Token:

A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:

1. Keywords
2. Identifiers
3. Constants
4. Special Symbols
5. Operators

1.Keyword: Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed.
Java language supports following keywords:

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

2.Identifiers: Identifiers are used as the general terminology for naming of variables, functions and arrays. These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore(_) as a first character. Identifier names must differ in spelling and case from any keywords. You cannot use keywords as identifiers; they are reserved for special use. Once declared, you can use the identifier in later program statements to refer to the associated value. A special kind of identifier, called a statement label, can be used in goto statements.

Examples of valid identifiers :

- Identifiers have from 1 to many characters:
  - 'a'..'z', 'A'..'Z', 'o'..'9', '_', $
  - Identifiers start with letter a1 is legal, 1a is not
- can also start with underscore or dollar sign: _ $
  - Java is case sensitive. A and a are different.

Examples of invalid identifiers :

1. My Variable // contains a space

2. 123geeks // Begins with a digit

3. a+c // plus sign is not an alphanumeric character

4. variable-2 // hyphen is not an alphanumeric character

5. sum_&_difference // ampersand is not an alphanumeric character

3.Constants/Literals: Constants are also like normal variables. But, the only difference is, their values can not be modified by the program once they are defined. Constants refer to fixed values. They are also called as literals.
Constants may belong to any of the data type.

Syntax:

final data_type variable_name;

4.**Special Symbols:** The following special symbols are used in Java having some special meaning and thus, cannot be used for some other purpose.

Ex: [] () {}, ; * =

- **Brackets[]:** Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.
- **Parentheses():** These special symbols are used to indicate function calls and function parameters.
- **Braces{}:** These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.
- **comma (,):** It is used to separate more than one statements like for separating parameters in function calls.
- **semi colon :** It is an operator that essentially invokes something called an initialization list.
- **asterick (*):** It is used to create pointer variable.
- **assignment operator:** It is used to assign values.

5.Operators: Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are-

1. Arithmetic Operators
2. Unary Operators
3. Assignment Operator
4. Relational Operators
5. Logical Operators
6. Ternary Operator
7. Bitwise Operators
8. Shift Operators
9. instance of operator
10. Precedence and Associativity

Statement:

Statements are roughly equivalent to sentences in natural languages.
A *statement* forms a complete unit of execution. The following types of expressions can be made into a statement by terminating the expression with a semicolon (;).

Statements in Java can be broadly classified into three categories:

- Declaration statement
- Expression statement
- Control flow statement

1. Declaration Statement:

A declaration statement is used **to declare a variable**. For example,

```
int num;
int num2 = 100;
String str;
```

2.Expression Statement

An **expression with a semicolon at the end** is called an expression statement. For example,

```
/Increment and decrement expressions
num++;
++num;
num--;
--num;

//Assignment expressions
num = 100;
num *= 10;
```

```
//Method invocation expressions
System.out.println("This is a statement");
someMethod(param1, param2);
```

3.Control Flow Statement

By default, all statements in a Java program are executed in the order they appear in the program. Sometimes you may want to execute a set of statements repeatedly for a number of times or as long as a particular condition is true.

All of these are possible in Java using control flow statements. If block, while loop and for loop statements are examples of control flow statements.

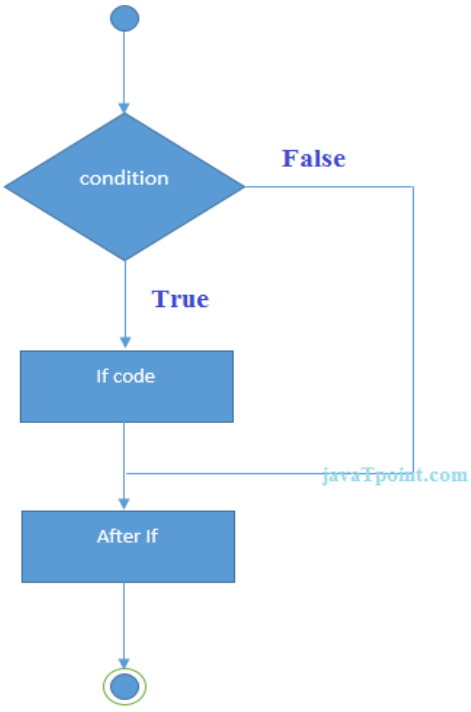
if Statement

The Java if statement tests the condition. It executes the *if block* if condition is true.

Syntax:

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

Flowchart:



Example:

```
//Java Program to demonstate the use of if statement.
public class IfExample
 {
public static void main(String[] args)
 {
     int age=20;    //defining an 'age' variable
   //checking the age
    if(age>18)
      {
        System.out.print("Age is greater than 18");
      }
 }
}
```

Output:

Age is greater than 18

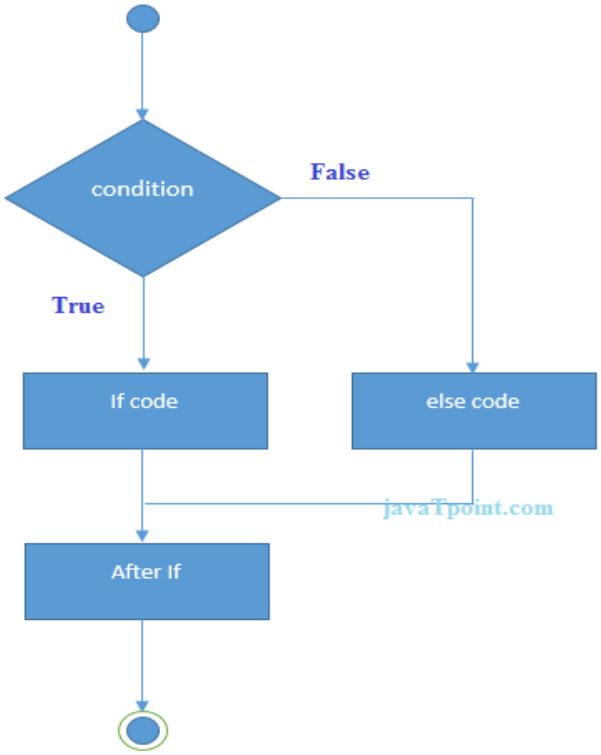
if-else Statement

The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

Syntax:

```
1.          if(condition){
2.          //code if condition is true
3.}        else{
4./          /code if condition is false
      }
```

Flowchart:



Example:

```
1.      //A Java Program to demonstrate the use of if-else statement.
2./       /It is a program of odd and even number.
   public class IfElseExample
      {
     public static void main(String[] args) {
3.         //defining a variable
4.           int number=13;
5.         //Check if the number is divisible by 2 or not
6.           if(number%2==0){
7.             System.out.println("even number");
8.           }
9.         else
10.                  {
11.                   System.out.println("odd number");
12.                 }
       }
```

Output:

odd number

3.Java if-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
1.  if(condition1){
    //code to be executed if condition1 is true
    }
    else if(condition2){
2.    //code to be executed if condition2 is true
    }
    else if(condition3){
3.    //code to be executed if condition3 is true
     }
4.   ...
     else{
5.    //code to be executed if all the conditions are false
6.   }
```

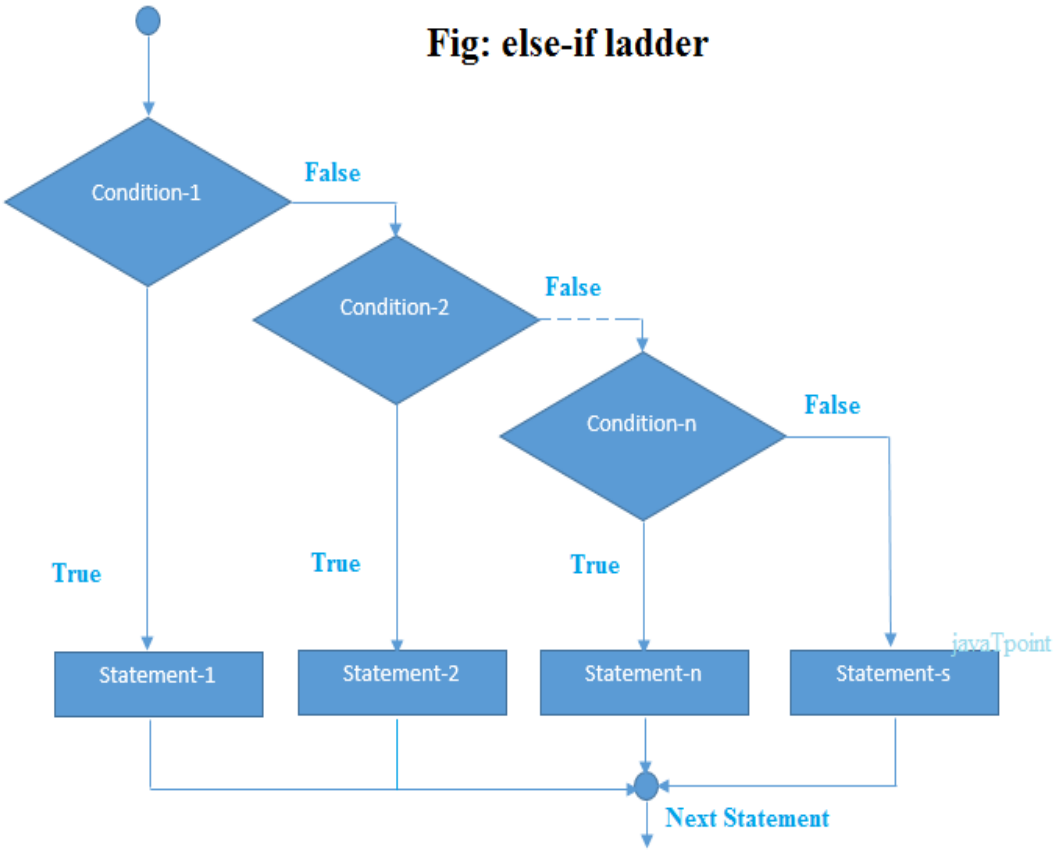


Fig: else-if ladder

Example:

```
1.      / /Java Program to demonstrate the use of If else-if ladder.
        //It is a program of grading system for fail, D grade, C grade, B grade, A grade and A+.
        public class IfElseIfExample {
        public static void main(String[] args) {
2.          int marks=65;
3.          if(marks<50){
4.              System.out.println("fail");
5.          }
6.          else if(marks>=50 && marks<60){
7.              System.out.println("D grade");
8.          }
9.          else if(marks>=60 && marks<70){
10.                 System.out.println("C grade");
11.             }
12.             else if(marks>=70 && marks<80){
13.                 System.out.println("B grade");
14.             }
15.             else if(marks>=80 && marks<90){
```

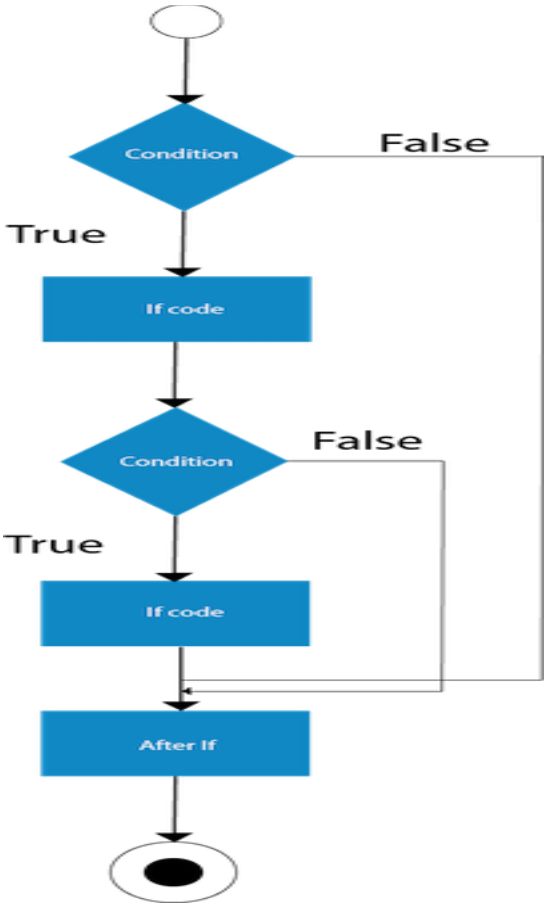
```
16.                     System.out.println("A grade");
17.                 }
18.                 else if(marks>=90 && marks<100){
19.                     System.out.println("A+ grade");
20.                  }
21.                 else{
22.                     System.out.println("Invalid!");
23.                     }
24.         }
25.                 Output:
                  NEGATIVE
```

4.Java 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:

```
1.          if(condition){
2.            //code to be executed
3.             if(condition){
4.            //code to be executed
5.               }
        }
```



Example:

```
1.          //Java Program to demonstrate the use of Nested If Statement.
    public class JavaNestedIfExample {
    public static void main(String[] args) {
2.      //Creating two variables for age and weight
3.      int age=20;
4.      int weight=80;
5.      //applying condition on age and weight
6.      if(age>=18){
7.      if(weight>50){
8.        System.out.println("You are eligible to donate blood");
9.      }
10.          }
11.     } }
```

5.Switch Statement

The Java *switch statement* executes one statement from multiple conditions.

It is like if-else-if ladder statement. The

switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long . Since Java 7, you can use strings in the switch statement.

In other words, the switch statement tests the equality of a variable against multiple values.

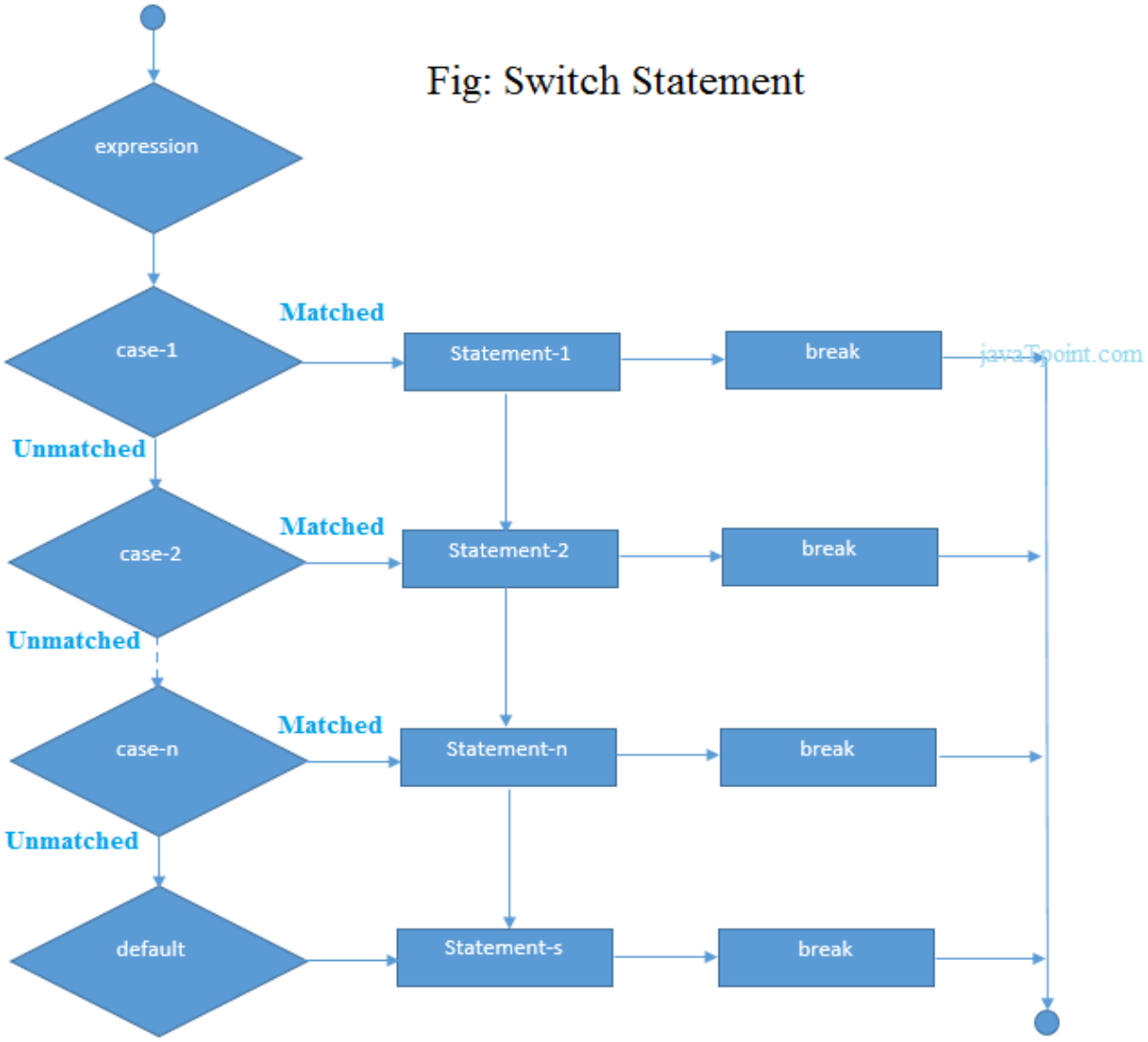
Points to Remember

- 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.

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;
}
```

Fig: Switch Statement



12. }

Example:

```
public class SwitchExample {
public static void main(String[] args) {
1.      //Declaring a variable for switch expression
2.          int number=20;
3.         //Switch expression
4.          switch(number){
5.         //Case statements
6.         case 10: System.out.println("10");
7.          break;
8.        case 20: System.out.println("20");
9.          break;
10.             case 30: System.out.println("30");
11.               break;
```

```
12.                 //Default case statement
13.                 default:System.out.println("Not in 10, 20 or 30");
14.                 }
15.             }
16.     }
17.                 Output:
        20
```

Loops in Java

In programming languages, loops are used to execute a set of instructions/functions repeatedly when some

conditions become true. There are three types of loops in Java.

- for loop
- while loop
- do-while loop

For Loop

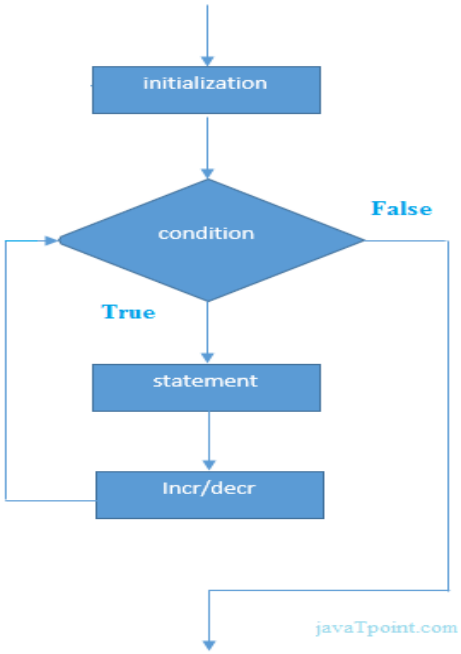
A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement

value. It consists of four parts:

1.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.
2.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.
3.Statement: The statement of the loop is executed each time until the second condition is false.
4.Increment/Decrement: It increments or decrements the variable value. It is an optional condition.

Syntax:

```
1.          for(initialization;condition;incr/decr){
2.              //statement or code to be executed
3.              }
4.
5.              Flowchart:
```



6.

12.

26.
27.
28.
29. }

Example:

```
//Java Program to demonstrate the example of for loop
//which prints table of 1
public class ForExample {
public static void main(String[] args) {
1.      //Code of Java for loop
2.     for(int i=1;i<=5;i++){
3.     System.out.println(i);
4.      }
5.     }
}
```

Output:

```
1
2
3
4
        5
```

While Loop

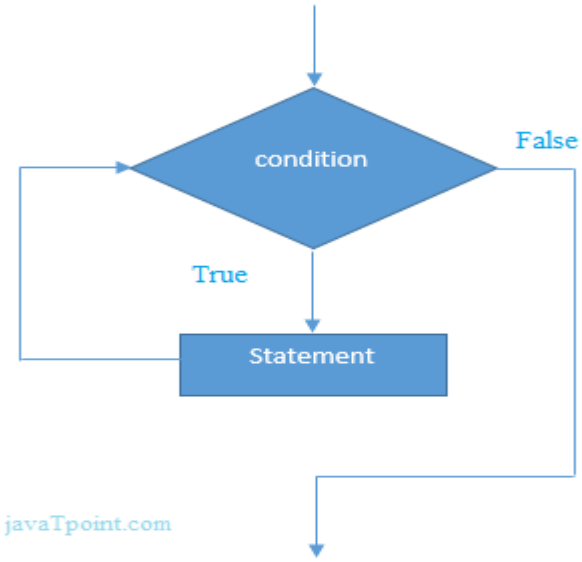
The Java *while loop* is used to iterate a part of the program several times. If the number of iteratio is not

fixed, it is recommended to use while loop.

Syntax:

```
while(condition){
1.      //code to be executed
2.      }
```

Flowchart:



Example:

```
public class WhileExample {
public static void main(String[] args) {
   int i=1;
   while(i<=5){
      System.out.println(i);
   i++;
   }
}
}
```

Output:

```
1
2
3
4
5
```

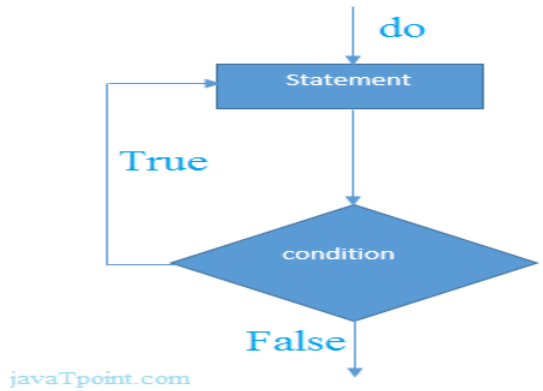
do-while Loop

The Java *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The Java *do-while loop* is executed at least once because condition is checked after loop body.

Syntax:

```
do{
//code to be executed
}while(condition);
```



Break Statement

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

The Java *break* statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:

jump-statement;

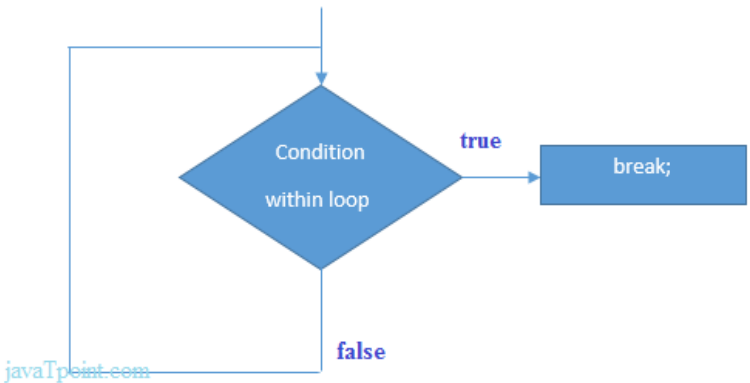


Figure: Flowchart of break statement

break;

Break Statement with Loop

Example:

```
//Java Program to demonstrate the use of break statement
//inside the for loop.
public class BreakExample {
public static void main(String[] args) {
   //using for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //breaking the loop
         break;
      }
      System.out.println(i);
   }
 }
}
```

Output:

```
1
2
3
4
```

Continue Statement

The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.

The Java *continue statement* is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.

We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:

```
jump-statement;
continue;
```

Java Continue Statement Example

Example:

```
//Java Program to demonstrate the use of continue statement
```

```
//inside the for loop.
public class ContinueExample {
public static void main(String[] args) {
   //for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //using continue statement
         continue;//it will skip the rest statement
      }
      System.out.println(i);
   }
}
}
```

Output:

```
1
2
3
4
6
7
8
9
10
```

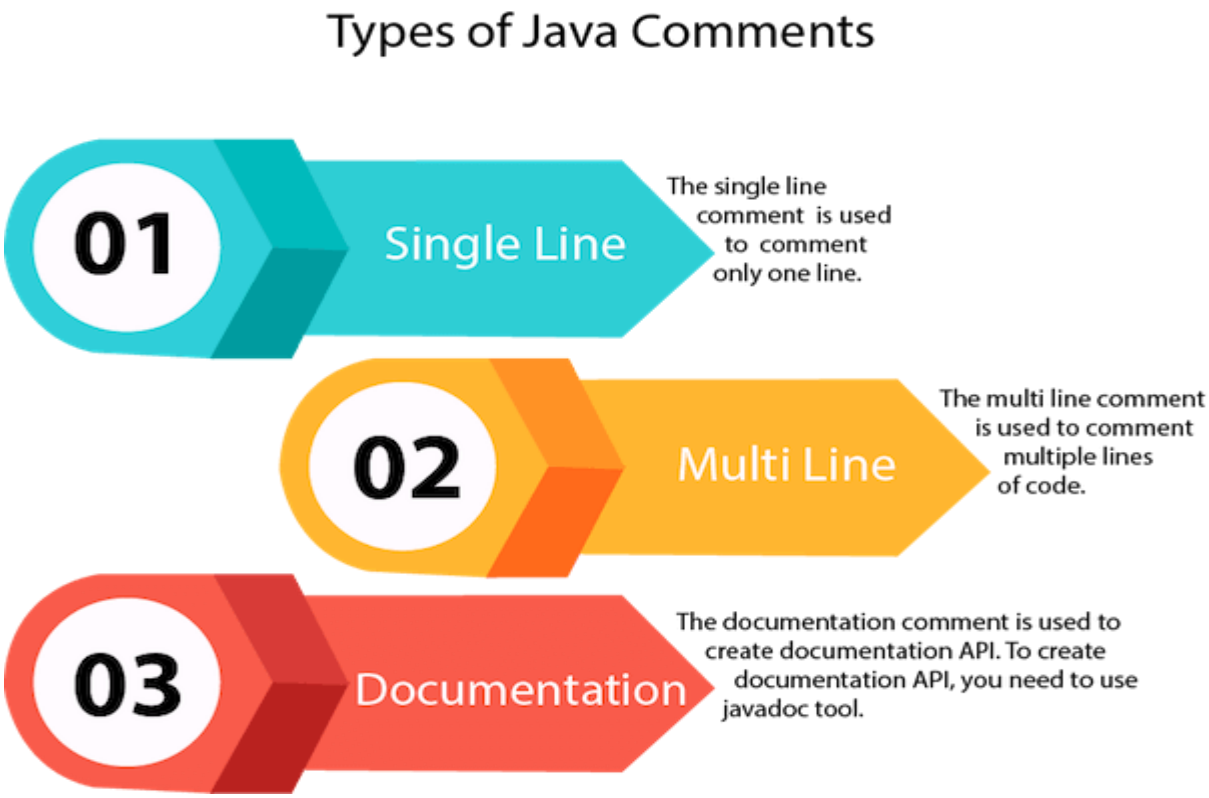
Java Comments

The Java comments are the statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code.

Types of Java Comments

There are three types of comments in Kava.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment



1.Java Single Line Comment

The single line comment is used to comment only one line.

Syntax: //This is single line comment

2) Java Multi Line Comment

The multi line comment is used to comment multiple lines of code.

Syntax:

```
/*
This
is
multi line
comment
*/
```

3) Java Documentation Comment

The documentation comment is used to create documentation API. To create documentation API, you need to use **javadoc tool**.

Syntax:

```
/**
This
is
documentation
comment
*/
```

JVM (Java Virtual Machine) Architecture

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

- **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
- **An implementation** Its implementation is known as JRE (Java Runtime Environment).
- **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

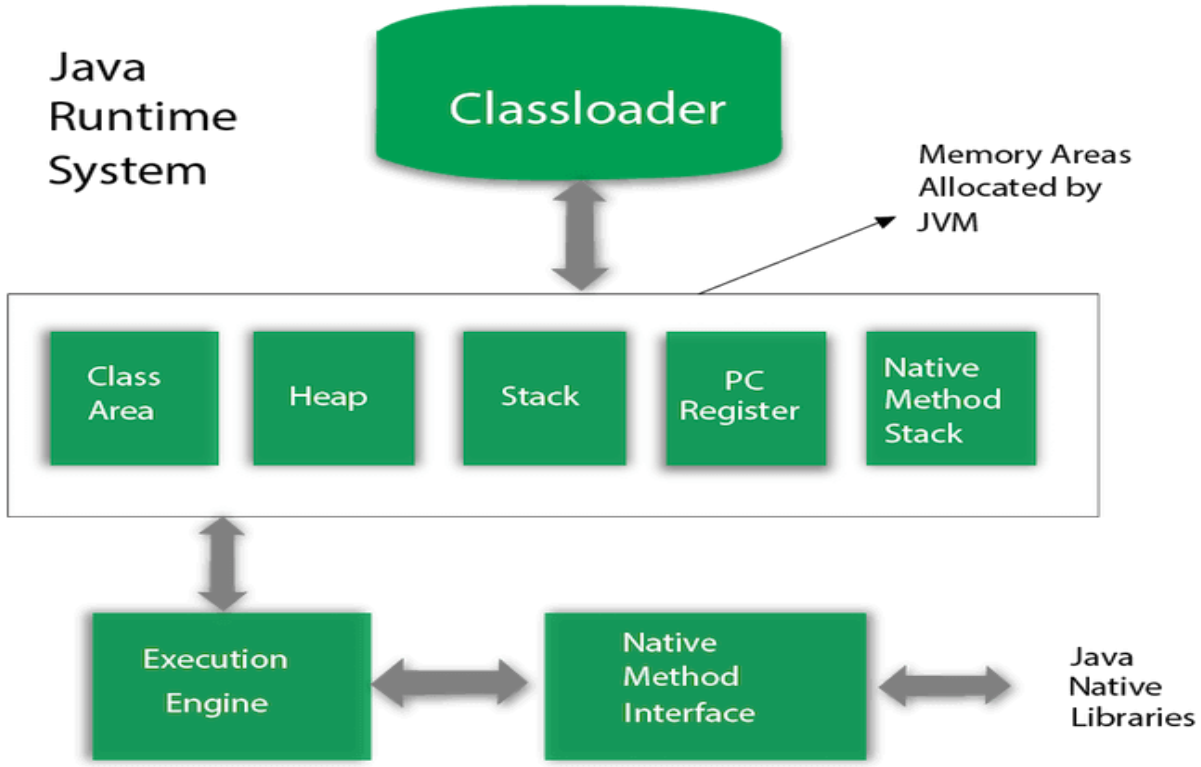
The JVM performs following operation:

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

JVM provides definitions for the:

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

JVM Architecture



1) Classloader

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

1. **Bootstrap ClassLoader**: This is the first classloader which is the super class of Extension classloader. It loads the *rt.jar* file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.
2. **Extension ClassLoader**: This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside *$JAVA_HOME/jre/lib/ext* directory.
3. **System/Application ClassLoader**: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

2) Class(Method) Area

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

3) Heap

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

4) Stack

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register

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

6) Native Method Stack

It contains all the native methods used in the application.

7) Execution Engine

It contains:

1. **A virtual processor**
2. **Interpreter:** Read bytecode stream then execute the instructions.
3. **Just-In-Time(JIT) compiler:** It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

8) Java Native Interface

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

Variables

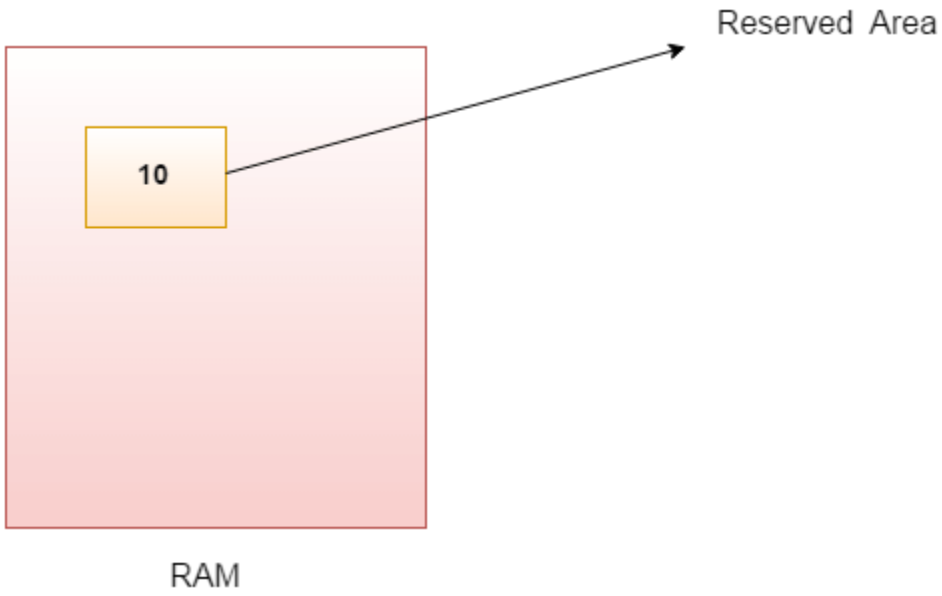
A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in Java: primitive and non-primitive.

Variable

Variable is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.




```
int data=50;//Here data is variable
```

Types of Variables

There are three types of variables in Java:

- local variable
- instance variable
- static variable

1) Local Variable

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

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

It is called instance variable because its value is instance specific and is not shared among instances.

3) Static variable

A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Example to understand the types of variables in java

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

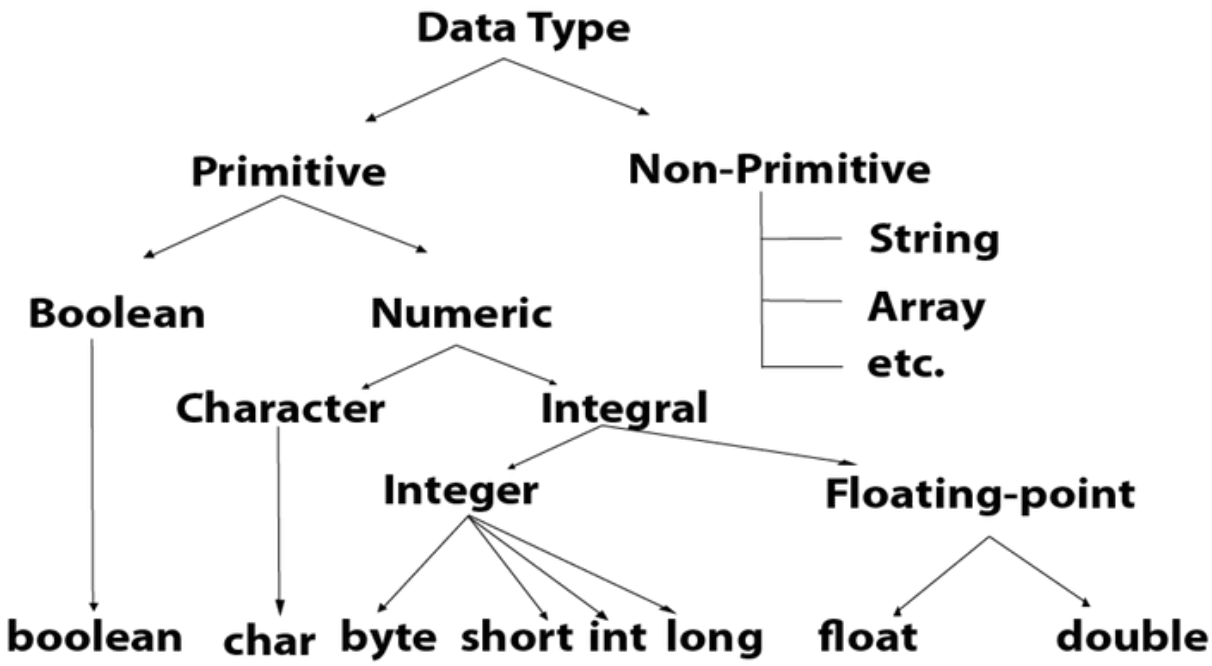
1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type



Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example: Boolean one = false

Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example: byte a = 10, byte b = -20

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example: short s = 10000, short r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is -2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example: int a = 100000, int b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example: long a = 100000L, long b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in

large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example: double d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example: char letterA = 'A'

Operators in Java

Operator in Java is a symbol which 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.

rator Type	Category	Precedence
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -

Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Java Unary Operator Example: ++ and --

```
class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
}}
```

Output:

```
10
```

```
12
12
10
```

Arithmetic Operators

Java arithmatic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

Arithmetic Operator Example

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

Output:

```
15
5
50
2
0
```

v Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Java Left Shift Operator Example

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}}
```

Output:

```
40
80
```

```
80
240
```

Java Right Shift Operator

The Java right shift operator >> is used to move left operands value to right by the number of bits specified by the right operand.

Java Right Shift Operator Example

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}}
```

Output:

```
2
5
2
```

Java AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

```
1. class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=5;
5. int c=20;
6. System.out.println(a<b&&a<c);//false && true = false
7. System.out.println(a<b&a<c);//false & true = false
8. }}
```

Output:

```
false
false
```

Java AND Operator Example: Logical && vs Bitwise &

```
1. class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=5;
5. int c=20;
```

```
6. System.out.println(a<b&&a++<c);//false && true = false
7. System.out.println(a);//10 because second condition is not checked
8. System.out.println(a<b&a++<c);//false && true = false
9. System.out.println(a);//11 because second condition is checked
10. }}
```

Output:

```
false
10
false
11
```

Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//true || true = true
System.out.println(a>b|a<c);//true | true = true
//|| vs |
System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);//10 because second condition is not checked
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);//11 because second condition is checked
}}
```

Output:

```
true
true
true
10
true
11
```

Java Ternary Operator

Java Ternary operator is used as one liner replacement for if-then-else statement and used a lot in Java programming. it is the only conditional operator which takes three operands.

Java Ternary Operator Example

```
class OperatorExample{
```

```
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Output:

```
2
```

Another Example:

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Output:

```
5
```

Java Assignment Operator

Java assignment operator is one of the most common operator. It is used to assign the value on its right to the operand on its left.

Java Assignment Operator Example

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}}
```

Output:

```
14
16
```

Decision Making and Looping

A loop is a way of repeating lines of code more than once. The block of code contained within the loop will be executed again and again until the condition required by the loop is met. For example, you could set up a loop to print out the even numbers between 1 and 100. The code that gets executed each time the loop is run will be the printing out of an even number, the condition the loop is looking to meet is reaching 100 (i.e., 2 4 6 8….96 98).

There are two types of loops:

- **Indeterminate** – an indeterminate loop does not know how many times it will run. For example, you could search through an int array looking for a specific value. The most logical way would be to search each element of the array in order until you find the right value. You don't know if the value is in the first element or the last so the number of times you loop around checking the next element of the array is unknown. Indeterminate loops are the while and do..while loops.
- **Determinate** – a determinate loop knows exactly how many times it will loop. For example, if you want to know how much money you'll be paid for the next twelve months minus tax you could perform the wage calculation 12 times. The determinate loop in Java is the for loop.

Note: Refer control flow statements for more information about looping in statements**.**

Classes and Objects in Java

Classes and Objects are basic concepts of Object Oriented Programming which revolve around the real life entities.

Class:

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. **Modifiers** : A class can be public or has default access (Refer this for details).
2. **Class name:** The name should begin with a initial letter (capitalized by convention).
3. **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
4. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
5. **Body:** The class body surrounded by braces, { }.

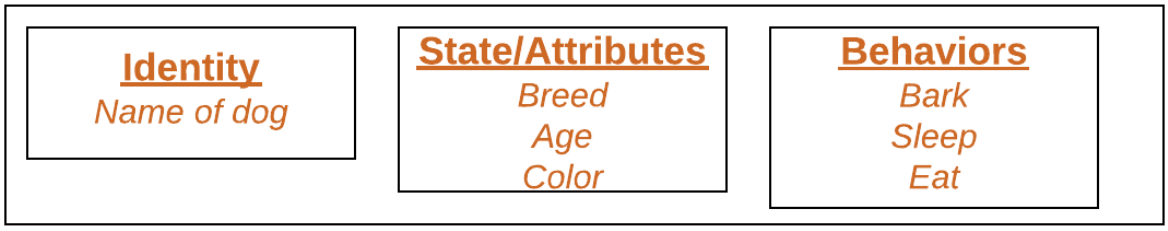
Constructors are used for initializing new objects. Fields are variables that provides the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

There are various types of classes that are used in real time applications such as nested classes, anonymous classes, lambda expressions.

Object:
It is a basic unit of Object Oriented Programming and represents the 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.

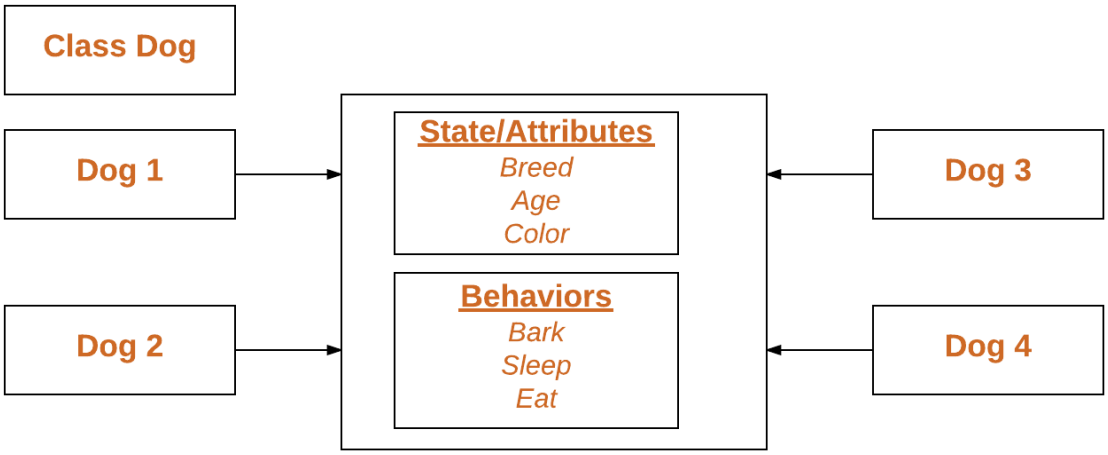
Example of an object : dog



Objects correspond to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, “menu”. An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.

Declaring Objects (Also called instantiating a class):

When an object of a class is created, the class is said to be **instantiated**. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.
Example :



As we declare variables like (type name;). This notifies the compiler that we will use name to refer to data whose type is type. With a primitive variable, this declaration also reserves the proper amount of memory for the variable. So for reference variable, type must be strictly a concrete class name. In general,we **can't** create objects of an abstract class or an interface.

```
Dog tuffy;
```

If we declare reference variable(tuffy) like this, its value will be undetermined(null) until an object is actually created and assigned to it. Simply declaring a reference variable does not create an object.

Initializing an object:

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.

Example:

```
// Class Declaration

public class Dog
{
   // Instance Variables
   String name;
   String breed;
   int age;
   String color;

   // Constructor Declaration of Class
   public Dog(String name, String breed,
              int age, String color)
   {
      this.name = name;
      this.breed = breed;
      this.age = age;
```

```
        this.color = color;
    }

    // method 1
    public String getName()
    {
        return name;
    }

    // method 2
    public String getBreed()
    {
        return breed;
    }

    // method 3
    public int getAge()
    {
        return age;
    }

    // method 4
    public String getColor()
    {
        return color;
    }

    @Override
    public String toString()
    {
        return("Hi my name is "+ this.getName()+
                ".\nMy breed,age and color are " +
                this.getBreed()+"," + this.getAge()+
                ","+ this.getColor());
    }

    public static void main(String[] args)
    {
        Dog tuffy = new Dog("tuffy","papillon", 5, "white");
        System.out.println(tuffy.toString());
    }
}
```

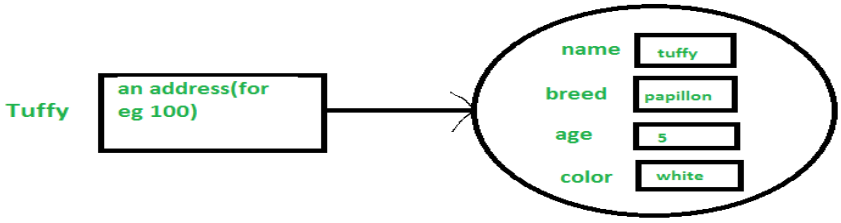
Output:

```
Hi my name is tuffy.
My breed,age and color are papillon,5,white
```

This class contains a single constructor. We can recognize a constructor because its declaration uses the same name as the class and it has no return type. The Java compiler

differentiates the constructors based on the number and the type of the arguments. The constructor in the *Dog* class takes four arguments. The following statement provides "tuffy","papillon",5,"white" as values for those arguments:

```
Dog tuffy = new Dog("tuffy","papillon",5, "white");
```



Note : All classes have at least **one** constructor. If a class does not explicitly declare any, the Java compiler automatically provides a no-argument constructor, also called the default constructor. This default constructor calls the class parent's no-argument constructor (as it contain only one statement i.e super();), or the *Object* class constructor if the class has no other parent (as Object class is parent of all classes either directly or indirectly).

Ways to create object of a class:
There are four ways to create objects in java.Strictly speaking there is only one way(by using *new* keyword),and the rest internally use *new* keyword.

Using new keyword : It is the most common and general way to create object in java. Example:

```
// creating object of class Test
Test t = new Test();
```

Using Class.forName(String className) method : There is a pre-defined class in java.lang package with name Class. The forName(String className) method returns the Class object associated with the class with the given string name.We have to give the fully qualified name for a class. On calling new Instance() method on this Class object returns new instance of the class with the given string name.

```
// creating object of public class Test
// consider class Test present in com.p1 package
Test obj = (Test)Class.forName("com.p1.Test").newInstance();
```

Using clone() method: clone() method is present in Object class. It creates and returns a copy of the object.

```
// creating object of class Test
Test t1 = new Test();


// creating clone of above object
Test t2 = (Test)t1.clone();
```

Creating multiple objects by one type only (A good practice):

In real-time, we need different objects of a class in different methods. Creating a number of references for storing them is not a good practice and therefore we declare a static reference variable and use it whenever required. In this case,wastage of memory is less. The objects that are not referenced anymore will be destroyed by Garbage Collector of java.

Example:

```
Test test = new Test();
test = new Test();
```

In inheritance system, we use parent class reference variable to store a sub-class object. In this case, we can switch into different subclass objects using same referenced variable. Example:

```
class Animal {}


class Dog extends Animal {}
class Cat extends Animal {}


public class Test
{
   // using Dog object
   Animal obj = new Dog();


   // using Cat object
   obj = new Cat();
}
```

Anonymous objects:

Anonymous objects are the objects that are instantiated but are not stored in a reference variable.

- They are used for immediate method calling.

- They will be destroyed after method calling.
- They are widely used in different libraries. For example, in AWT libraries, they are used to perform some action on capturing an event(eg a key press).
- In example below, when a key is button(referred by the btn) is pressed, we are simply creating anonymous object of EventHandler class for just calling handle method.

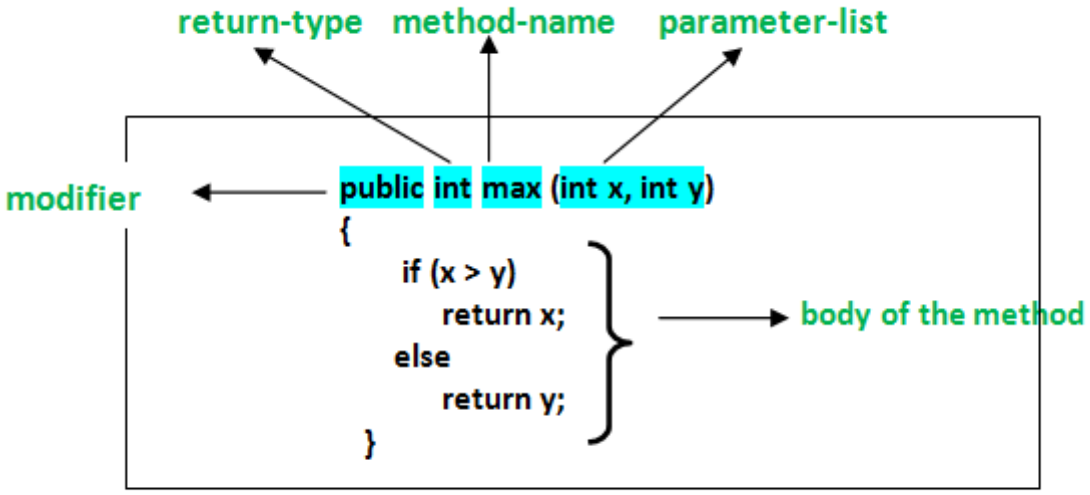
```
btn.setOnAction(new EventHandler()
{
   public void handle(ActionEvent event)
   {
      System.out.println("Hello World!");
   }
});
```

Methods in Java

A method is a collection of statements that perform some specific task and return the result to the caller. A method can perform some specific task without returning anything. Methods allow us to **reuse** the code without retyping the code. In Java, every method must be part of some class which is different from languages like C, C++, and Python.
Methods are **time savers** and help us to **reuse** the code without retyping the code.

Method Declaration
In general, method declarations has six components

- **Modifier**-: Defines **access type** of the method i.e. from where it can be accessed in your application. In Java, there 4 type of the access specifiers.
  - public: accessible in all class in your application.
  - protected: accessible within the class in which it is defined and in its **subclass(es)**
  - private: accessible only within the class in which it is defined.
  - default (declared/defined without using any modifier) : accessible within same class and package within which its class is defined.
- **The return type** : The data type of the value returned by the method or void if does not return a value.
- **Method Name** : the rules for field names apply to method names as well, but the convention is a little different.
- **Parameter list** : Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
- **Exception list** : The exceptions you expect by the method can throw, you can specify these exception(s).
- **Method body** : it is enclosed between braces. The code you need to be executed to perform your intended operations.



Method signature: It consists of the method name and a parameter list (number of parameters, type of the parameters and order of the parameters). The return type and exceptions are not considered as part of it.
Method Signature of above function:
max(int x, int y)

How to name a Method?:

A method name is typically a single word that should be a **verb** in lowercase or multi-word, that begins with a **verb** in lowercase followed by **adjective, noun…..** After the first word, first letter of each word should be capitalized. For example, findSum, computeMax, setX and getX
Generally, A method has a unique name within the class in which it is defined but sometime a method might have the same name as other method names within the same class as method overloading is allowed in Java.

Calling a method
The method needs to be called for using its functionality. There can be three situations when a method is called:
A method returns to the code that invoked it when:

- It completes all the statements in the method
- It reaches a return statement
- Throws an exception

Example:

```
// Program to illustrate methodsin java
import java.io.*;

class Addition {

   int sum = 0;
```

```
    public int addTwoInt(int a, int b){

        // adding two integer value.
        sum = a + b;

        //returning summation of two values.
        return sum;
    }

}

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

        // creating an instance of Addition class
        Addition add = new Addition();

        // calling addTwoInt() method to add two integer using instance created
        // in above step.
        int s = add.addTwoInt(1,2);
        System.out.println("Sum of two integer values :"+ s);

    }
}
```

Output :
Sum of two integer values :3

example to understand method call in detail :

```
// Java program to illustrate different ways of calling a method
import java.io.*;

class Test
{
    public static int i = 0;
    // constructor of class which counts
    //the number of the objects of the class.
    Test()
    {
        i++;

    }
    // static method is used to access static members of the class
    // and for getting total no of objects
    // of the same class created so far
    public static int get()
    {
        // statements to be executed....
        return i;
```

```
    }

    // Instance method calling object directly
    // that is created inside another class 'GFG'.
    // Can also be called by object directly created in the same class
    // and from another method defined in the same class
    // and return integer value as return type is int.
    public int m1()
    {
        System.out.println("Inside the method m1 by object of GFG class");

        // calling m2() method within the same class.
        this.m2();

        // statements to be executed if any
        return 1;
    }

    // It doesn't return anything as
    // return type is 'void'.
    public void m2()
    {

        System.out.println("In method m2 came from method m1");
    }
}

class GFG
{
    public static void main(String[] args)
    {
        // Creating an instance of the class
        Test obj = new Test();

        // Calling the m1() method by the object created in above step.
        int i = obj.m1();
        System.out.println("Control returned after method m1 :" + i);

        // Call m2() method
        // obj.m2();
        int no_of_objects = Test.get();

        System.out.print("No of instances created till now : ");
        System.out.println(no_of_objects);

    }
}
```

Output :

```
Inside the method m1 by object of GFG class
```

```
In method m2 came from method m1
Control returned after method m1 :1
No of instances created till now :
```

Arrays

Normally, an array is a collection of similar type of elements which have a contiguous memory location.

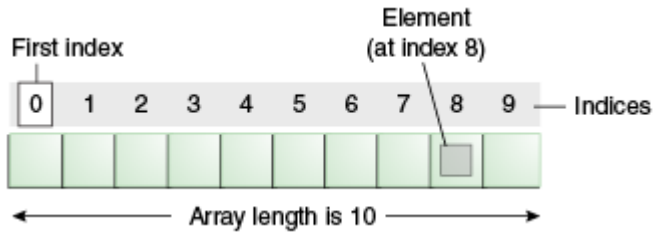
Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator.

In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java. Like C/C++, we can also create single dimentional or multidimentional arrays in Java.

Moreover, Java provides the feature of anonymous arrays which is not available in C/C++.



Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Syntax to Declare an Array in Java

dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];

Instantiation of an Array in Java

arrayRefVar=**new** datatype[size];

Example of Java Array

```
/Java Program to illustrate how to declare, instantiate, initialize
//and traverse the Java array.
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Output:

```
10
20
70
40
50
```

Multidimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java

dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in Java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Example to initialize Multidimensional Array in Java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

Example of Multidimensional Java Array

```
/Java Program to illustrate the use of multidimensional array
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}
}
```

Output:

```
1 2 3
2 4 5
4 4 5
```

Jagged Array

If we are creating odd number of columns in a 2D array, it is known as a jagged array. In other words, it is an array of arrays with different number of columns.

```
//Java Program to illustrate the jagged array
class TestJaggedArray{
   public static void main(String[] args){
      //declaring a 2D array with odd columns
      int arr[][] = new int[3][];
      arr[0] = new int[3];
      arr[1] = new int[4];
      arr[2] = new int[2];
      //initializing a jagged array
      int count = 0;
      for (int i=0; i<arr.length; i++)
         for(int j=0; j<arr[i].length; j++)
            arr[i][j] = count++;

      //printing the data of a jagged array
      for (int i=0; i<arr.length; i++){
         for (int j=0; j<arr[i].length; j++){
            System.out.print(arr[i][j]+" ");
         }
         System.out.println();//new line
      }
   }
}
```

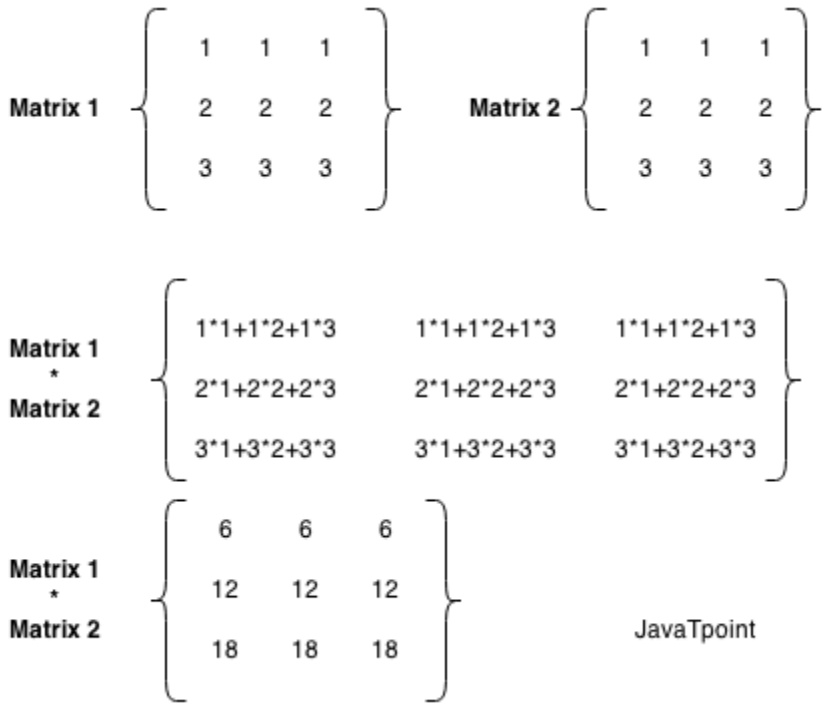
Output:

```
0 1 2
3 4 5 6
7 8
```

Example 2:

Multiplication of 2 Matrices in Java

In the case of matrix multiplication, a one-row element of the first matrix is multiplied by all the columns of the second matrix which can be understood by the image given below.



Let's see a simple example to multiply two matrices of 3 rows and 3 columns.

```
//Java Program to multiply two matrices
public class MatrixMultiplicationExample{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,1,1},{2,2,2},{3,3,3}};
int b[][]={{1,1,1},{2,2,2},{3,3,3}};

//creating another matrix to store the multiplication of two matrices
int c[][]=new int[3][3];  //3 rows and 3 columns

//multiplying and printing multiplication of 2 matrices
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
c[i][j]=0;
for(int k=0;k<3;k++)
{
c[i][j]+=a[i][k]*b[k][j];
}//end of k loop
System.out.print(c[i][j]+" ");  //printing matrix element
}//end of j loop
System.out.println();//new line
}
}}
```

Output:

```
6 6 6
```

```
12 12 12
18 18 18
```

Strings

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

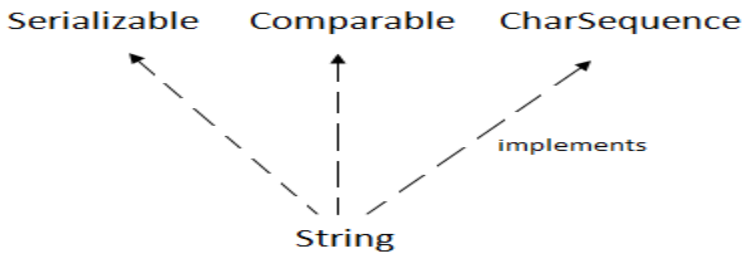
```
char[] ch={'j','a','v','a','t','p','o','i','n','t'};
String s=new String(ch);
```

is same as:

```
String s="javatpoint";
```

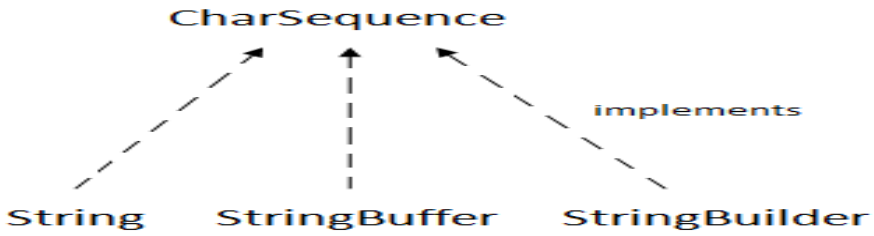
Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class
implements *Serializable*, *Comparable* and *CharSequence* interfaces.



CharSequence Interface

The CharSequence interface is used to represent the sequence of characters.
String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.



The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

We will discuss immutable string later. Let's first understand what is String in Java and how to create the String object.

What is String in java

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

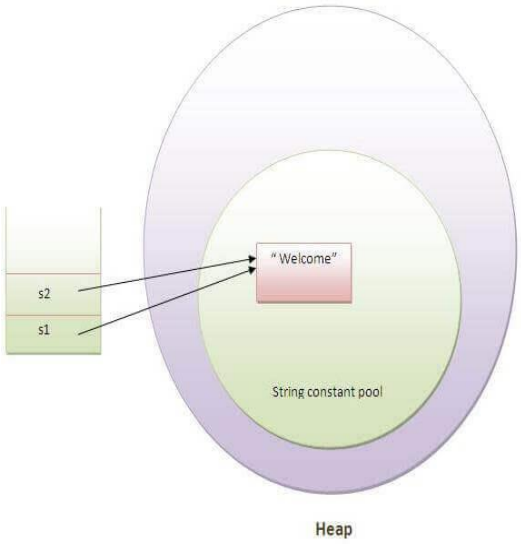
1) String Literal

Java String literal is created by using double quotes. For Example:

```
1. String s="welcome";
```

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

```
1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance
```



In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool, that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

2) By new keyword

String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

Java String Example

```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

Output:

```
java
strings
example
```

Difference between String and StringBuffer

There are many differences between String and StringBuffer. A list of differences between String and

StringBuffer are given below:

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

Immutable String in Java

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

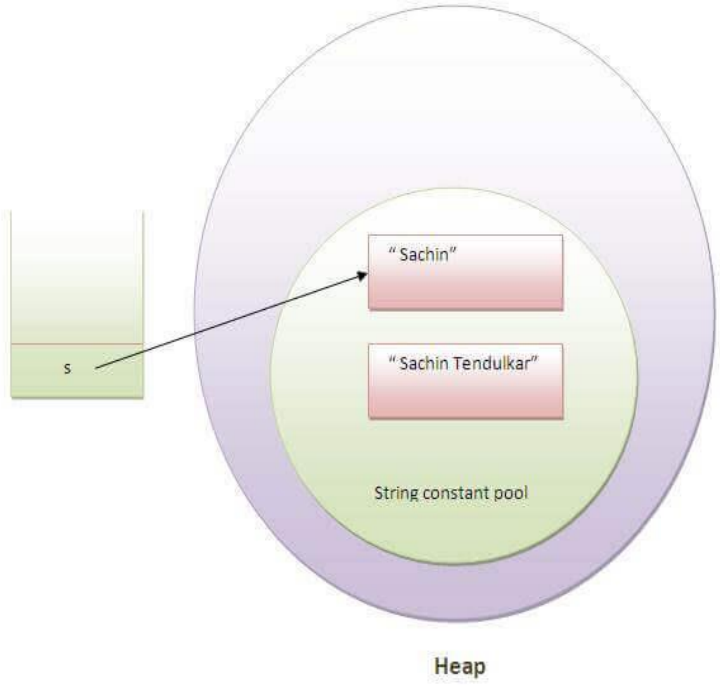
Let's try to understand the immutability concept by the example given below:

```
class Testimmutablestring{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends the string at the end
   System.out.println(s);//will print Sachin because strings are immutable objects
 }
}
```

Output:

Sachin

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with sachintendulkar. That is why string is known as immutable.



As you can see in the above figure that two objects are created but s reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.For example:

```
class Testimmutablestring1{
 public static void main(String args[]){
   String s="Sachin";
   s=s.concat(" Tendulkar");
   System.out.println(s);
 }
}
```

Output:
Sachin Tendulkar

In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object is not modified.

Why string objects are immutable in java?

Because java uses the concept of string literal.Suppose there are 5 reference variables,all referes to one obj "sachin".If one reference variable changes the value of the object, it will be affected to all the referen ce va is why string objects are immutable in java.

INTERFACES:

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways −

- An interface can contain any number of methods.
- An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a .class file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including −

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Declaring Interfaces

The interface keyword is used to declare an interface. Here is a simple example to declare an interface −

Example

Following is an example of an interface –

```
/* File name : NameOfInterface.java */
import java.lang.*;
// Any number of import statements

public interface NameOfInterface {
   // Any number of final, static fields
   // Any number of abstract method declarations\
}
```

Interfaces have the following properties −

- An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

Example

```
/* File name : Animal.java */
interface Animal {
   public void eat();
   public void travel();
}
```

Implementing Interfaces

When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Example

```
/* File name : MammalInt.java */
public class MammalInt implements Animal {

   public void eat() {
      System.out.println("Mammal eats");
   }
```

```
  public void travel() {
    System.out.println("Mammal travels");
  }

  public int noOfLegs() {
    return 0;
  }

  public static void main(String args[]) {
    MammalInt m = new MammalInt();
    m.eat();
    m.travel();
  }
}
```

Output

```
Mammal eats
Mammal travels
```

When overriding methods defined in interfaces, there are several rules to be followed −

- Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.
- The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.
- An implementation class itself can be abstract and if so, interface methods need not be implemented.

When implementation interfaces, there are several rules −

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, in a similar way as a class can extend another class.

Extending Interfaces

An interface can extend another interface in the same way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

The following Sports interface is extended by Hockey and Football interfaces.

Example

```
// Filename: Sports.java
public interface Sports {
  public void setHomeTeam(String name);
  public void setVisitingTeam(String name);
```

```
}

// Filename: Football.java
public interface Football extends Sports {
   public void homeTeamScored(int points);
   public void visitingTeamScored(int points);
   public void endOfQuarter(int quarter);
}

// Filename: Hockey.java
public interface Hockey extends Sports {
   public void homeGoalScored();
   public void visitingGoalScored();
   public void endOfPeriod(int period);
   public void overtimePeriod(int ot);
}
```

The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports.

Extending Multiple Interfaces

A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.

The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

For example, if the Hockey interface extended both Sports and Event, it would be declared as −

Example

```
public interface Hockey extends Sports, Event
```

Tagging Interfaces

The most common use of extending interfaces occurs when the parent interface does not contain any methods. For example, the MouseListener interface in the java.awt.event package extended java.util.EventListener, which is defined as −

Example

```
package java.util;
public interface EventListener
{}
```

An interface with no methods in it is referred to as a tagging interface. There are two basic design purposes of tagging interfaces −

Creates a common parent − As with the EventListener interface, which is extended by dozens of other interfaces in the Java API, you can use a tagging interface to create a common parent among a group of interfaces. For example, when an interface extends EventListener, the JVM knows that this particular interface is going to be used in an event delegation scenario.

Adds a data type to a class − This situation is where the term, tagging comes from. A class that implements a tagging interface does not need to define any methods (since the interface does not have any), but the class becomes an interface type through polymorphism.

Packages In Java

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

- Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
- Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.
- Packages can be considered as data encapsulation (or data-hiding).

All we need to do is put related classes into packages. After that, we can simply write an import class from existing packages and use it in our program. A package is a container of a group of related classes where some of the classes are accessible are exposed and others are kept for internal purpose.
We can reuse existing classes from the packages as many time as we need it in our program.

How packages work?
Package names and directory structure are closely related. For example if a package name is *college.staff.cse*, then there are three directories, *college*, *staff* and *cse* such that *cse* is present in *staff* and *staff* is present *college*. Also, the directory *college* is accessible through CLASSPATH variable, i.e., path of parent directory of college is present in CLASSPATH. The idea is to make sure that classes are easy to locate.

Package naming conventions : Packages are named in reverse order of domain names, i.e., org.geeksforgeeks.practice. For example, in a college, the recommended convention is college.tech.cse, college.tech.ee, college.art.history, etc.

Adding a class to a Package : We can add more classes to a created package by using package name at the top of the program and saving it in the package directory. We need a new **java** file to define a public class, otherwise we can add the new class to an existing **.java** file and recompile it.

Subpackages: Packages that are inside another package are the **subpackages**. These are not imported by default, they have to imported explicitly. Also, members of a subpackage have no access privileges, i.e., they are considered as different package for protected and default

access specifiers.

Example :

```
import java.util.*;
```

util is a subpackage created inside **java** package.

Accessing classes inside a package

Consider following two statements :

```
// import the Vector class from util package.
import java.util.vector;


// import all the classes from util package
import java.util.*;
```

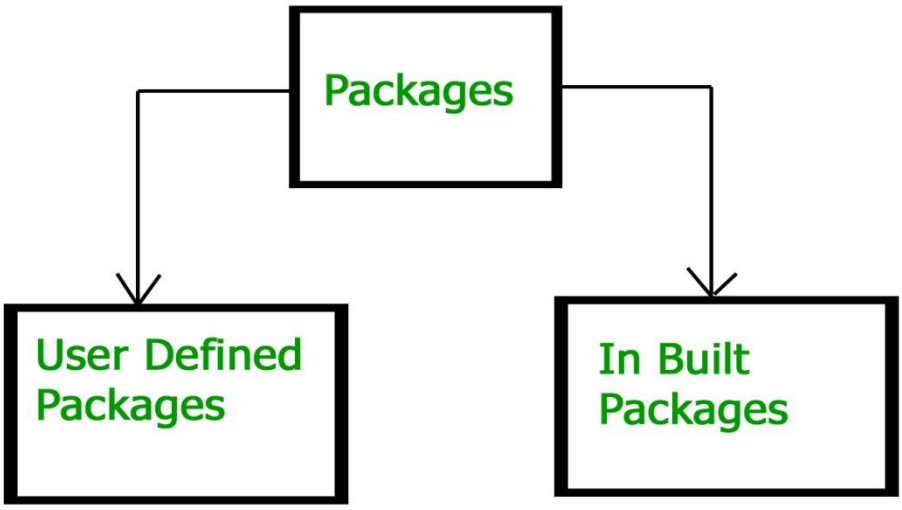
- First Statement is used to import **Vector** class from **util** package which is contained inside **java**.
- Second statement imports all the classes from **util** package.

```
// All the classes and interfaces of this package
// will be accessible but not subpackages.
import package.*;


// Only mentioned class of this package will be accessible.
import package.classname;


// Class name is generally used when two packages have the same
// class name. For example in below code both packages have
// date class so using a fully qualified name to avoid conflict
import java.util.Date;
import my.packag.Date;
```

```
// Java program to demonstrate accessing of members when
// corresponding classes are imported and not imported.
import java.util.Vector;


public class ImportDemo
{
```

```
  public ImportDemo()
  {
    // java.util.Vector is imported, hence we are
    // able to access directly in our code.
    Vector newVector = new Vector();

    // java.util.ArrayList is not imported, hence
    // we were referring to it using the complete
    // package.
    java.util.ArrayList newList = new java.util.ArrayList();
  }

  public static void main(String arg[])
  {
    new ImportDemo();
  }
}
```

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);
    }
}
```

Now we can use the **MyClass** class in our program.

```
/* import 'MyClass' class from 'names' myPackage */
import myPackage.MyClass;

public class PrintName
{
   public static void main(String args[])
   {
      // Initializing the String variable
      // with a value
      String name = "GeeksforGeeks";

      // Creating an instance of class MyClass in
      // the package.
      MyClass obj = new MyClass();

      obj.getNames(name);
   }
}
```

Note : MyClass.java must be saved inside the **myPackage** directory since it is a part of the package.

Using Static Import:

Static import is a feature introduced in **Java** programming language (versions 5 and above) that allows members (fields and methods) defined in a class as public **static** to be used in Java code without specifying the class in which the field is defined.
Following program demonstrates **static import** :

```
// Note static keyword after import.
import static java.lang.System.*;

class StaticImportDemo
{
   public static void main(String args[])
   {
       // We don't need to use 'System.out'
       // as imported using static.
       out.println("GeeksforGeeks");
   }
}
```

Output:

```
GeeksforGeeks
```

Errors and Exceptions In Java

Error : An Error "indicates serious problems that a reasonable application should not try to catch."
Both Errors and Exceptions are the subclasses of java.lang.Throwable class. Errors are the conditions which cannot get recovered by any handling techniques. It surely cause termination of the program abnormally. Errors belong to unchecked type and mostly occur at runtime. Some of the examples of errors are Out of memory error or a System crash error.

Example:

```
//Java program illustrating stack overflow error
// by doing infinite recursion
  class StackOverflow {
    public static void test(int i)
    {
        // No correct as base condition leads to
        // non-stop recursion.
        if (i == 0)
            return;
        else {
            test(i++);
        }
    }
}
public class ErrorEg {

    public static void main(String[] args)
    {

        // eg of StackOverflowError
        StackOverflow.test(5);
```

```
    }
}
```

Output:

```
Exception in thread "main" java.lang.StackOverflowError
    at StackOverflow.test(ErrorEg.java:7)
    at StackOverflow.test(ErrorEg.java:7)
    at StackOverflow.test(ErrorEg.java:7)
    at StackOverflow.test(ErrorEg.java:7)
    at StackOverflow.test(ErrorEg.java:7)
...
```

Exceptions : An Exception "indicates conditions that a reasonable application might want to catch."
Exceptions are the conditions that occur at runtime and may cause the termination of program. But they are recoverable using try, catch and throw keywords. Exceptions are divided into two catagories : checked and unchecked exceptions. Checked exceptions like IOException known to the compiler at compile time while unchecked exceptions like ArrayIndexOutOfBoundException known to the compiler at runtime. It is mostly caused by the program written by the programmer.

Example:

```
// Java program illustrating exception thrown
// by AritmeticExcpetion class

public class ExceptionEg {

    public static void main(String[] args)
    {
        int a = 5, b = 0;

        // Attempting to divide by zero
        try {
```

```
            int c = a / b;
        }
        catch (ArithmeticException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
java.lang.ArithmeticException: / by zero
    at ExceptionEg.main(ExceptionEg.java:8)
```

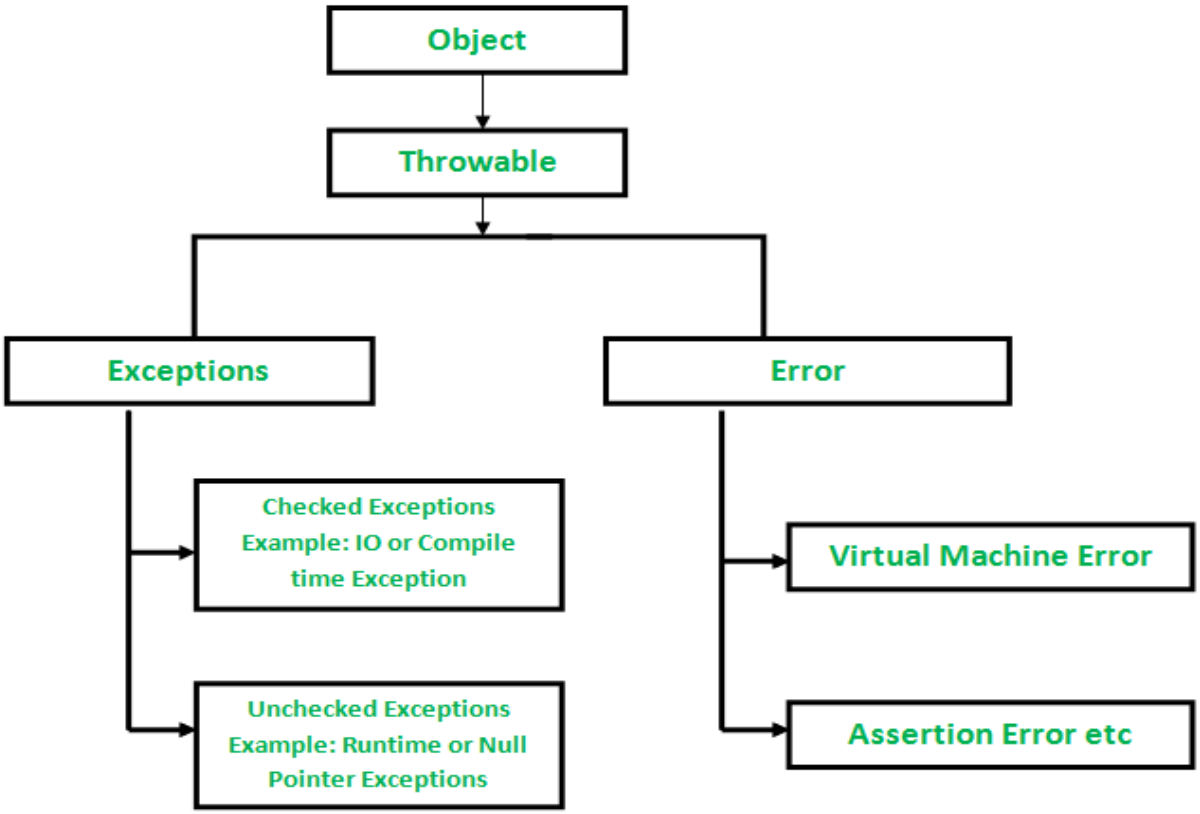
Summary of Differences

ERRORS	EXCEPTIONS
Recovering from Error is not possible.	We can recover from exceptions by either using try-catch block or throwing exceptions back to caller.
All errors in java are unchecked type.	Exceptions include both checked as well as unchecked type.
Errors are mostly caused by the environment in which program is running.	Program itself is responsible for causing exceptions.
Errors occur at runtime and not	All exceptions occurs at runtime but

known to the compiler.	checked exceptions are known to compiler while unchecked are not.
They are defined in java.lang.Error package.	They are defined in java.lang.Exception package
Examples : java.lang.StackOverflowError, java.lang.OutOfMemoryError	Examples : Checked Exceptions : SQLException, IOException Unchecked Exceptions : ArrayIndexOutOfBoundException, NullPointerException, ArithmeticException

What is an Exception?
An 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.

Exception Hierarchy
All exception and errors types are sub classes of class **Throwable**, which is base class of 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** are 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.



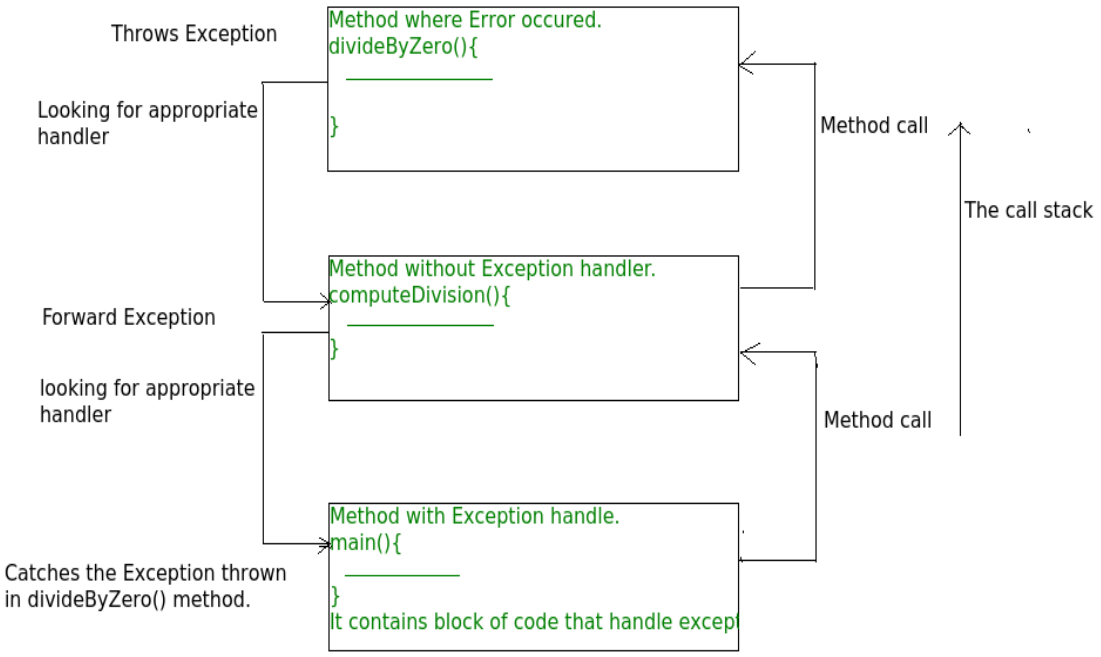
How JVM handle an Exception?

Default Exception Handling : Whenever inside a method, if an exception has occurred, the method creates an Object known as Exception Object and hands it off to the run-time system(JVM). The exception object contains name and description of the exception, and current state of the program where exception has occurred. Creating the Exception Object and handling it to the run-time system is called throwing an Exception.There might be the list of the methods that had been called to get to the method where exception was occurred. This ordered list of the methods is called **Call Stack**.Now the following procedure will happen.

- The run-time system searches the call stack to find the method that contains block of code that can handle the occurred exception. The block of the code is called **Exception handler**.
- The run-time system starts searching from the method in which exception occurred, proceeds through call stack in the reverse order in which methods were called.
- If it finds appropriate handler then it passes the occurred exception to it. Appropriate handler means the type of the exception object thrown matches the type of the exception object it can handle.
- If run-time system searches all the methods on call stack and couldn't have found the appropriate handler then run-time system handover the Exception Object to **default exception handler** , which is part of run-time system. This handler prints the exception information in the following format and terminates program **abnormally**.

```
Exception in thread "xxx" Name of Exception : Description
... ..... ..  // Call Stack
```

See the below diagram to understand the flow of the call stack.



The call stack and searching the call stack for exception handler.

Example :

```
// Java program to demonstrate how exception is thrown.

class ThrowsExecp{

    public static void main(String args[]){

      String str = null;

      System.out.println(str.length());

    }

}
```

Output :

```
Exception in thread "main" java.lang.NullPointerException
   at ThrowsExecp.main(File.java:8)
```

Let us see an example that illustrate how run-time system searches appropriate exception handling code on the call stack :

```
// Java program to demonstrate exception is thrown
// how the runTime system searches th call stack
// to find appropriate exception handler.
class ExceptionThrown
{
    // It throws the Exception(ArithmeticException).
    // Appropriate Exception handler is not found within this method.
    static int divideByZero(int a, int b){
     // this statement will cause ArithmeticException(/ by zero)
      int i = a/b;
      return i;
    }
    // The runTime System searches the appropriate Exception handler
    // in this method also but couldn't have found. So looking forward
    // on the call stack.
    static int computeDivision(int a, int b) {

        int res =0;
        try
        {
          res = divideByZero(a,b);
        }
        // doesn't matches with ArithmeticException
        catch(NumberFormatException ex)
        {
```

```
            System.out.println("NumberFormatException is occured");
        }
        return res;
    }

    // In this method found appropriate Exception handler.
    // i.e. matching catch block.
    public static void main(String args[]){

        int a = 1;
        int b = 0;
      try
        {
            int i = computeDivision(a,b);
         }
       // matching ArithmeticException
        catch(ArithmeticException ex)
        {
            // getMessage will print description of exception(here / by zero)
            System.out.println(ex.getMessage());
        }
    }
}
```

Output :

```
/ by zero.
```

How Programmer handles an exception?

Customized Exception Handling : Java exception handling is managed via five keywords: **try**, **catch**, **throw**, **throws**, and **finally**. Briefly, here is how they work. Program statements that you think can raise exceptions are contained within a try block. If an exception occurs within the try block, it is thrown. Your code can catch this exception (using catch block) and handle it in some rational manner. System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword throw. Any exception that is thrown out of a method must be specified as such by a throws clause. Any code that absolutely must be executed after a try block completes is put in a finally block.

Detailed Article: Control flow in try catch finally block
Need of try-catch clause(Customized Exception Handling)
Consider the following java program.

```
// java program to demonstrate

// need of try-catch clause


class GFG {

    public static void main (String[] args) {


        // array of size 4.

        int[] arr = new int[4];


        // this statement causes an exception

        int i = arr[4];


        // the following statement will never execute

        System.out.println("Hi, I want to execute");

    }

}
```

Output :

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
    at GFG.main(GFG.java:9)
```

Explanation : In the above example an array is defined with size i.e. you can access elements only from index 0 to 3. But you trying to access the elements at index 4(by mistake) that's why it is throwing an exception.In this case, JVM terminates the program **abnormally**. The statement *System.out.println("Hi, I want to execute");* will never execute. To execute it, we must handled the exception using try-catch. Hence to continue normal flow of the program, we need try-catch clause.

How to use try-catch clause

```
try {
// block of code to monitor for errors
// the code you think can raise an exception
}
catch (ExceptionType1 exOb) {
// exception handler for ExceptionType1
}
catch (ExceptionType2 exOb) {
// exception handler for ExceptionType2
}
// optional
finally {
// block of code to be executed after try block ends
}
```

Points to remember :

- In a method, there can be more than one statements that might throw exception, So put all these statements within its own **try** block and provide separate exception handler within own **catch** block for each of them.
- If an exception occurs within the **try** block, that exception is handled by the exception handler associated with it. To associate exception handler, we must put **catch** block after it. There can be more than one exception handlers. Each **catch** block is a exception handler that handles the exception of the type indicated by its argument. The argument, ExceptionType declares the type of the exception that it can handle and must be the name of the class that inherits from **Throwable** class.
- For each try block there can be zero or more catch blocks, but **only one** finally block.
- The finally block is optional.It always gets executed whether an exception occurred in try block or not . If exception occurs, then it will be executed after **try and catch blocks.** And if exception does not occur then it will be executed after the **try** block. The finally block in java is used to put important codes such as clean up code e.g. closing the file or closing the connection.

INTERNET

INTRODUCTION:

- The **Internet** is a massive network of networks, a networking infrastructure. It connects millions of computers together globally, forming a network in which any computer can communicate with any other computer as long as they are both connected to the **Internet.**
- The **Internet** is a worldwide network of computer networks that connects university, government, commercial, and other computers in over 150 countries. ... Using the **Internet**, you can send electronic mail, chat with colleagues around the world, and obtain information on a wide variety of subjects.
- The **Internet**, sometimes called simply "the Net," is a worldwide system of computer networks -- a network of networks in which users at any one computer can (if they have permission) get information from any other computer (and sometimes talk directly to users at other computers).
- The internet is global network of interconnected network, enabling users to share information along multiple channels.
- A majority of widely accessible information on the internet consists of interlinked hypertext documents and other resources of the World Wide Web(www).
- The movement of information in the internet is achieved via system of interconnected computer networks that share data by packet switching using the standardized Internet Protocol suite(TCP/IP).
- The Internet is a worldwide network of computer networks that connects university, government, commercial, and other computers in over 150 countries. There are thousands of networks, tens of thousands of computers, and millions of users on the Internet, with the numbers expanding daily. Using the Internet, you can send electronic mail, chat with colleagues around the world, and obtain information on a wide variety of subjects.
- **Three principal uses of the Internet are:**

1. **Electronic mail:** Electronic mail, or e-mail, lets you electronically "mail" messages to users who have Internet E-mail addresses. Delivery time varies, but it's possible to send mail across the globe and get a response in minutes. LISTSERVs are special interest mailing lists which allow for the exchange of information between large numbers of people.
2. **USENET newsgroups:** USENET is a system of special interest discussion groups, called newsgroups, to which readers can send, or "post" messages which are then

distributed to other computers in the network. (Think of it as a giant set of electronic bulletin boards.) Newsgroups are organized around specific topics, for example, **alt.education.research, alt.education.distance, and misc.education.science**.

3. **Information files:** Government agencies, schools, and universities, commercial firms, interest groups, and private individuals place a variety of information on-line. The files were originally text only, but increasingly contain pictures and sound.

PURPOSE:

- The **purpose of the Internet** has always been to share and distribute information via networked computers. The **Internet** dates back to research commissioned by the federal government of the United States in the 1960s to build robust, fault-tolerant communication with computer networks.
- **Information technology** helps business improve the processes of business it drives revenue growth, helps them achieve cost efficiency and more importantly, ensures they increase revenue growth while maintaining a competitive edge in the market place.
- **Internet** Science and Technology (IST) focuses on improving the interaction between people and/or automated processes over time and distance through the application of information and communication technology.

OBJECTIVES:

- Explain various terminology used in Internet
- Use various services provided by internet
- Search the desired information over internet
- Define e-mail and various features
- Explain the process of downloading file.

NETWORKS:

A computer network is mainly of four types:

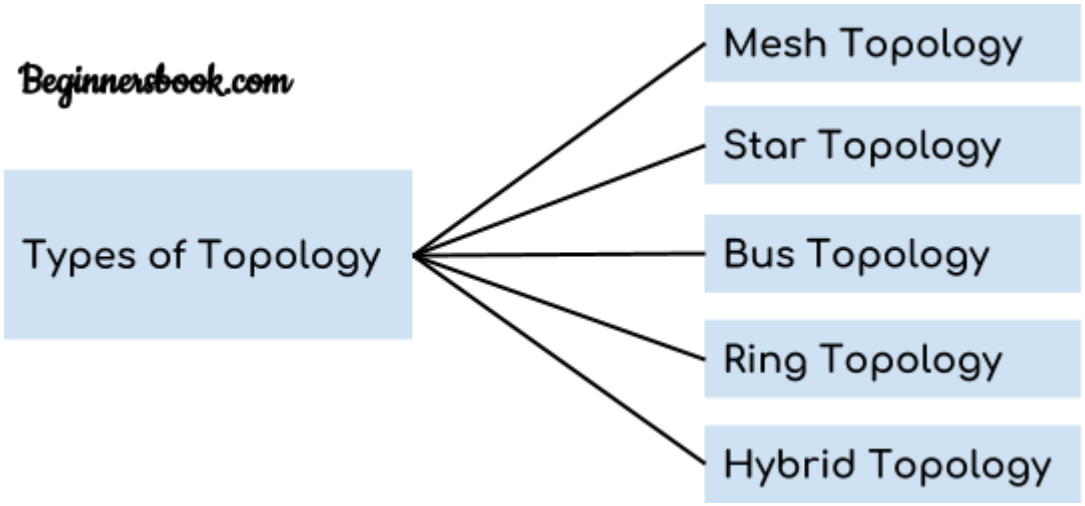
1. LAN(Local Area Network)
2. PAN(Personal Area Network)
3. MAN(Metropolitan Area Network)
4. WAN(Wide Area Network)

Advantages of Networking:

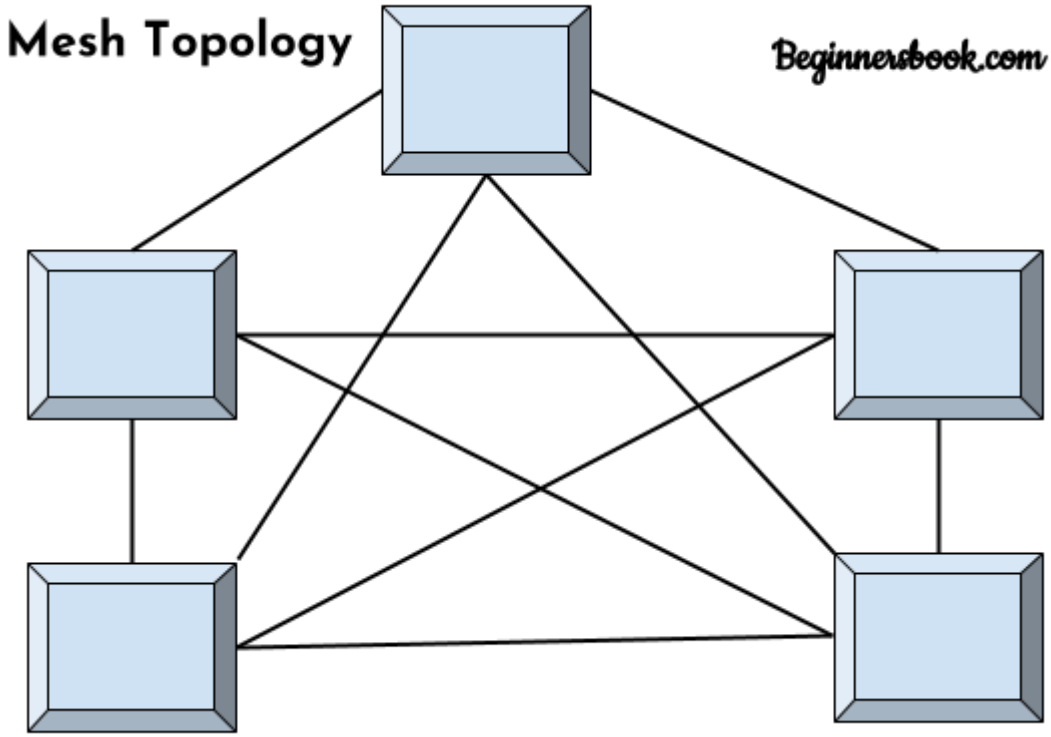
- File sharing – you can easily share data between different users, or access it remotely if you keep it on other connected devices.
- Resource sharing – using **network**-connected peripheral devices like printers, scanners and copiers, or sharing software between multiple users, saves money.

NETWORK TOPOLOGIES:

- Geometric representation of how the computers are connected to each other is known as topology.
- There are five types of topology – Mesh, Star, Bus, Ring and Hybrid.



1. **Mesh Topology-** In mesh topology each device is connected to every other device on the network through a dedicated point-to-point link. When we say dedicated it means that the link only carries data for the two connected devices only.



Advantages of Mesh topology

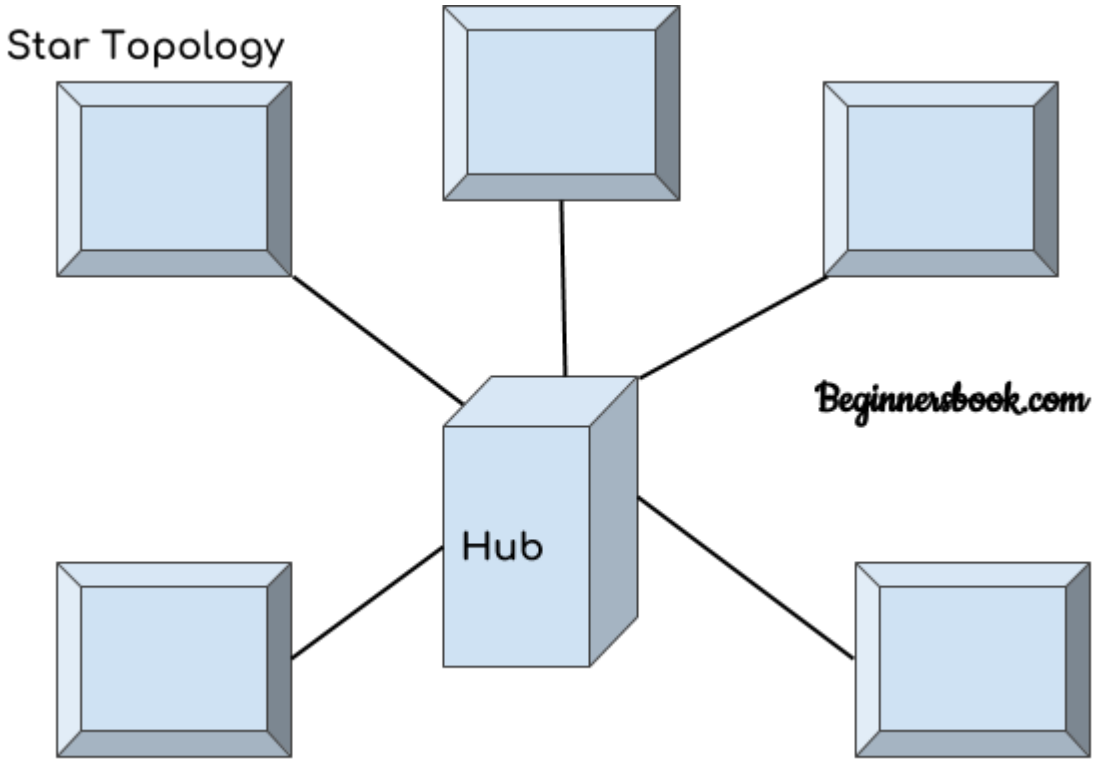
1. No data traffic issues as there is a dedicated link between two devices which means the link is only available for those two devices.
2. Mesh topology is reliable and robust as failure of one link doesn't affect other links and the communication between other devices on the network.
3. Mesh topology is secure because there is a point to point link thus unauthorized access is not possible.
4. Fault detection is easy.

Disadvantages of Mesh topology

1. Amount of wires required to connected each system is tedious and headache.
2. Since each device needs to be connected with other devices, number of I/O ports required must be huge.
3. Scalability issues because a device cannot be connected with large number of devices with a dedicated point to point link.

2. **Star Topology-** In star topology each device in the network is connected to a central device called hub. Unlike Mesh topology, star topology doesn't allow direct communication between devices, a device must have to communicate through hub. If

one device wants to send data to other device, it has to first send the data to hub and then the hub transmit that data to the designated device.



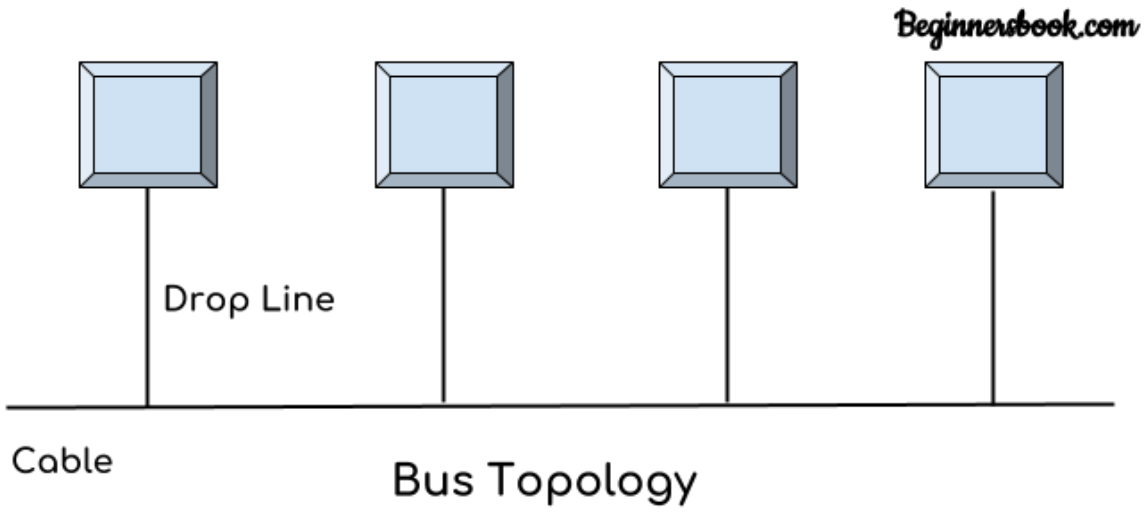
Advantages of Star topology

1. Less expensive because each device only need one I/O port and needs to be connected with hub with one link.
2. Easier to install
3. Less amount of cables required because each device needs to be connected with the hub only.
4. Robust, if one link fails, other links will work just fine.
5. Easy fault detection because the link can be easily identified.

Disadvantages of Star topology

1. If hub goes down everything goes down, none of the devices can work without hub.
2. Hub requires more resources and regular maintenance because it is the central system of star topology.

3. **Bus Topology-** In bus topology there is a main cable and all the devices are connected to this main cable through drop lines. There is a device called tap that connects the drop line to the main cable. Since all the data is transmitted over the main cable, there is a limit of drop lines and the distance a main cable can have.



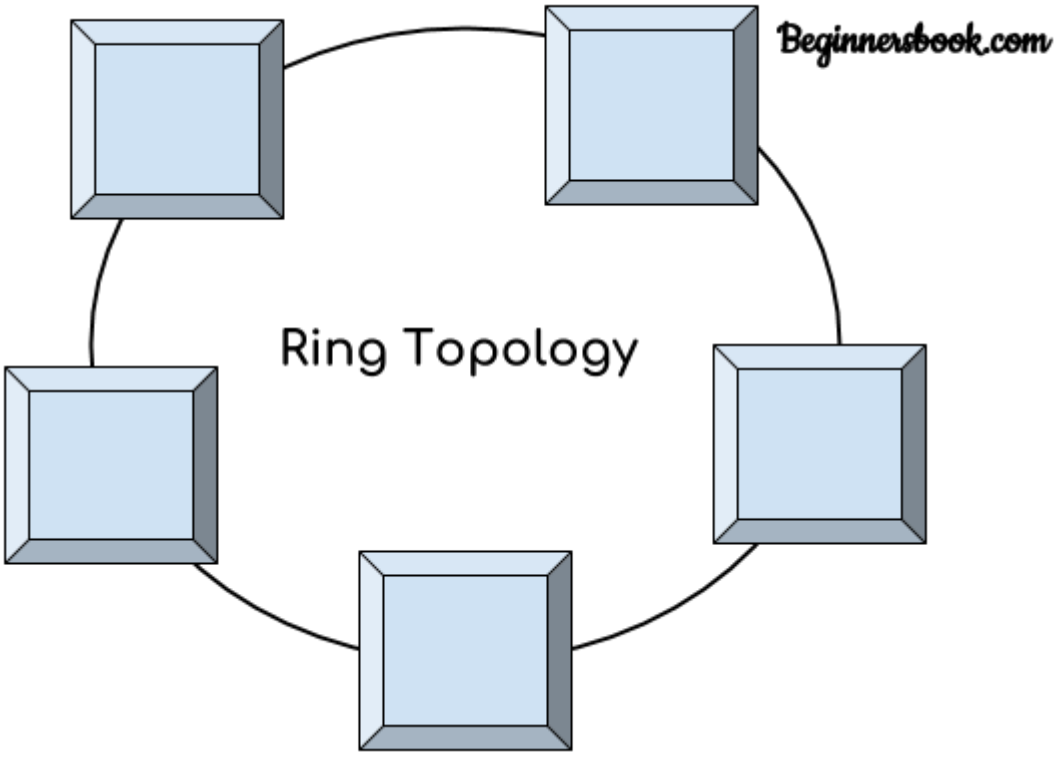
Advantages of bus topology

1. Easy installation, each cable needs to be connected with backbone cable.
2. Less cables required than Mesh and star topology

Disadvantages of bus topology

1. Difficultly in fault detection.
2. Not scalable as there is a limit of how many nodes you can connect with backbone cable.

4. **Ring Topology-** In ring topology each device is connected with the two devices on either side of it. There are two dedicated point to point links a device has with the devices on the either side of it. This structure forms a ring thus it is known as ring topology. If a device wants to send data to another device then it sends the data in one direction, each device in ring topology has a repeater, if the received data is intended for other device then repeater forwards this data until the intended device receives it.



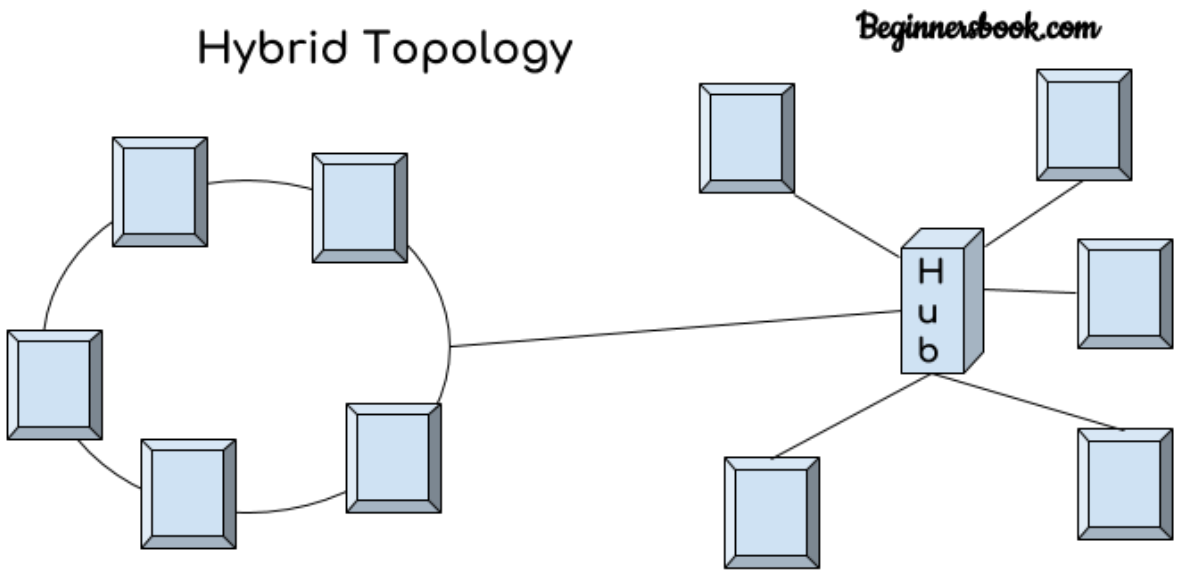
Advantages of Ring Topology

1. Easy to install.
2. Managing is easier as to add or remove a device from the topology only two links are required to be changed.

Disadvantages of Ring Topology

1. A link failure can fail the entire network as the signal will not travel forward due to failure.
2. Data traffic issues, since all the data is circulating in a ring.

5. **Hybrid Topology-** A combination of two or more topology is known as hybrid topology. For example a combination of star and mesh topology is known as hybrid topology.



Advantages of Hybrid topology

1. We can choose the topology based on the requirement for example, scalability is our concern then we can use star topology instead of bus technology.
2. Scalable as we can further connect other computer networks with the existing networks with different topologies.

Disadvantages of Hybrid topology

1. Fault detection is difficult.
2. Installation is difficult.
3. Design is complex so maintenance is high thus expensive.

TCP/IP PROTOCOL:

- **TCP/IP** stands for Transmission Control **Protocol**/Internet **Protocol**, which **is** a set of networking **protocols** that allows two or more computers to communicate.
- **TCP/IP** allows one computer to talk to another computer via the **Internet** through compiling packets of data and sending them to right location.
- **Packet-** is a unit of data transmitted from one location to another.
- The **difference** is that **TCP** is responsible for the data delivery of a packet and **IP** is responsible for the logical addressing. In other words, **IP** obtains the address and **TCP** guarantees delivery of data to that address.

- The **Internet Protocol** (**IP**) is the principal communications **protocol** in the **Internet protocol** suite for relaying datagrams across network boundaries. Its routing function enables **internetworking**, and essentially establishes the **Internet**.
- **Four layers of TCP/IP** model are :
  1) Application **Layer**
  2) Transport **Layer**
  3) Internet **Layer**
  4) Network Interface.
- Application **layer** interacts with an application program, which is the highest level of OSI model.

OSI MODEL:

The Open Systems Interconnection (OSI) model is a conceptual model created by the International Organization for Standardization which enables diverse communication systems to communicate using standard protocols. In plain English, the OSI provides a standard for different computer systems to be able to communicate with each other.

The OSI model can be seen as a universal language for computer networking. It's based on the concept of splitting up a communication system into seven abstract layers, each one stacked upon the last.

7 LAYERS OF OSI MODEL ARE:

1. The Physical Layer- Transmits raw bit stream over the physical medium.
2. The Data Link Layer- Defines the format of data on the network
3. The Network Layer- Decides which physical path the data will take
4. The Transport Layer- Transmits data using transmission protocols including TCP and UDP
5. The Session Layer- Maintains connections and is responsible for controlling ports and sessions
6. The Presentation Layer- Ensure that data is in a usable format and is where data encryption occurs
7. The Application Layer- Human-computer interaction layer, where applications can access the network services. This is the only layer that directly interacts with data from the user.

CLIENT-SERVER MODEL:

- **Client-Server Model**. The **client-server model** describes how a **server** provides resources and services to one or more **clients**.
- Examples of servers include web servers, mail servers, and file servers. Each of these servers provide resources to **client** devices, such as desktop computers, laptops, tablets, and smartphones.

- Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs, which share their resources with clients. A client does not share any of its resources, but it requests content or service from a server. Clients, therefore, initiate communication sessions with servers, which await incoming requests. Examples of computer applications that use the client-server model are Email, network printing, and the World Wide Web.
- **Client-Server role-** The *client-server* characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services. Servers are classified by the services they provide.
- Whether a computer is a client, a server, or both, is determined by the nature of the application that requires the service functions. For example, a single computer can run a web servers and file server software at the same time to serve different data to clients making different kinds of requests.
- **Client and server communication-** Clients and servers exchange messages in a request–response messaging pattern. The client sends a request, and the server returns a response. This exchange of messages is an example of inter-process communication. To communicate, the computers must have a common language, and they must follow rules so that both the client and the server know what to expect. The language and rules of communication are defined in a communications protocol.

Advantages of Client-Server model:

1. Centralized system with all data in a single place.
2. Cost efficient requires less maintenance cost and Data recovery is possible.
3. The capacity of the Client and Servers can be changed separately.

Disadvantages of Client-Server model:

1. Clients are prone to viruses, Trojans and worms if present in the Server or uploaded into the Server.
2. Servers are prone to Denial of Service (DOS) attacks.
3. Data packets may be spoofed o modified during transmission.
4. Phishing or capturing login credentials or other useful information of the user are common and MITM (Man in the Middle) attacks are common.

INTERNET PROGRAMMING

OVERVIEW:

An Internet application is a client-server application that uses standard Internet protocols for connecting the client to the server. You can use exactly the same techniques to create a true Internet application, which is available publicly through the World Wide Web, or to create an intranet application. An intranet application is one which runs on your corporate intranet, and is only available to the staff in your corporation. Whenever we talk about Internet applications, we mean either true Internet applications or intranet applications.

Internet applications are thin-client, thick-server. This means that the client end, the part the end-user sees and interacts with, is only responsible for the user interface. The client runs on a Web browser - the standard tool for accessing the Internet. All the processing is done at the server end, where your corporate data is.

Programming Languages:

- **HTML5** - HyperText Markup Language
  - Semantic Strucure
- **CSS3** - Cascading Style Sheets
  - Presentation with style formatting
- **Javascript6**
  - Behavior – responses to events (mouse clicks, hover, …)
- **C# Programming Language**
  - A Java like language in the C/C++ family
  - Server-side programs that control how pages are generated.
- **SQL and Linq** - Structured Query Languages
  - Used to manage data in Relational Databases like SQL Server

WWW:

The **World Wide Web** (**WWW**), commonly known as **the Web**, is an information system where documents and other web resources are identified by Uniform Resource Locators (URLs, such as *https://www.example.com/*), which may be interlinked by hypertext, and are accessible over the Internet.[1][2] The resources of the WWW are transferred via the Hypertext Transfer Protocol (HTTP) and may be accessed by users by a software application called a *web browser* and are published by a software application called a *web server*.

HYPERTEXT:

Hypertext is text displayed on a computer display or other electronic devices with references (hyperlinks) to other text that the reader can immediately access. Hypertext documents are interconnected by hyperlinks, which are typically activated by a mouse click, key press set or by touching the screen. Apart from text, the term "hypertext" is also sometimes used to describe tables, images, and other presentational content formats with integrated hyperlinks. Hypertext is one of the key underlying concepts of the World Wide Web, where Web pages are often written in the Hypertext Markup Language (HTML). As implemented on the Web, hypertext enables the easy-to-use publication of information over the Internet.

The definition of **hypertext** is a word or words that contain a link to a website. An **example** of **hypertext** is the word "Facebook" that links to the Facebook page. Your Dictionary definition and usage **example.**

Hypertext can be just one word, set of words, a sentence, a complete paragraph, image, photo, or included in a video. **Hypertext** allows the content creator to provide easy to use connections to other locations that the reader or viewer might find **useful** or interesting.

Hypertext was **important** because it presented two fundamental changes in the storage and retrieval of data. The first was the capability to move rapidly from one part of a document to another by means of an associative link.

Below are four of the existing forms of hypertext:

- Axial hypertexts are the most simple in structure. ...
- Arborescent hypertexts are more complex than the axial form. ...
- Networked hypertexts are more complex still than the two previous forms of hypertext. ...
- Layered hypertext consists of two layers of linked pages.

Function of Hypertext:

HTTP is a communications protocol which is used to connect to Web servers on the Internet or on a local network (intranet). Its primary **function** is to establish a connection with the server and send HTML pages back to the user's browser.

Hypertext systems allow users to move rapidly and flexibly between information sources of various kinds. They are able to learn faster, find information quicker, or put stored knowledge to use more effectively.

HTTP:

The **Hypertext Transfer Protocol** (**HTTP**) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser.

HTTP functions as a request–response protocol in the client–server computing model. A web browser, for example, may be the *client* and an application running on a computer hosting a website may be the *server*. The client submits an HTTP *request* message to the server. The server, which provides *resources* such as HTML files and other content, or performs other functions on behalf of the client, returns a *response* message to the client. The response contains completion status information about the request and may also contain requested content in its message body.

Stands for "Hypertext Transfer Protocol." **HTTP** is the protocol used to transfer data over the web. It is part of the Internet protocol suite and defines commands and services used for transmitting webpage data. **HTTP uses** a server-client model. A client, for example, may be a home computer, laptop, or mobile device.

HTTP is a connectionless text based protocol. Clients (web browsers) send requests to web servers for web elements such as web pages and images. After the request is serviced by a server, the connection between client and server across the Internet is disconnected. A new connection must be made for each request.

The primary or most-commonly-used **HTTP** verbs (or **methods**, as they are properly called) are POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations, respectively. ... Of those less-frequent **methods**, OPTIONS and HEAD are used more often than others.

Purpose of **HTTP** Protocol:

The name hypertext transfer protocol refers to **HTTP's** role in transmitting website data across **the** internet. ... **The purpose** of **the HTTP** protocol is to provide a standard way for web browsers and servers to talk to each other.

Difference between HTTP and HTTPS:

HTTP by default operates on port 80 whereas **HTTPS** by default operates on port 443. **HTTP** transfers data in plain text while **HTTPS** transfers data in cipher text (encrypt text). **HTTP** is fast as compared to **HTTPS** because **HTTPS** consumes computation power to encrypt the communication channel.

Is HTTP a Web service?
A **web service** is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a **web service**. ... **Web services** are built on top of open standards such as TCP/IP, **HTTP**, Java, HTML, and XML.

URL:

URL stands for Uniform Resource Locator, and is used to specify addresses on the World Wide Web. A **URL** is the fundamental network identification for any resource connected to the web (e.g., hypertext pages, images, and sound files).

A URL (Uniform Resource Locator) is a unique identifier used to locate a resource on the internet. It is also referred to as a web address. URLs consist of multiple parts -- including a protocol and domain name -- that tell a web browser how and where to retrieve a resource.

End users use URLs by typing them directly into the address bar of a browser or by clicking a hyperlink found on a webpage, bookmark list, in an email or from another application.

How is a URL structured?
The URL contains the name of the **protocol** needed to access a resource, as well as a resource name. The first part of a URL identifies what protocol to use as the primary access medium. The second part identifies the **IP address** or **domain name** -- and possibly subdomain -- where the resource is located.

URL protocols include **HTTP** (Hypertext Transfer Protocol) and **HTTPS** (HTTP Secure) for web resources, mail to for email addresses, FTP for files on a File Transfer Protocol (**FTP**) **server**, and **telnet** for a session to access remote computers. Most URL protocols are followed by a colon and two forward slashes; "mail to" is followed only by a colon.

Optionally, after the domain, a URL can also specify:

- a path to a specific page or file within a domain;
- a network port to use to make the connection;
- a specific reference point within a file, such as a named anchor in an HTML file; and
- a query or search parameters used -- commonly found in URLs for search results.

Importance of a URL design:

URLs can only be sent over the Internet using the **ASCII** character-set. Because URLs often contain non-ASCII characters, the URL must be converted into a valid ASCII format. URL encoding replaces unsafe ASCII characters with a "%" followed by two hexadecimal digits. URLs cannot contain spaces.

URL examples:

When designing URLs, there are different theories about how to make the syntax most usable for readers and archivists. For example, in the URL's path, dates, authors, and topics can be included in a section referred to as the "slug." Consider, for example, the URL for this definition:

https://searchnetworking.techtarget.com/definition/URL

Look past the protocol (identified as HTTPS) and the permalink (searchNetworking.com) and we see the path includes the path (definition) and the title of the definition (URL).

Additionally, some URL designers choose to put the date of the post, typically, as (YYYY/MM/DD).

Parts of a URL:
Using the URL https://whatis.techtarget.com/search/query?q=URL as an example, components of a URL can include:

- **The protocol or scheme**. Used to access a resource on the internet. Protocols include http, https, ftps, mailto and file. The resource is reached through the domain name system (DNS) name. In this example, the protocol is https.
- **Host name or domain name**. The unique reference the represents a webpage. For this example, whatis.techtarget.com.
- **Port name.** Usually not visible in URLs, but necessary. Always following a colon, port 80 is the default port for web servers, but there are other options. For example, :port80.
- **Path**. A path refers to a file or location on the web server. For this example, search/query.
- **Query**. Found in the URL of dynamic pages. The query consists of a question mark, followed by parameters. For this example, ?.
- **Parameters**. Pieces of information in a query string of a URL. Multiple parameters can be separated by ampersands (&). For this example, q=URL.
- **Fragment**. This is an internal page reference, which refers to a section within the webpage. It appears at the end of a URL and begins with a hashtag (#). Although not in the example above, an example could be #history in the URL https://en.wikipedia.org/wiki/Internet#History.

HTML/XML:

HTML is an abbreviation for HyperText Markup Language. **XML** stands for eXtensible Markup Language. **HTML** was designed to display data with focus on how data looks. **XML** was designed to be a software and hardware independent tool used to transport and store data, with focus on what data is.

HTML vs XML:

HTML: HTML (**Hyper Text Markup Language**) is used to create web pages and web applications. It is a markup language. By HTML we can create our own static page. It is used for displaying the data not to transport the data. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. A markup language is used to define the text document within tag which defines the structure of web pages. This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly.

Example:

```
<!DOCTYPE html>
<html>
<head>
   <title>GeeksforGeeks</title>
</head>
<body>
   <h1>GeeksforGeeks</h1>
   <p>A Computer Science portal for geeks</p>
</body>
</html>
```

Output:

GeeksforGeeks

A Computer Science portal for geeks

XML: XML (**eXtensible Markup Language**) is also used to create web pages and web applications. It is dynamic because it is used to transport the data not for displaying the data. The design goals of XML focus on simplicity, generality, and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary data structures such as those used in web services.

Example:

```
<?xml version = "1.0"?>
<contactinfo>
   <address category = "college">
      <name>G4G</name>
      <College>Geeksforgeeks</College>
      <mobile>2345456767</mobile>
   </address>
</contactinfo>
```

Output:

G4G

Geeksforgeeks

2345456767

HTML	XML
HTML stands for **Hyper Text Markup Language.**	XML stands for **eXtensible Markup Language.**
HTML is static.	XML is dynamic.
HTML is a markup language.	XML provides framework to define markup languages.
HTML can ignore small errors.	XML does not allow errors.
HTML is not Case sensitive.	XML is Case sensitive.
HTML tags are predefined tags.	XML tags are user defined tags.
There are limited number of tags in HTML.	XML tags are extensible.
HTML does not preserve white spaces.	White space can be preserved in XML.
HTML tags are used for displaying the data.	XML tags are used for describing the data not for displaying.
In HTML, closing tags are not necessary.	In XML, closing tags are necessary.

FOLLOWING WEBSITES ARE REFERRED FOR ALL THE CONCEPTS COVERED IN THIS DOCUMENT

Courtesy:

1. ***www.geeksforgeeks.org***
2. ***www.tutorialspoint.com***
3. ***www.javapoint.com***
4. ***www.programiz.com***