends with a semicolon.

➢ **Data types:**
Data types are the means for the tasks related to identifying and assessing the type of data. Data types specify the different sizes and values that can be stored in the variable. Java is rich in data types which allows the programmer to select the appropriate type needed to build variables of an application.

- **Why data types are important:**

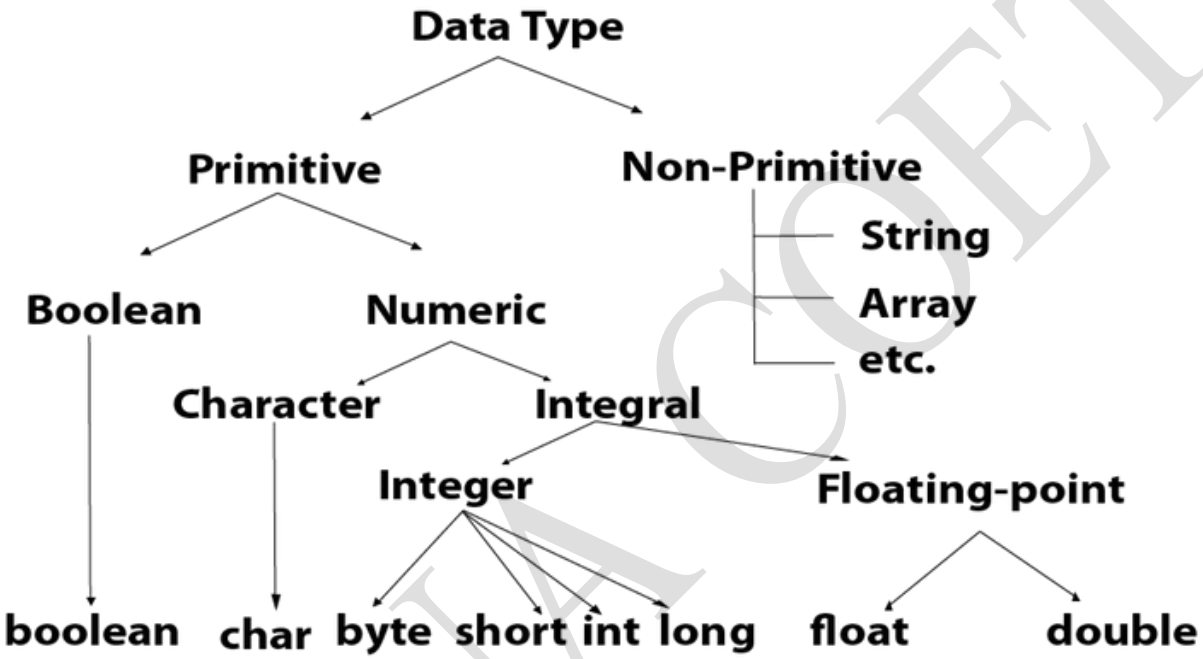
  - Every variable in Java has a data type which tells the compiler what type of variable it as and what type of data it is going to store.
  - Data type specifies the size and type of values.
  - Information is stored in computer memory with different data types.

- Whenever a variable is declared, it becomes necessary to define a data type that what will be the type of data that variable can hold.

There are **two types of data types** in Java:

Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double. They are also called as built-in data types.

Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays. They are also called as user-defined data types.



Integer Data Type:

Integer is the whole number without any fractional point. It can hold whole numbers such as 196, -52, 4036, etc. Java supports four different types of integers byte, short, int, long.

Floating Point Data Type:

It is used to hold whole numbers containing fractional part such as 36.74, or -23.95. There are two types of floating point storage in java ,float and double.

Character Data Type: It is used to store character constants in memory. Java provides a character data type called char whose type consumes a size of two bytes but can hold only a single character.

Boolean Data Type: Boolean type is used to test a particular condition during program execution. Boolean variables can take either *true* or *false* and is denoted by the keyword boolean and usually consumes one byte of storage.

NAME	DEFAULT	SIZE	TYPE	EXAMPLE
byte	0	8-bit	Integral	byte b = 100;
short	0	16-bit	Integral	short s = 10000;
int	0	32-bit	Integral	int i = 100000;
long	0L	64-bit	Integral	long l = 9999999;
float	0.0f	32-bit	Floating Point	float f = 123.4f;
double	0.0d	64-bit	Floating Point	double d = 12.4;
boolean	false	1-bit	Boolean	boolean b = true;
char	'\u0000'	16-bit	Character	char c = 'C';

- **Variables in Java**

Variable is a quantity whose value can be changed program execution. It is an identifier that denotes a storage location used to store a data value. A variable name may consists of alphabets, digits or underscore.

- **Declaration of Variables in Java**: Declaring a variable means where the data will store. Variables display named storage locations, whose values can be changed during the execution of the program. It is the basic unit of storage in a Java program.
- **Syntax:**

Datatype variable_name;

Datatype variable_name, variable_name, variable_name;

Here's the Data type specifies what type of data the variable will hold.

- Example:

```
int    width, height;
float  age, area;
double d;
```

- **Initialization of Variables in Java :** This means assigning a value to variables. In Java, you can assign a value to variables in two ways:
- Static - This means that the memory is determined for variables when the program starts.
- Dynamic - Dynamic means that in Java, you can declare variables anywhere in the program, because when the statement is executed the memory is assigned to them.
  - Example:

```
width = 10;
```

age = 26.5;

- **Rules of Declaring variables in Java:**

- A variable name can consist of Capital letters A-Z, lowercase letters a-z, digits 0-9, and two special characters such as underscore and dollar Sign.
- The first character must be a letter.
- Blank spaces cannot be used in variable names.
- Java keywords cannot be used as variable names.
- Variable names are case-sensitive.

- **Scope of Variables in Java:**
  Variable Scope means - That limit, as far as the variable can be used.
  In Java there are various types of variable scope:
- Local variables
- Instance variables
- Static variables
- A variable that is declared **within the method** that is called **local variables**. It is defined in method or other statements, such as defined and used within the cache block, and outside the block or method, the variable cannot be used.
- A non-static variable that is declared **within the class but not in the method** is called **instance variable**. Instance variables are related to a specific object; they can access class variables.
- A variable that is declared **with static keyword in a class** is called static or class variable. The static keyword belongs to the class than an instance of the class.

```
class A {
 int amount = 100; //instance variable
static int pin = 2315; //static variable
 public static void main(String[] args) {
 int age = 35; //local variable
}}
```

- **Tokens in java:**
  Java Tokens are the smallest individual building block or smallest unit of a Java program; the Java compiler uses it for constructing expressions and statements. Java program is a collection of different types of tokens, comments, and white spaces.
  There are various tokens used in Java:
- Reserved Keywords
- Identifiers
- Literals
- Operators
- Separators

  White space is also considered as a token.

- **Java operators:**

Java operators are symbols that are used to perform mathematical or logical manipulations. Java is rich with built-in operators. Operators are tokens that perform some calculations when they are applied to variables.

There are many types of operators available in Java such as:

- Arithmetic Operators
- Unary Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Compound Assignment Operators
- Conditional Operator
- instanceof Operator
- Member Selection or Dot Operator

- **Arithmetic operators**: They can be used to perform arithmetic operations by writing arithmetic expressions. The modulus operator returns the remainder of a division operation. In java modulo division operator % can be used for both integer and floating point arithmetic operations. Plus (+) operator can be used for addition as well as Concatenation of strings.

There are various arithmetic operators used in Java:

Operator	Meaning	Work
+	Addition	To add two operands.
-	Subtraction	To subtract two operands.
*	Multiplication	To multiply two operands.
	Division	To divide two operands.
%	Modulus	To get the area of the division of two operands

Write a program for arithmetic operators in java.

```
public class Main {
 public static void main(String[] args) {
  //Variables Definition and Initialization
  int number1 = 12, number2 = 4, sum ;
 //Addition Operation
sum = number1 + number2;
  System.out.println("Sum is: " + sum);
  //Subtraction Operation
  int dif = number1 - number2;
  System.out.println("Difference is : " + dif);
 //Multiplication Operation
  int mul = number1 * number2;
  System.out.println("Multiplied value is : " + mul);
 //Division Operation
  int div = number1 / number2;
  System.out.println("Quotient is : " + div);
```

```
//Modulus Operation
  int rem = number1 % number2;
  System.out.println("Remainder is : " + rem);
 }
}
```

Output: Sum is: 16
Difference is : 8
Multiplied value is : 48
Quotient is : 3
Remainder is : 0

- **Unary arithmetic operators**: Used to increasing or decreasing the value of an operand. Increment operator adds 1 to the value of a variable, whereas the decrement operator decreases a value.
  Increment and decrement unary operator works as follows:
  Syntax:
  Val++; ++Val;
  Val--; --Val;
  These two operators have two forms: Postfix and Prefix. Both do increment or decrement in appropriate variables. These two operators can be placed before or after of variables. When it is placed before the variable, it is called prefix. And when it is placed after, it is called postfix.

Example	Description
val = a++;	Store the value of "a" in "val" then increments.
val = a--;	Store the value of "a" in "val" then decrements.
val = ++a;	Increments "a" then store the new value of "a" in "val".
val = --a;	Decrements "a" then store the new value of "a" in "val".

- **Relational operators:** Compare between operands and determine the relationship between them. These operators are mainly used when applying control statements in the program.
  There are six types of relational operators in Java, these are:

Operator	Meaning
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

- **Logical Operators**: Work on the Boolean operand. It's also called Boolean logical operators. It operates on two Boolean values, which return Boolean values as a result.

	Operator	Meaning	Work
•	&&	Logical AND	If both operands are true then only "logical AND operator“ evaluate true.

•	||	Logical OR	The logical OR operator is only evaluated as true when one of its operands evaluates true. If either or both expressions to true, then the result is true.
•	!	Logical Not	Logical NOT is a Unary Operator, it operates on single operands. It reverses the value of operands, if the value is true, then it gives false, and if it is false, then it gives true.

Example:

```
public class Main
{ public static void main(String[] args) {
System. out . println((5>3) & (6>5));
System. out. println((5>3) ol (6>5));
 System. out. println(!(5>3));} }
```

Output:

True

True

False

- **Bitwise Operators:** Allow access and modification of a particular bit inside a section of the data. It can be applied to integer types and bytes, and cannot be applied to float and double.

Operator	**Meaning**	**Work**
&	Binary AND Operator	There are two types of AND operators in Java: the logical && and the binary &. Binary & operator work very much the same as logical && operators works, except it works with two bits instead of two expressions. The "Binary AND operator" returns 1 if both operands are equal to 1.
|	Binary OR Operator	"AND operators ", Java has two different "OR" operators: the logical || and the binary |. Binary | Operator work similar to logical || operators works, except it, works with two bits instead of two expressions. The "Binary OR

		operator" returns 1 if one of its operands evaluates as 1. if either or both operands evaluate to 1, the result is 1.
^	Binary XOR Operator	It stands for "exclusive OR" and means "one or the other", but not both. The "Binary XOR operator" returns 1 if and only if exactly one of its operands is 1. If both operands are 1, or both are 0, then the result is 0.

There are bitwise shift operator in java listed below:

- ~ Binary Complement Operator
- << Binary Left Shift Operator
- >> Binary Right Shift Operator
- >>> Shift right zero fill operator

Example:

```
public class Main{

public static void main(sString[] args) {

int a=6;

int b=5;

int result= a|b;

System. out. println(”OR value is " +result);

}}
```

Output:

OR value is 7

Example:

```
public class Main
{public static void main(String[] args) {
int a=12;
int 1shift= a<<2;
System. out. println(”left shift value is"+lshift);
 } }
```

Output:

left shift value is 48

- **Assignment Operators:** Are used when you want to assign a value to the expression. The assignment operator denoted by the single equal sign =.
  In a Java assignment statement, any expression can be on the right side and the left side must be a variable name. For example, this does not mean that "a" is equal to "b", instead, it means assigning the value of 'b' to 'a'. It is as follows:
- **Syntax:**
  variable = expression;

- **Example:**
  int a = 6;
  +float b = 6.8F

- **Compound Assignment Operators**: Also known as **Shorthand Assignment Operators**. It's called shorthand because it provides a short way to assign an expression to a variable.

  This operator can be used to connect Arithmetic operator with an Assignment operator. For example, you write a statement:
  a = a+6;
  In Java, you can also write the above statement like this:
  a += 6;

  There are various compound assignment operators used in Java:

Operator	**Meaning**
+=	Increments then assigns
-=	Decrements then assigns
*=	Multiplies then assigns
/=	Divides then assigns
%=	Modulus then assigns
<<=	Binary Left Shift and assigns
>>=	Binary Right Shift and assigns
>>>=	Shift right zero fill and assigns
&=	Binary AND assigns
^=	Binary exclusive OR and assigns
|=	Binary inclusive OR and assigns

➢ **Keywords** ;

- They are words that have already been defined for Java compiler. They have special meaning for the compiler. Java Keywords must be in your information because you can not use them as a variable, class or a method name
- You can't use keyword as identifier in your Java programs, its reserved words in Java library and used to perform an internal operation.
  true, ***false*** and ***null*** are not reserved words but cannot be used as identifiers, because it is literals of built-in types.

List of keywords in java:

1. abstract	13. double	25. int	37. strictfp
2. assert	14. else	26. interface	38. super
3. boolean	15. enum	27. long	39. switch
4. break	16. extends	28. native	40. synchronized
5. byte	17. final	29. new	41. this
6. case	18. finally	30. package	42. throw
7. catch	19. float	31. private	43. throws
8. char	20. for	32. protected	44. transient
9. class	21. if	33. public	45. try
10. continue	22. implements	34. return	46. void
11. default	23. import	35. short	47. volatile
12. do	24. instanceof	36. static	48. while

- **Identifier**
  Java Identifiers are the user-defined names of variables, methods, classes, arrays, packages, and interfaces. Once you assign an identifier in the Java program, you can use it to refer the value associated with that identifier in later statements. There are some standards which you must follow while naming the identifiers such as:
  - Identifiers must begin with a letter, dollar sign or underscore.
  - Apart from the first character, an identifier can have any combination of characters.
  - Identifiers in Java are case sensitive.
  - Java Identifiers can be of any length.
  - Identifier name cannot contain white spaces.
  - Any identifier name must not begin with a digit but can contain digits within.
  - Most importantly, keywords can't be used as identifiers in Java.

  Examples:

  Valid identifiers:_ testvariable $testvariable sum_of_array

  Invalid identifiers: Test Variable 123javatpoint

- **Literals**
  Literals in Java are similar to normal variables but their values cannot be changed once assigned. In other words, literals are constant variables with fixed values. These are defined by users and can belong to any data type. Java supports five types of literals which are as follows:
  - Integer
  - Floating Point
  - Character
  - String
  - Boolean

- **Operator Precedence:**

  Operators in java have its own priority. It is essential to understand the priority to write the correct expressions. Parentheses can change the priority of the operators inside them. Precedence of the java operators are mentioned below. Parentheses(Bracket operation) has highest priority.

Highest						
++ (postfix)	-- (postfix)					
++ (prefix)	-- (prefix)	~	!	+ (unary)	- (unary)	(type-cast)
*	/	%				
+	-					
>>	>>>	<<				
>	>=	<	<=	instanceof		
==	!=					
&						
^						
|						
&&						
||						
?:						
=	op= (compound assignment)					
Lowest						

➢ Java Type Casting

- The process of converting the value of one data type (int, float, double, etc.) to another data type is known as typecasting.
- In Java, there are types of type conversion.
  1. Widening Type Casting
  2. Narrowing Type Casting

• Widening Type Casting:

In **Widening Type Casting**, Java automatically converts one data type to another data type. In the case of **Widening Type Casting**, the lower data type (having smaller size) is converted into the higher data type (having larger size). Hence there is no loss in data. This is why this type of conversion happens automatically. This is also known as **Implicit Type Casting**

Example: Converting int to double

```
class Main {
  public static void main(String[] args) {
    int num = 10;
    System.out.println("The integer value: " + num);
    double data = num;
    System.out.println("The double value: " + data);
  }}
```

Output:

The integer value: 10 The double value: 10.0

• Narrowing Type Casting

In **Narrowing Type Casting**, we manually convert one data type into another using the parenthesis.In the case of **Narrowing Type Casting**, the higher data types (having larger size) are converted into lower data types (having smaller size). Hence there is the loss of data. This is why this type of conversion does not happen automatically.

Note: This is also known as **Explicit Type Casting**.

Example: Converting double into an int

```
class Main {
  public static void main(String[] args) {
       double num = 10.99;
    System.out.println("The double value: " + num);
     int data = (int)num; //
    System.out.println("The integer value: " + data);
  }
}
```

Output:

The double value: 10.99 The integer value: 10

• Program for Type conversion from int to String

```
class Main {
  public static void main(String[] args) {
   int num = 10;
    System.out.println("The integer value is: " + num);
    String data = String.valueOf(num);
    System.out.println("The string value is: " + data);
  }
}
```

Output:
The integer value is: 10
The string value is: 10
String data = String.valueOf(num);
Here, we have used the valueOf() method of the java string class to convert the int type variable into a string.

- **Type conversion from String to int**

```
class Main {
  public static void main(String[] args) {

    String data = "10";
    System.out.println("The string value is: " + data);
    int num = Integer.parseInt(data);
    System.out.println("The integer value is: " + num);
  }
}
```

Output
The string value is: 10
The integer value is: 10
Here, we have used the parseInt() method of the Java Integer class to convert a string type variable into an int variable.
Note: If the string variable cannot be converted into the integer variable then an exception named NumberFormatException occurs.

➢ Java Control statement

- A programming language uses control statements to cause the flow of execution to advance and branch based on changes to the state of a program.
- Java's program control statements can be put into the following categories: **selection, iteration, and jump.**
- Selection statements allow your program to choose different paths of execution based upon the outcome of an expression or the state of a variable.
- Iteration statements enable program execution to repeat one or more statements (that is, iteration statements form loops).
- Jump statements allow your program to execute in a nonlinear fashion.

- **Java's Selection Statements**
  Java supports two selection statements: if and switch. These statements allow us to control the flow of our program's execution based upon conditions known only during run time.

- **if statement:**
  The if statement is Java's conditional branch statement. It can be used to route program execution through two different paths. Here is the general form of the if statement:

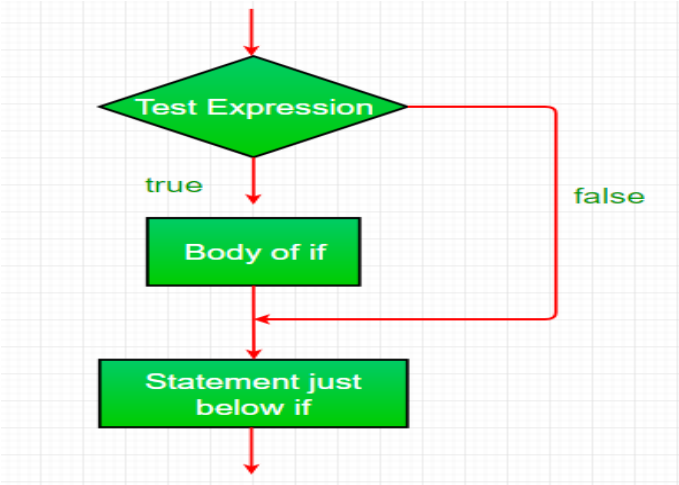
Syntax:

```
if(condition)
{
 // Statements to execute if
```

// condition is true
}
statement2;

If the condition is true, then statement1 is executed. Otherwise, statement2 (if it exists) is executed.

Flowchart:



Program:

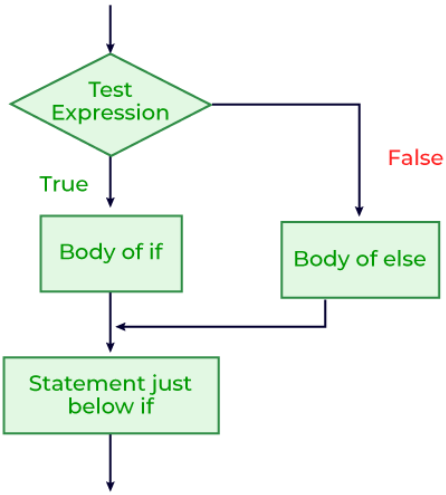
```
class If {
 public static void main(String args[])
{
   int i = 10;
  if (i < 15){
 System.out.println("10 is less than 15");}
        System.out.println("Outside if-block");
        }}
```

Output:

```
10 is less than 15
Outside if-block
```

- **if-else statement**: if condition in Java can be used to execute a block of code only when a condition is true, if the condition is false we can execute a different block of code. When we use if-else, it works like, if the condition is true then, statement1 is executed. If it is false, statement2 is executed.
- **Syntax:**

if(condition)
{statement1;}
else
{statement2;}



Program:

```
class PrintPassFail
{public static void main(String arg[])
   {int marks = 62;
      if(marks > 35)                {
         System.out.println("Pass");
      }
      else
      {
         System.out.println("Fail");
      }}
}
```

Output: Pass

- **Java if-else-if ladder Statement:** The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed. The final else acts as a default condition; that is, if all other conditional tests fail, then the last else statement is performed. If there is no final else and all other conditions are false, then no action will take place.
- **Syntax:**

```
if (condition)
 statement 1;
else if (condition)
statement 2;
.
.
else
statement;
```

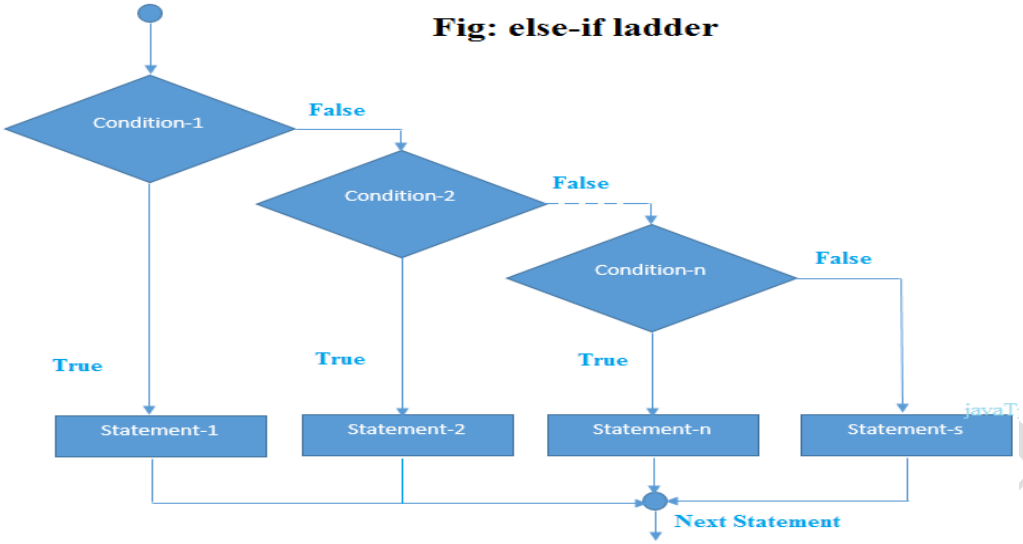


Fig: else-if ladder

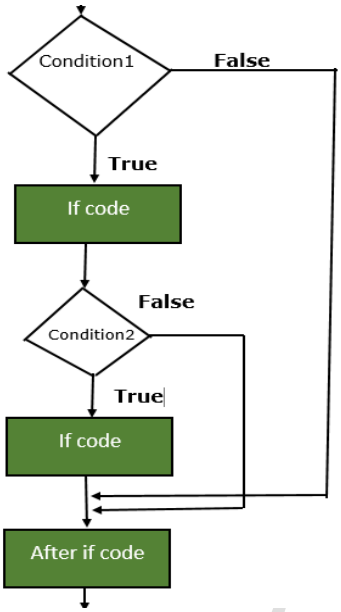
Program:

```
class Pn {
public static void main(String[] args) {
int number=-13;
 if(number>0){
 System.out.println("POSITIVE");
}else if(number<0){
        System.out.println("NEGATIVE");
        }else{
        System.out.println("ZERO");
         }  }
        }
```

Output: NEGATIVE

- **Nested if statement:** The nested if statement represents the if block within another if block. Here, the inner if block condition executes only when outer if block condition is true.
- **Syntax:**

```
if(condition){
  //code to be executed
      if(condition){
        //code to be executed
 }
  }
```



Program:

```
public class NestedIf {
public static void main(String[] args) {
   int age=20;
   int weight=80;
      if(age>=18){
      if(weight>50){
         System.out.println("You are eligible to donate blood");
      }
   }
}}
```

Output: You are eligible to donate blood

• **Switch Statement:** The switch statement is Java's multiway branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression. As such, it often provides a better alternative than a large series of if-else-if statements.

• **Syntax:**

```
switch(expression){
case value1:    //code to be executed;
 break;  //optional
case value2:     //code to be executed;
 break;  //optional
.....
default:  code to be executed if all cases are not matched;
}
```

Points to Remember:

- There can be one or N number of case values for a switch expression.
- The case value must be of switch expression type only. The case value must be literal or constant. It doesn't allow variables.

- The case values must be unique. In case of duplicate value, it renders compile-time error.
- The Java switch expression must be of byte, short, int, long (with its Wrapper type), enums and string.
- Each case statement can have a break statement which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
- The case value can have a default label which is optional.

```
class SwitchExample {
public static void main(String[] args) {
    int number=20;
    switch(number){
    case 10: System.out.println("10");
    break;
    case 20: System.out.println("20");
    break;
    case 30: System.out.println("30");
    break;
     default:System.out.println("Not in 10,
20 or 30");
    }
}
}
```

Output:20

```
public class SwitchVowelExample {
public static void main(String[] args) {
    char ch='i';
    switch(ch)    {
      case 'a':System.out.println("Vowel");
           break;
      case 'e': System.out.println("Vowel");
           break;
      case 'i': System.out.println("Vowel");
           break;
      case 'o': System.out.println("Vowel");
           break;
      case 'u': System.out.println("Vowel");
           break;
           default:
           System.out.println("Consonant");
    }  }    }
```

Output: Vowel

- **Iteration Statements** Java's iteration statements are for, while, and do-while. These statements create what we commonly call loops. As you probably know, a loop repeatedly executes the same set of instructions until a termination condition is met.

- **For Loop statement:** When you know exactly how many times you want to loop through a block of code, use the for loop.
- **Syntax:**

  for(initialization;condition;incr/decr){

  //statement or code to be executed }

- **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.

- **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
- **Statement**: The statement of the loop is executed each time until the second condition is false.
- **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.
- **Program**:

```
public class For{
public static void main(String[] args) {
  for(int i=1;i<=10;i++){
  System.out.println(i);
 } }
}
```

Output: 1 2 3 4 5 6 7 8 9 10

- **Nested For Loop:**If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.
- Example:

```
public class PyramidExample {
 public static void main(String[] args) {
 for(int i=1;i<=5;i++){
 for(int j=1;j<=i;j++){
        System.out.print("* ");
 }
 System.out.println();//new line
 }  } }
```

- **Output:**

```
*
* *
* * *
* * * *
* * * * *
```

- **While Loop:** In Java, While loop is a control statement. It is used for iterating a part of the program several times. When the number of iteration is not fixed then while loop is used.
- Syntax: while(condition)

```
{  //code for execution;

}
```

- **Program:**

```
public class WhileDemo1
```

```
{
        public static void main(String[] args)
        {
        int i=1;
        while(i<=10)
        {
                System.out.print(i);
                i++;
                } } }
```

- **Output:** 1 2 3 4 5 6 7 8 9 10

- **do-while:** loop always executes its body at least once, because its conditional expression is at the bottom of the loop. Each iteration of the do-while loop first executes the body of the loop and then evaluates the conditional expression. If this expression is true, the loop will repeat. Otherwise, the loop terminates. As with all of Java's loops, condition must be a Boolean expression.
- **Syntax:**

```
do
{  //code for execution
}
while(condition);
```

- **Program:**

```
public class DoWhileDemo1
{
        public static void main(String[] args) {
        int i=1;
        do{
        System.out.print(i);
                i++;
                }while(i<=10);
        }
}
```

- **Output:** 1 2 3 4 5 6 7 8 9 10
- **Jump Statements**
  Java supports different jump statements. break, continue are two of them.
- **Break:**In Java, the break statement has three uses.
  1. To terminate a statement sequence in a switch statement.
  2. It can be used to exit a loop.
  3. It can be used as a "civilized" form of goto.

- Using break to Exit a Loop
  By using break, you can force immediate termination of a loop, bypassing the conditional expression and any remaining code in the body of the loop. When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.
  The break statement can be used with any of Java's loops, including intentionally infinite loops.
  The break statement in the inner loop only causes termination of that loop. The outer loop is unaffected.
- **Program:**

```
// break to exit a loop
class Break
```

```
{  public static void main(String args[]) {
      // Initially loop is set to run from 0-9
      for (int i = 0; i < 10; i++)  {
         // terminate loop when i is 5.
         if (i == 5)
            break;
         System.out.println("i: " + i); }
      System.out.println("Loop complete.");
   }}
```

Output:

```
i: 0
i: 1
i: 2
i: 3
i: 4
Loop complete.
```

- **Continue:** Sometimes it is useful to force an early iteration of a loop. That is, you might want to continue running the loop but stop processing the remainder of the code in its body for this particular iteration. This is, in effect, a goto just past the body of the loop, to the loop's end. The continue statement performs such an action.
- **Program:**

```
class ContinueDemo
{ public static void main(String args[])  {
      for (int i = 0; i < 10; i++) {
         if (i%2 == 0)
            continue;
         System.out.print(i + " ");
      } }}
```

- **Output:** 1 3 5 7 9

- **Return:** The return statement is used to explicitly return from a method. That is, it causes a program control to transfer back to the caller of the method.
- **Program:**

```
            class Return {
               public static void main(String args[])  {
         boolean t = true;
          System.out.println("Before the return.");
          if (t)
         return;
         System.out.println("This won't execute.");
           }
         }
```

- **Output:**

```
Before the return.
```