Java For Android

1. Introduction to Java
   1. What is java =>

Java is a popular programming language, created in 1995.It is owned by Oracle, and more than 3 billion devices run Java.

It is used for:

* Mobile applications (specially Android apps)
* Desktop applications
* Web applications
* Web servers and application servers
* Games
* Database connection
* And much, much more!

1.2 Features of Java=>

1. [Simple](https://www.javatpoint.com/features-of-java#Simple)
2. [Object-Oriented](https://www.javatpoint.com/features-of-java#Object-Oriented)
3. [Portable](https://www.javatpoint.com/features-of-java#Portable)
4. [Platform independent](https://www.javatpoint.com/features-of-java#Platform-independent)
5. [Secured](https://www.javatpoint.com/features-of-java#Secured)

1.3 History =>

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was best suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". [Java](https://www.javatpoint.com/java-tutorial)

was developed by James Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc

1.4 Advantage and Disadvantages of java =>

Advantages:

**1. Simple**

Java is a simple programming language since it is easy to learn and easy to understand. Its syntax is based on C++, and it uses automatic garbage collection; therefore, we don't need to remove the unreferenced objects from memory. Java has also removed the features like explicit pointers, operator overloading, etc., making it easy to read and write.

**2. Object-Oriented**

Java uses an object-oriented paradigm, which makes it more practical. Everything in Java is an object which takes care of both data and behaviour. Java uses [object-oriented concepts](https://www.javatpoint.com/java-oops-concepts)

like [object, class, inheritance, encapsulation, polymorphism and abstraction.](https://www.javatpoint.com/object-and-class-in-java" \l "object)

**3. Secured**

Java is a secured programming language because it doesn't use Explicit pointers. Also, Java programs run inside the virtual machine sandbox.

[JRE also provides a](https://www.javatpoint.com/java-jre)[class loader, which is used to load the class into JVM dynamically.](https://www.javatpoint.com/classloader-in-java)

[[https://www.javatpoint.com/jvm-java-virtual-machine](https://www.javatpoint.com/java-jre)](https://www.javatpoint.com/jvm-java-virtual-machine) [It separates the class packages of the local file system from the ones that are being imported from the network.](https://www.javatpoint.com/java-jre)

**4. Robust**

Java is a robust programming language since it uses strong memory management. We can also handle exceptions through the Java code. Also, we can use type checking to make our code more secure. It doesn't provide explicit pointers so that the programmer cannot access the memory directly from the code.

**5. Platform independent**

Java code can run on multiple platforms directly, I.e., we need not compile it every time. It is right once, runs anywhere language (WORA) which can be converted into byte code at the compile time. The byte code is a platform-independent code that can run on multiple platforms.

**6. Multi-Threaded**

Java uses a multi-threaded environment in which a bigger task can be converted into various threads and run separately. The main advantage of multi-threading is that we need not provide memory to every running thread.

### Disadvantages

**1. Performance**

Java needs to be interpreted during runtime, which allows it to run on every operating system, but it also makes it perform slower than the languages like [C](https://www.javatpoint.com/c-programming-language-tutorial)

and [C++](https://www.javatpoint.com/cpp-tutorial)

. On the other hand, the C++ program needs to be compiled on each operating system, directly to binary and therefore runs faster.

**2. Memory consumption**

Java program consumes more memory since it runs on top of Java virtual machine.

**3. Cost**

Java programming language is a bit costly due to its higher processing and memory requirements. We need better hardware to run the Java program.

**4. Less machine interactive**

Java lacks when it comes to interacting directly with machines, making it less viable for the software that needs to run quickly and run directly with the machine, as explicit pointers are also missing in Java.

**5. Garbage collection**

Java provides automatic garbage collection that cannot be controlled by the programmer. It doesn't provide the methods like delete() and free() to free the memory.

1.5 Paradigms of java =>

1. Technical introduction
   1. Introduction to JDK, JRE & JVM =>

JDK

What is JDK?

JDK is a software development environment used for making applets and Java applications. The full form of JDK is Java Development Kit. Java developers can use it on Windows, macOS, Solaris, and Linux. JDK helps them to code and run Java programs. It is possible to install more than one JDK version on the same computer.

## Why use JDK?

Here are the important reasons of using JDK:

* JDK contains tools required to write Java programs, and JRE to execute them.
* It includes a compiler, Java application launcher, Applet viewer, etc.
* Compiler converts code written in Java into byte code.
* Java application launcher opens a JRE, loads the necessary class, and executes its main method.

## Features of JDK

Here are theimportant features of JDK:

* It enables you to handle multiple extensions in a single catch block.
* JDK includes all features that JRE has.
* It contains development tools such as a compiler, debugger, etc.
* JDK provides the environment to develop and execute Java source code.
* It can be installed on Windows, Unix, and Mac operating systems.
* Diamond operator can be used in specifying a generic type interface instead of writing the exact one.

## JRE

## What is JRE?

JRE is a piece of a software which is designed to run other software. It contains the class libraries, loader class, and JVM. In simple terms, if you want to run Java program you need JRE. If you are not a programmer, you don’t need to install JDK, but just JRE to run Java programs. Though, all JDK versions comes bundled with Java Runtime Environment, so you do not need to download and install the JRE separately in your PC. The full form of JRE is Java Runtime Environment.

## Why use JRE?

Here are the important reasons of using JRE:

* JRE contains class libraries, JVM, and other supporting files. It does not contain any tool for Java development like a debugger, compiler, etc.
* It uses important package classes like math, swing etc, util, lang, awt, and runtime libraries.
* If you have to run Java applets, then JRE must be installed in your system.

## Features of JRE

Here are theimportant features of JRE:

* Java Runtime Environment is a set of tools using which the JVM actually runs.
* JRE contains deployment technology, including Java Web Start and Java Plug-in.
* Developers can easily run the source code in JRE, but he/she cannot write and compile the Java program.
* It includes integration libraries like Java Database Connectivity (JDBC), Remote Method Invocation (RMI), Java Naming and Directory Interface (JNDI), and more.
* JRE has JVM and Java Hotspot virtual machine client.

## JVM

## What is JVM?

JVM is an engine that provides a runtime environment to drive the Java Code or applications. It converts Java bytecode into machine language. JVM is a part of Java Run Environment (JRE). It cannot be separately downloaded and installed. To install JVM, you need to install JRE. The full form of JVM is Java Virtual Machine.

In many other programming languages, the compiler produces machine code for a specific system. However, Java compiler produces code for a virtual machine which is called as JVM.

Why JVM?

Here are the important reasons of using JVM:

* JVM provides a platform-independent way of executing Java source code.
* It has numerous libraries, tools, and frameworks.
* Once you run Java program, you can run on any platform and save lots of time.
* JVM comes with JIT(Just-in-Time) compiler that converts Java source code into low-level machine language. Hence, it runs more faster as a regular application.

## Features of JVM

Here are the important features of JVM:

* It enables you to run applications in a cloud environment or in your device.
* Java Virtual Machine converts byte code to the machine-specific code. It provides basic java functions like memory management, security, garbage collection,

and more.

* JVM runs the program by using libraries and files given by Java Runtime Environment.
* JDK and JRE both contain Java Virtual Machine.
* It can execute the java program line by line hence it is also called as interpreter.
* JVM is easily customizable for example, you can allocate minimum and maximum memory to it.
* It is independent from hardware and the operating system. So, you can write a java program once and run anywhere.
  1. Running first programme =>

class Hello

{

static public void main(String as[])

{

System.out.println ("Welcome to java world");

}

}

## Steps to Compile and Run your first Java program--

Step 1: Open a text editor and write the code as above.

Step 2: Save the file as Hello.java

Step 3: Open command prompt and go to the directory where you saved your first java program assuming it is saved in C drive.

Step 4: Type javac Hello.java and press Return(Enter KEY) to compile your code. This command will call the Java Compiler asking it to compile the specified file. If there are no errors in the code the command prompt will take you to the next line.

Step 5: Now type java Hello on command prompt to run your program.

Step 6: You will be able to see Hello world program printed on your command prompt.

* 1. Explanation of first program

class : class keyword is used to declare classes in Java

public : It is an access specifier. Public means this function is visible to all.

static : static is again a keyword used to make a function static. To execute a static function you do not have to create an Object of the class. The main() method here is called by JVM, without creating any object for class.

void : It is the return type, meaning this function will not return anything.

main : main() method is the most important method in a Java program. This is the method which is executed, hence all the logic must be inside the main() method. If a java class is not having a main() method, it causes compilation error.

String[] args : This represents an array whose type is String and name is args. We will discuss more about array in Java Array section.

System.out.println : This is used to print anything on the console like *printf* in C language.

1.5 .java and .class difference =>

the files which contain java bytecode which is produced by the Java compiler is called .class file

the file That contains java source code given is called .java

1.6 keyword introduction =>

A Java keyword is one of 50 reserved terms that have a special function and a set definition in the [Java](https://www.theserverside.com/definition/Java) programming language. The fact that the terms are reserved means that they cannot be used as identifiers for any other program elements, including [classes](https://whatis.techtarget.com/definition/class), subclasses, variables, methods and [objects](https://searchapparchitecture.techtarget.com/definition/object).

Keywords in the Java programming language

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | strictfb | volatile |
| const | float | native | super | while |

1. Data Types

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](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

d\w primitive and non primitive data types

primitives –

* Already defined in java
* Cannot used to call method to perform certains operations.
* Always have a value.
* Starts with lowercase letters.
* Size of primitive type depends on data types.

Non -primitive

* They are created by the programmer but they are not defined by java .
* Can used to call methods to perform certain operations.
* Value can be null.
* Starts with a upper case letter.
* Have all the same size.
  1. Declarations of Data Types=>

int mynum=5;

float mynum=5.668f;

char myletter=’d’;

boolean mybool=true;

String mytext = “hello”;

* 1. Defining of Variable =>

It is a type of data container that stores the data values during java program execution.

Example – int x =100;

Int is data type ,x is variable name and 100 is value.

Every variables is assigned data type which designates the type and quantity of value it can hold.

Types of variables – local, instance, static.

Local variable – variables that are declared inside the body of a method.

Instance variable – defined without the Static keyword. They are defined outside a method declaration. They are object specific.

Static variable – variables are initialized only once, at the start of program execution. These variables should be initialized of any instance variables.

Example-

Class add {

Static int a=1; // static

int data =2; //instance

void method()

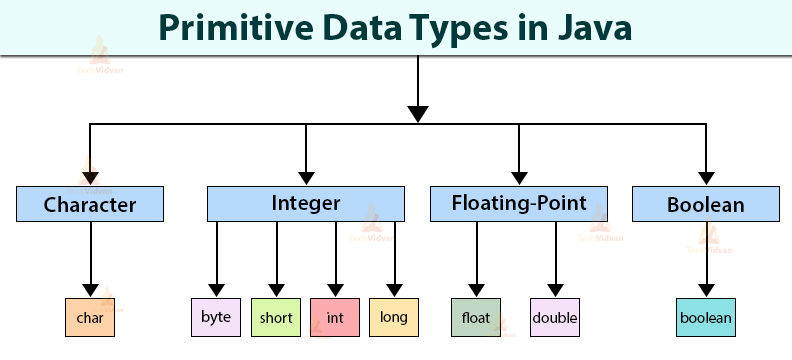
{

int b =3; // local

}

}

* 1. Implementation of DataTypes



Integer type-

* Byte- store whole number from -128 to 127. This can be used instead of int or other integer type to save memory when you are certain that the value will be with in -128 to 127.

Byte mynum=100;

* Short – it can be store whole number from -32768 to 32768;

short mynum=100;

* Int – it is preffed data type when we create variables with a numeric values. stores whole numbe from -2147483648 to 2147483648.

int mynum=100000;

* Long – this is used when int is not large enough to store the value. You should end the value with an “L”.

long mynum=1500000000;

floating point type-

you should use a floating point type whenever you need a number with a decimal.

* Float- float mynum=5.75f;
* Double- double mynum =13.31;

d/w float and double

float double

the precision of float is only six or double variables have a precision of about 15

seven decimal digits. Digits .

takes 4 bytes space. takes 8 bytes space.

Boolean – it is declared with the Boolean with the Boolean keyword and can only take the values true and false

Example - boolean x = true;

boolean y=false;

sout(y);

input is false.

Boolean values are mostly used for conditional testings.

Characters- used to store a single characters. Characters must be surrounding by single quotes, like ‘a’, ’A’.

Example char x=’a’;

Sout(x);

Example2 you can use ASCII values to display certain characters.

Char myvar1=65,myvar2=66;

Sout(myvar1);

Sout(myvar);

Input is A and B .

Strings - used to stores a sequence of characters (text).It must be surrounded by double quotes.

Example- String a= “ hii”

Sout(a);

Input is hii .

1.4 Size of Different Data Types

Char – 1 byte

Short – 2 byte

Int – 4 byte

Long – 8 bytes

Long long – 8 bytes

Float – 4 bytes

Double - 8 bytes

Long double – 16 bytes

1. Operators
   1. Arithmetic Operators- it is used to perform arithmetic operations on variables and data .

Operators operations

+ addition

- substractions

\* multiply

/ divide

% modules

Implementations –

Class Main{

Public void static main(strings[] args)

{

Int a =12,b=5;

Sout(“a+b=” (a+b));

Sout(“a-b=” (a-b));

Sout(“a\*b=” (a\*b));

Sout(“a/b=” (a/b));

Sout(“a%b=” (a%b));

}

Output –

a+b = 17

a-b = 7

a\*b = 60

a/b = 2

a%b = 2

* 1. Assignment Operator –

Used to assign the values to variables in java .

Ex- int age; age 5;

Operators example equivalent to

= a=b; a=b;

+= a+=b; a=a+b;

-= a-=b; a=a-b;

\*= a\*=b; a=a\*b;

/= a/=b; a=a/b;

%= a%=b; a=a%b;

Implementations-

Int a =4;

Int var;

Var=a;

Sout(“var using “ + var );

Var+==a;

Sout(“var using+=: “ + var);

Var\*=a;

Sout(“var using \*= : “ + var);

Output –

Var using =: 4

Var using+=: 8

Var using \*= : 32

* 1. Relational Operators-

Used to check the relationship between two operands.

Operators description example return

== is equal to 3==5 false

!= not equal to 3!=5 true

> greater than 3>5 false

< less than 3<5 true

>= greater than equal to 3>=5 false

<= less than equal to 3<=5 true

Implementation-

int a=7, b=11;

sout(“a is” +a+” b is “ +b);

sout(a==b) //false

sout(a!=b) //true

sout(a>b) //false

sout(a<b) //true

sout(a>=b) //false

sout(a<=b) //true

* 1. Logical Operators –

It is used to check whether an expression is true or false. They are used in decision making.

Operator example meaning

&& exp 1 && exp 2 true(only if both exp 1 and exp 2)

|| exp 1 || exp 2 true(if either exp 1 or exp 2 )

! !exp true(if exp is false and vice versa)

And-

A B result

F F FALSE

F T FALSE

T F FALSE

T T TRUE

OR-

A B result

F F FALSE

F T TRUE

T F TRUE

T T TRUE

NOT-

A result

T F

F T

Implementations-

Class Main{

public static void main(String[] args)

sout((5>3) && (8>5)); //true

sout((5>3) && (8<5)); //false

sout((5<3) && (8>5)); //true

sout((5>3) && (8<5)); //true

sout((5<3) && (8<5)); //false

sout(!(5==3)); //true

sout(!(5>3)); //false

* 1. Unary Operators – it is used with only one operand.

Operator meaning

+ unary plus=>not necessary to use since number are positive without using it.

- unary minus=>invert the sign of an expression.

++ increment=>increment value by 1.

-- decrement=>decrement value by 2.

! logical complement operator=>invert the value of a Boolean.

Implementation-

Class Main{

public static void main(String[] args)

{

int a=12, b=12;

int result1,result2;

sout(“value of a “ +a);

result1 = ++a;

sout(“after increment “ +result1);

sout(“value of b “ +b);

result2=--b;

sout(“after decrement: “ +result2);

}

}

Output-

Value of a: 12

After increment: 13

Value of b: 12

After decrement : 11

* 1. Ternary Operator-

Operator meaning

&& conditional AND

|| uconditional OR

? : ternary (shortand for if-then-else statement)

Syntax –

resultValue=testConditionStatement ? value1:value2;

explaination of syntax-

result value- this is the variable which gets value assigned.

testConditionStatement- gets evaluated which returns Boolean value

i.e true or false

value 1- if test condition statement get evaluated as ‘true’, then value1 gets assigned to result value

value2- if test condition statement get evaluated as ‘false’,then value2 get assigned to result value.

* 1. Bitwise Operators & Shift Operators –

In Java, an operator is a symbol that performs the specified operations. In this section, we will discuss only the bitwise operator and its types with proper examples.

Types of Bitwise Operator

There are six types of the bitwise operator in Java:

* Bitwise AND
* Bitwise exclusive OR
* Bitwise inclusive OR
* Bitwise Compliment
* Bit Shift Operators

Operator symbol uses

Bitwise AND & op1 &op2

Exclusive OR ^ op1 ^ op2

Inclusive OR | op1 | op2

Compliment ~ ~op1

Left shift << op1<<op2

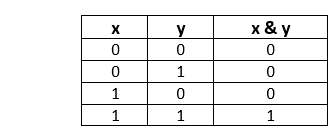
Right shift >> op1>>op2

Unsigned right >>>op>>> number of place to shift

Shift operator

### Bitwise AND (&)

It is a binary operator denoted by the symbol **&**. It returns 1 if and only if both bits are 1, else returns 0.



public static void main(String[] args)

{

int x = 9, y = 8;

// bitwise and

// 1001 & 1000 = 1000 = 8

System.out.println("x & y = " + (x & y));

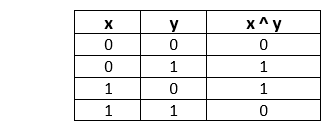
}

Output –

x & y =8

### Bitwise exclusive OR (^)

It is a binary operator denoted by the symbol **^** (pronounced as caret). It returns 0 if both bits are the same, else returns 1.



public static void main(String[] args)

{

int x = 9, y = 8;

// bitwise XOR

// 1001 ^ 1000 = 0001 = 1

System.out.println("x ^ y = " + (x ^ y));

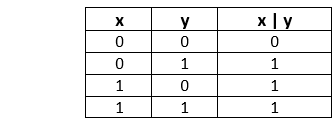
}

Output-

x^ y =1

### Bitwise inclusive OR (|)

It is a binary operator denoted by the symbol **|** (pronounced as a pipe). It returns 1 if either of the bit is 1, else returns 0.



public static void main(String[] args)

{

int x = 9, y = 8;

// bitwise inclusive OR

// 1001 | 1000 = 1001 = 9

System.out.println("x | y = " + (x | y));

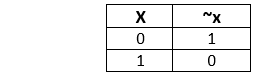
}

Output –

X| y =9

### Bitwise Complement (~)

It is a unary operator denoted by the symbol ~ (pronounced as the tilde). It returns the inverse or complement of the bit. It makes every 0 a 1 and every 1 a 0.



public static void main(String[] args)

{

int x = 2;

// bitwise compliment

// ~0010= 1101 = -3

System.out.println("~x = " + (~x));

}

Output –

~x =-3

### Bit Shift Operators

Shift operator is used in shifting the bits either right or left. We can use shift operators if we divide or multiply any number by 2. The general format to shift the bit is as follows:

1. variable << or >> number of places to shift;

For example, if a=10

1. a>>2; //shifts two bits
2. a>>4; //shifts 4 bits

Java provides the following types of shift operators:

* Signed Right Shift Operator or Bitwise Right Shift Operator
* Unsigned Right Shift Operator
* Signed Left Shift Operator or Bitwise Left Shift Operator

### Signed Right Shift Operator (>>)

The signed right shift operator shifts a bit pattern of a number towards the **right** with a specified number of positions and fills 0. The operator is denoted by the symbol **>>.** It also preserves the leftmost bit (sign bit). If **0** is presented at the leftmost bit, it means the number is **positive**. If **1** is presented at the leftmost bit, it means the number is **negative**.

In general, if we write a>>n, it means to shift the bits of a number toward the right with a specified position (n). In the terms of mathematics, we can represent the signed right shift operator as follows:

b= a>>n -> b=a/2n (2 ki power n )

**Example: Apply the signed right shift operator with specified positions 4 if x = 256 and x = -256.**

**If x = 256**

256 >> 4

256/24 = **16**

If x = -256

-256 >> 4

-256/24 = **-16**

In the above example, we have observed that after shifting the operator 256 converted into 16 and -256 converted into -16.

Let's create a Java program and implement the left shift operator.

**SignedRightShiftOperatorExample.java**

public class SignedRightShiftOperatorExample

{

public static void main(String args[])

{

int x = 50;

System.out.println("x>>2 = " + (x >>2));

}

}

Output –

x>>2= 12

### **Signed Left Shift Operator (<<)**

The signed left shift operator (<<) shifts a bit pattern to the left. It is represented by the symbol **<<.** It also preserves the leftmost bit (sign bit). It does not preserve the sign bit.

In general, if we write a<<n, it means to shift the bits of a number toward the left with specified position (n). In the terms of mathematics, we can represent the signed right shift operator as follows:

B= a>>n -> b=a\*(2n)

**Example 1: What will be the result after shifting a<<3. The value of a is 20.**

Representation of 20 in binary is = 00010100

After performing the left shift operator, we get:

a << 3 = 10100000 (last three bits are the filled bits)

a << 3 = **160**

Let's check the result by using the formula.

20 << 3

20\*23= 20\*8 = **160**

### Unsigned Right Shift Operator (>>>)

It shifts a zero at the leftmost position and fills 0. It is denoted by the symbol **>>>.** Note that the leftmost position after >> depends on the sign bit. It does not preserve the sign bit.

**Example: If a=11110000 and b=2, find a>>>b?**

a >>> b = 11110000 >>> 2 = **00111100**

public static void main(String args[])

{

int x = 20;

System.out.println("x>>>2 = " + (x >>>2));

}

Output-

x>>>2 =5

## Difference between >> and >>> operator

Both >> and >>> are used to shift the bits towards the right. The difference is that the >> preserve the sign bit while the operator >>> does not preserve the sign bit. To preserve the sign bit, you need to add 0 in the MSB.

public static void main(String args[])

{

byte x, y;

x=10;

y=-10;

System.out.println("Bitwise Left Shift: x<<2 = "+(x<<2));

System.out.println("Bitwise Right Shift: x>>2 = "+(x>>2));

System.out.println("Bitwise Zero Fill Right Shift: x>>>2 = "+(x>>>2));

System.out.println("Bitwise Zero Fill Right Shift: y>>>2 = "+(y>>>2));

}

Output-

Bitwise Left Shift: x<<2 = 40

Bitwise Right Shift: x>>2 = 2

Bitwise Zero Fill Right Shift: x>>>2 = 2

Bitwise Zero Fill Right Shift: y>>>2 = 1073741821

1. Java I/O Packages

3.1 Java Scanner ,Print classes-

Scanner-

Scanner class in Java is found in the java.util package. Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.

The Java Scanner class is widely used to parse text for strings and primitive types using a regular expression. It is the simplest way to get input in Java. By the help of Scanner in Java, we can get input from the user in primitive types such as int, long, double, byte, float, short, etc.

The Java Scanner class provides nextXXX() methods to return the type of value such as nextInt(), nextByte(), nextShort(), next(), nextLine(), nextDouble(), nextFloat(), nextBoolean(), etc. To get a single character from the scanner, you can call next().charAt(0) method which returns a single character.

Example-

import java.util.\*;

public class ScannerExample {

public static void main(String args[]){

          Scanner in = new Scanner(System.in);

          System.out.print("Enter your name: ");

          String name = in.nextLine();

          System.out.println("Name is: " + name);

          in.close();

          }

}

Output-

Enter your name:

Java

Name is : java

3.2 FileOutputStream-

java FileOutputStream is an output stream used for writing data to a [file](https://www.javatpoint.com/java-file-class).

Example-

import java.io.FileOutputStream;  
  
public class OutputStream {  
  
  
 public static void main(String args[]){  
 try{  
 FileOutputStream fout=new FileOutputStream("D:\\notepad.txt");  
 fout.write(65);  
 fout.write(66);  
 fout.write(67);  
 fout.close();  
 System.*out*.println("success...");  
 }catch(Exception e)  
 {  
 System.*out*.println(e);}  
  
 }  
}

Output-

Success….

// Check text file

Writing data is done like

ABC //

3.3 FileInputStream-

Java FileInputStream class obtains input bytes from a [file](https://www.javatpoint.com/java-file-class).

It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use [FileReader](https://www.javatpoint.com/java-filereader-class) class.

Example-

import java.io.FileInputStream;  
  
public class InputStreamReadAllChar {  
  
 public static void main(String args[]){  
 try{  
 FileInputStream fin=new FileInputStream("D:\\notepad.txt");  
 int i=0;  
 while((i=fin.read())!=-1){  
 System.*out*.print((char)i);  
 }  
 fin.close();  
 }catch(Exception e){System.*out*.println(e);}  
  
 }  
}

output-

ABC

3.4 ByteArrayOutputStream-

Java ByteArrayOutputStream class is used to **write common data** into multiple files.

Example-

package com.javatpoint;

import java.io.\*;

public class DataStreamExample {

public static void main(String args[])throws Exception{

      FileOutputStream fout1=new FileOutputStream("D:\\f1.txt");

      FileOutputStream fout2=new FileOutputStream("D:\\f2.txt");

      ByteArrayOutputStream bout=new ByteArrayOutputStream();

      bout.write(65);

      bout.writeTo(fout1);

      bout.writeTo(fout2);

      bout.flush();

      bout.close();//has no effect

      System.out.println("Success...");

     }

    }

Output-

Success…

3.5 ByteArrayInputStream-

The ByteArrayInputStream is composed of two words: ByteArray and InputStream. As the name suggests, it can be used to read byte [array](https://www.javatpoint.com/array-in-java) as input stream.

In this stream, the data is read from a byte array.

Example-

package com.javatpoint;

import java.io.\*;

public class ReadExample {

  public static void main(String[] args) throws IOException {

    byte[] buf = { 35, 36, 37, 38 };

    // Create the new byte array input stream

    ByteArrayInputStream byt = new ByteArrayInputStream(buf);

    int k = 0;

    while ((k = byt.read()) != -1) {

      //Conversion of a byte into character

      char ch = (char) k;

      System.out.println("ASCII value of Character is:" + k + "; Special character is: " + ch);

    }

  }

}

Output-

ASCII value of character is 35; special character is: #

ASCII value of character is 36; special character is: $

ASCII value of character is 37; special character is: %

ASCII value of character is 38; special character is: &

1. Conditional Statements-

You can use these conditions to perform different actions for different decisions.

Java has the following conditional statements:

* Use if to specify a block of code to be executed, if a specified condition is true
* Use else to specify a block of code to be executed, if the same condition is false
* Use else if to specify a new condition to test, if the first condition is false
* Use switch to specify many alternative blocks of code to be executed

2.1 if –

Use the if statement to specify a block of Java code to be executed if a condition is true.

In the example below, we test two values to find out if 20 is greater than 18. If the condition is true, print some text:

lint x=20;

int y=18;

if(x> y)

{

System.out.println(“20 is greater than 18”)

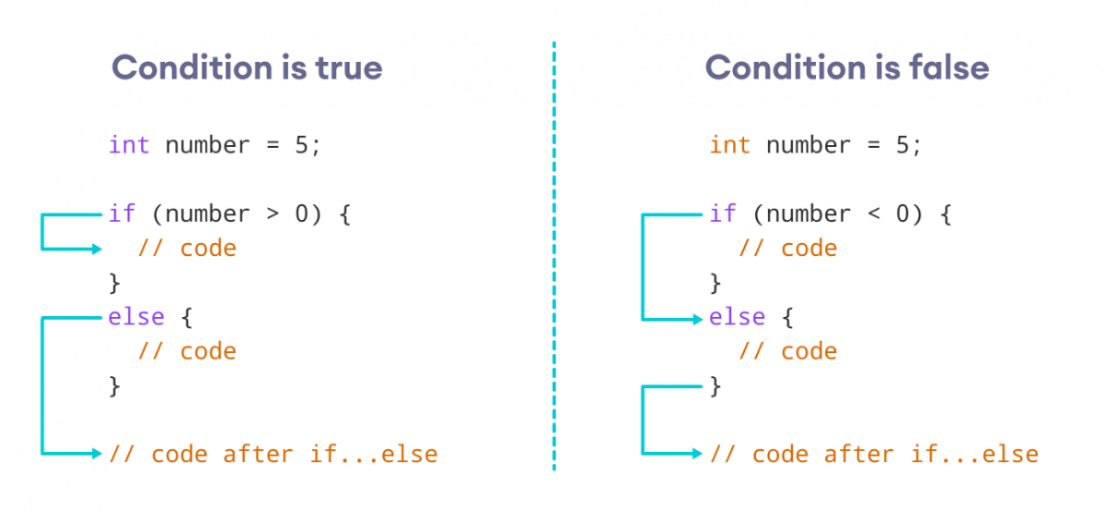
}

Output –

20 is greater than 18

2.2 if-else-

Use the else statement to specify a block of code to be executed if the condition is false.



int time=20;

if(time<18)

{

System.out.println(“good day”);

}

else

{

System.out.println(“good evening”);

}

Output-

Good evening

2.3 if-else-if-

int number =0;

//check number is greater than zero

If(number > 0)

{

System.out.println(“the number is positive”)

}

// number is less than than 0

Else if ( number < 0 )

{

System.out.println(“the number is negative”)

}

// if both condition is false

Else {

System.out.println(“the number is zero”)

}

}

Output –

The number is zero.

2.3 Nested if-else-

In Java, it is also possible to use if..else statements inside an if...else statement. It's called the nested if...else statement.

Class Main{

public static void main(String[] args)

{

//declaring double type variables

Double n1 =-1.0,n2=4.5,n3=-5.3,largest;

//checks if n1 is greater than or equals to n2

If(n1 >=n2) {

// if..else statement inside the if block

// check if n1 is greater than or equal to n3

if(n1 >= n3)

{

largest =n1;}

else{

largest =n3;}

} else{

// if…else statement inside else block

// checks if n2 is greater than or equal to n3

If(n2 >=n3)

{

Largest =n2;}

else{

largest =n3;

}

}

System.out.println(“largest number:”+largest);

}

}

1. Loops

2.1 Simple For Loop-

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to be executed a specific number of times.

A for loop is useful when you know how many times a task is to be repeated.

Flow chart-



Here is the flow of control in a for loop −

* The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables and this step ends with a semi colon (;).
* Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop will not be executed and control jumps to the next statement past the for loop.
* After the body of the for loop gets executed, the control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank with a semicolon at the end.
* The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

Example-

For(int x=10;x<20;x=x+1)

{

System.out.print(x);

System.out.print(“\n”);

}

Output-

Value of x=10

Value of x=11

Value of x=12

Value of x=13

Value of x=14

Value of x=15

Value of x=16

Value of x=17

Value of x=18

Value of x=19

2.2 For each loop-

We use **Java foreach loop**, also called **Enhanced for loop** to traverse through an array or collection. Using a for-each loop, we can easily iterate through the array, array list, or any collection. As the name suggests, we can traverse through every element in an array or collection.

(basically sply designed to print array)

String[] lang={“java”,”c”,”c++”,”php”};

For(String names:lang)[

System.out.println(names);

}

}

Output-

Java

C

C++

Php

2.3 Laballed for loop-

A label is a valid variable name that denotes the name of the loop to where the control of execution should jump. To label a loop, place the label before the loop with a colon at the end. Therefore, a loop with the label is called a labeled loop.

In layman terms, we can say that label is nothing but to provide a name to a loop. It is a good habit to label a loop when using a nested loop. We can also use labels with continue and break statements.

Example-

public static void main(String[] args)

{

Int I,j;

//outer loop

Outer:

//label

For(i=1;i<=5,i++)

{

System.out.println();

Inner: //inner loop

For(j=1;j<=10;j++)

{

System.out.print(j+” ”);

If(j==9)

Break inner;

}

}

}

}

Output-

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9

2.4 While loop-

A while loop statement in Java programming language repeatedly executes a target statement as long as a given condition is true.

## Syntax

The syntax of a while loop is –

While(Boolean expression)

{

//statements

}

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non zero value.

When executing, if the *boolean\_expression* result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true.

When the condition becomes false, program control passes to the line immediately following the loop.

FLOW CHART-



Example-

public static void main(String[] args)

{

Int x=10;

While(x < 20)

{

System.out.print(“value of x :” +x);

x++;

System.out.print(“\n”);

}

}

Output-

Value of x: 10

Value of x: 11

Value of x: 12

Value of x: 13

Value of x: 14

Value of x: 15

Value of x: 16

Value of x: 17

Value of x: 18

Value of x: 19

2.5 Do while loop-

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

## Syntax

Following is the syntax of a do...while loop –

do

{

//statements

}

While(Boolean expression);

Notice that the Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.

If the Boolean expression is true, the control jumps back up to do statement, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

FLOW CHART-



Example-

public static void main(String[] args)

{

Int x=10;

do

{

System.out.print(“value of x: ”+x);

x++;

System.out.print(“\n”);

}

while(x < 20);

}

}

Output –

Value of x: 10

Value of x: 11

Value of x: 12

Value of x: 13

Value of x: 14

Value of x: 15

Value of x: 16

Value of x: 17

Value of x: 18

Value of x: 19

2.6 Switch loop-

Scanner input=new Scanner(System.in);

int selection = input.nextInt();

while (selection<4)

{

switch(selection){

case 1:

System.out.println("Please enter amount");

double amount=input.nextDouble(); //object of scanner class

break;

case 2:

System.out.println("Enter ID number");

break;

case 3:

System.out.println("Enter amount to be credited");

break;

}

System.out.println("1. Transfer\n2.Check balance\n3.Recharge");

}

If I run this code, the output is as follows:

1

Please enter amount

2000

1. Transfer

2.Check balance

3.Recharge

Please enter amount

2

1. Transfer

2.Check balance

3.Recharge

Please enter amount

1. Array

3.1 What is Array-

Normally, an array is a collection of similar type of elements which has 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 size of 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.

3.2 Syntax of Array-

Example-

String[] cars={“volvo”, “BMW”, “ford”, “mazda”};

Cars[0]= “Opel”;

System.out.print(cars[0]);

Output-

Opel

3.3 One Dimensional Array-

### Using Scanner

1) Read the array length as sc.nextInt() and store it in the variable len and declare an array int[len].

**2)** To store elements in to the array for i=0 to i<length of an array read the element using sc.nextInt() and store the element at the index a[i].

**3)** Display the elements of an array for loop iterates from i=0 to i<length of an array print the array element a[i].

Example-

int len;

Scanner sc=new Scanner(System.in);

System.out.println(“enter array length”);

len=sc.nextInt[];

int a[]=new int[len]; //declaration

System.out.print(“ enter”+len+ “ elements to store in array: \n”);

For(int i=0;i<len;i++)

{

a[i]=sc.nextInt();

}

System.out.print(“elements in array are: \n”);

For(int j=0;j<len;j++)

{

System.out.print(a[j]+ “ ”);

}}}

**Output:**

**Enter array length:**

**4**

**Enter 4 length of array:**

**1**

**2**

**3**

**4**

**Elements of array are-**

**1 2 3 4**

3.4 Two Dimensional Array-

1. Read the row length, column length of an array using sc.nextInt() method of Scanner class.

2) Declare the array with the dimension row, column.

3) for i=0 to i<row for j=0 to j<column sc.nextInt() reads the entered number and insert the element at a[i][j].

Example-

Scanner sc =new Scanner(System.in);

System.out.println(“enter row”);

int row=sc.nextInt();

System.out.println(“enter column”);

int column=sc.nextInt();

int a[][]= new int [row][column];

System.out.print(“enter”+row\*column+ “ elements to store iin array: \n”);

for(i=o;i<row;i++)

{

for(j=o;j<column;j++)

{

a[i][j]=sc.nextInt();

}

}

System.out.print(“elements in array are: \n”);

For(int i=0;i<row;i++)

{

For(int j =0;j<column;j++)

{

System.out.println(“row[”+i+ “];column [”+j+ “]:”+a[i][j]);

}}}}

|  |  |  |
| --- | --- | --- |
|  |  |  |

Output:

Enter row-

2

Enter column-

3

Enter 6 elements to store in array-

1

2

3

4

5

6

Elements in array are :

Row[0] column[0] :1

Row[0] column[1] :2

Row[0] column[2] :3

Row[1] column[0] :4

Row[1] column[1] :5

Row[1] column[2] :6

3.5 Jagged-

Jagged array is a multidimensional array where member arrays are of different size.

Example-

Int[][] a=new int[2][];

a[0]=new int[3];

a[1]=new int[2];

int counter=0;

//initializing array

for( int row =0;row<a.length;row++)

{

for(int col=0;col<a.length;col++)

{

a[row][col]=counter;

}

}

//printing array

For(int row=0;row<a.length;row++)

{

System.out.println();

For(int col=0;col<a[row].length;col++)

{

System.out.print(a[row][col]+ “ ”);

}}}}

Output-

0 1 2

3 4 5 6

3.6 Continue-

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.

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

5

6

7

8

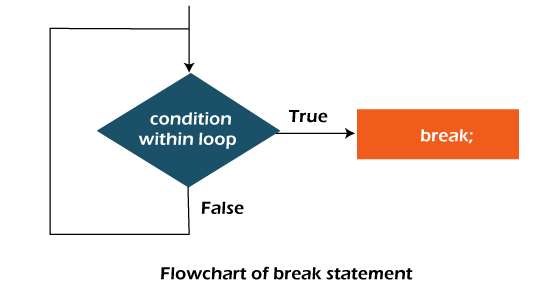
9

10

3.7 Break-

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](https://www.javatpoint.com/java-switch) statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner 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

1. String

4.1 What is String-

In [Java](https://www.javatpoint.com/java-tutorial), string is basically an object that represents sequence of char values. An [array](https://www.javatpoint.com/array-in-java) 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);

4.2 Ways of Defining String-

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

4.3 Mutable & Immutable String-

Mutable string

The mutable objects are objects whose value can be changed after initialization. We can change the object's values, such as field and states, after the object is created. For example, [Java.util.Date](https://www.javatpoint.com/java-util-date)**,**[StringBuilder](https://www.javatpoint.com/StringBuilder-class)**,**[StringBuffer](https://www.javatpoint.com/StringBuffer-class), etc.

When we made a change in existing mutable objects, no new object will be created; instead, it will alter the value of the existing object. These object's classes provide methods to make changes in it.

The Getters and Setters ( get() and set() methods ) are available in mutable objects. The Mutable object may or may not be thread-safe.

public class JtpExample {

    private String s;

    JtpExample(String s) {

    this.s = s;

    }

    public String getName() {

    return s;

    }

    public void setName(String coursename) {

    this.s = coursename;

    }

    public static void main(String[] args) {

    JtpExample obj = new JtpExample("JavaTpoint");

    System.out.println(obj.getName());

// Here, we can update the name using the setName method.

    obj.setName("Java Training");

    System.out.println(obj.getName());

    }

    }

Output-

javaTpoint

java Training

### Immutable string -

### The immutable objects are objects whose value can not be changed after initialization. We can not change anything once the object is created. For example, **primitive objects** such as [int](https://www.javatpoint.com/int-keyword-in-java), [long](https://www.javatpoint.com/long-keyword-in-java), [float](https://www.javatpoint.com/float-keyword-in-java), [double](https://www.javatpoint.com/double-keyword-in-java), **all**[legacy classes](https://www.javatpoint.com/legacy-class-in-java)**,**[Wrapper class](https://www.javatpoint.com/wrapper-class-in-java)**,**[String class](https://www.javatpoint.com/methods-of-string-class), etc.

In a nutshell, immutable means unmodified or unchangeable. Once the immutable objects are created, its object values and state can not be changed.

Only Getters ( get() method) are available not Setters ( set() method) for immutable objects.

Example-

      private final String s;

        JtpExample1(final String s) {

        this.s = s;

        }

        public final String getName() {

        return s;

         }

         public static void main(String[] args) {

        JtpExample obj = new JtpExample("Core Java Training");

        System.out.println(obj.getName());

      }

      }

Output-

Core Java Training

4.4 String Buffer & String Builder-

Java StringBuffer class is used to create mutable (modifiable) String objects. The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.

Example of string buffer- // class append() method

public static void main(String args[]){

StringBuffer sb=new StringBuffer("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java

}

Output-

Hello java

Java StringBuilder class is used to create mutable (modifiable) String. The Java

StringBuilder class is same as StringBuffer class except that it is non-synchronized.

Example for string builder- // append() method by string builder..

public static void main(String args[]){

StringBuilder sb=new StringBuilder("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java

}

Output-

Hello java

4.5 String comparison-

There are three ways to compare String in Java:

1. By Using equals() Method-

The String class equals() method compares the original content of the string. It compares values of string for equality. String class provides the following two methods:

Skip Ad

* **public boolean equals(Object another)** compares this string to the specified object.
* **public boolean equalsIgnoreCase(String another)** compares this string to another string, ignoring case.

1. class Teststringcomparison1{

 public static void main(String args[]){

   String s1="Sachin";

   String s2="Sachin";

   String s3=new String("Sachin");

   String s4="Saurav";

   System.out.println(s1.equals(s2));//true

   System.out.println(s1.equals(s3));//true

   System.out.println(s1.equals(s4));//false

 }

**Output:**

**True**

**True**

**False**

1. By Using == Operator –

The == operator compares references not values.

Example-

public static void main(String args[]){

   String s1="Sachin";

   String s2="Sachin";

   String s3=new String("Sachin");

   System.out.println(s1==s2);//true (because both refer to same instance)

   System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)

 }

Output-

True

false

1. By compareTo() Method-

The String class compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two String objects. If:

* s1 == s2 : The method returns 0.
* s1 > s2 : The method returns a positive value.
* s1 < s2 : The method returns a negative value.

Example-

 public static void main(String args[]){

    String s1="Sachin";

   String s2="Sachin";

   String s3="Ratan";

   System.out.println(s1.compareTo(s2));//0

   System.out.println(s1.compareTo(s3));//1(because s1>s3)

   System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )

 }

Output-

0

1

-1

4.6 Method of String-

1. Functional Programming

4.1 Defining static method-

The **static keyword** in [Java](https://www.javatpoint.com/java-tutorial) is used for memory management mainly. We can apply static keyword with [variables](https://www.javatpoint.com/java-variables), methods, blocks and [nested classes](https://www.javatpoint.com/java-inner-class). The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)-

If you declare any variable as static, it is known as a static variable.

* The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
* The static variable gets memory only once in the class area at the time of class loading.

### Advantages of static variable

It makes your program **memory efficient** (i.e., it saves memory).

Example-

//Java Program to demonstrate the use of static variable

class Student{

   int rollno;//instance variable

   String name;

   static String college ="ITS";//static variable

   //constructor

   Student(int r, String n){

   rollno = r;

   name = n;

   }

   //method to display the values

   void display (){System.out.println(rollno+" "+name+" "+college);}

}

//Test class to show the values of objects

public class TestStaticVariable1{

 public static void main(String args[]){

 Student s1 = new Student(111,"Karan");

 Student s2 = new Student(222,"Aryan");

 //we can change the college of all objects by the single line of code

 //Student.college="BBDIT";

 s1.display();

 s2.display();

 }

}

Output-

111 Karan ITS

222 Aryan ITS

1. Method (also known as a class method)-

If you apply static keyword with any method, it is known as static method.

* A static method belongs to the class rather than the object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* A static method can access static data member and can change the value of it.

Example-

//Java Program to demonstrate the use of a static method.

class Student{

     int rollno;

     String name;

     static String college = "ITS";

     //static method to change the value of static variable

     static void change(){

     college = "BBDIT";

     }

     //constructor to initialize the variable

     Student(int r, String n){

     rollno = r;

     name = n;

     }

     //method to display values

     void display(){System.out.println(rollno+" "+name+" "+college);}

}

//Test class to create and display the values of object

public class TestStaticMethod{

    public static void main(String args[]){

    Student.change();//calling change method

    //creating objects

    Student s1 = new Student(111,"Karan");

    Student s2 = new Student(222,"Aryan");

    Student s3 = new Student(333,"Sonoo");

    //calling display method

    s1.display();

    s2.display();

    s3.display();

    }

}

Output-

111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT

1. Block-

Is used to initialize the static data member.

It is executed before the main method at the time of classloading.

Example-

class A2{

  static{System.out.println("static block is invoked");}

  public static void main(String args[]){

   System.out.println("Hello main");

  }

}

Output-

Static block is invoked

Hello main

4.2 Accessing static methodes-

public class CalculateCube

{

//defining a static method

static void cube()

{

int x=7\*7\*7;

System.out.println("Cube of 7 is: "+x);

}

public static void main(String args[])

{

//calling the static method without using the object of the CalculateCube class

cube();

}

}

Output-

Cube of 7 is: 343

4.3 Non-argumented static method-

No argument is passed.

class Demo

{

//non-static function

void display()

{

System.out.println("A non-static function is called.");

}

//static function

static void show()

{

System.out.println("The static function is called.");

}

}

public class StaticFunctionExample

{

public static void main(String args[])

{

//creating an object of the class A

Demo obj = new Demo();

//calling a the non-static function by using the object of the class

obj.display();

//calling a static function by using the class name

Demo.show();

}

}

Output-

A non static function is called.

The static function is called.

4.4 Argumented static method-

4.5 static variable and classes-

Class variables are also known as static variables, and they are declared outside a method, with the help of the keyword ‘static’.

Example-

public class demo

{

Static int my\_count=2;

public void increment()

{

My\_count++;

}

public static void main(String[] args)

{

demo obj1=new demo();

demo obj2=new demo();

obj1.increment();

obj2.increment();

sout(obj1.my\_count);

sout(obj2.my\_count);

}

}

Output-

4

4

1. Introduction to Oops Concept

5.1 What is Object oriented programming?

**Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects.

5.2 Advantage of Oops-

* OOP is faster and easier to execute
* OOP provides a clear structure for the programs
* OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
* OOP makes it possible to create full reusable applications with less code and shorter development time

5.3 Defining of class and function-

Class/objects-

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

Example –

public class Main

{

Int x=5;

public static void main(String[] args)

{

Main m1=new Main();

sout(m1.x);

}

}

Output-

5

5.4 Functions

5.4.1 Argument Function

5.4.2 Non Argument Function

5.5 Defining of object-

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

5.5 Accessing Function and Variables in classes

Function-block of statement and perform particular task called function.

1. Constructor

4.1 What is constructor-

Same name as the class name called constructor.

No explicit return type.

All classes are constructor ,whether defined or not,because java automatically provides default constructor and that initialize all member to zero.

Once we defined our own constructor, default constructor is longer used.

4.2 Non- Argumented Constructor-

No argument constructor of java does not accept any parameters.

Example-

Public class MyClass

{

Int num;

Public MyClass()

{

Num=100;

}

}

Psvm()

{

MyClass c1=new MyClass();

MyClass c2=new MyClass();

Sout(c1.num + “ ” +c2.num);

}

}

Output-

100 100

4.3 Argumented Constructor-

Also known as parameterised constructor. you will need a constructor that accepts one or more parameter. Parameters are added to a constructor in a same way that they are added to method and declare them inside a parenthesis after the constructor’name.

Example-

Class MyClass

{

Int x;

MyClass(int i)

{

x=i;

}

}

Psvm() {

MyClass c1=new MyClass(10);

MyClass c2=new MyClass(20);

Sout(c1.x + “ ” +c2.x);

}}

Output-

10 20

4.4 Constructor Overloading-

It is a technique of having more than one constructor with different parameter lists.

They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types

(same name of constructor but different argument in every constructor called constructor overloading.)

Example –

Class Student

{

Int id,age;

String name;

Student(int i, String n )

{

id =I;

name=n;

}

Student(int i, String n, int a )

{

id =I;

name=n;

age=a;

}

Void display()

{

Sout(id+ “ ”+ names + “ ”age) ;

}

Psvm()

{

Student s1=new Student(111, “karan” );

Student s2=new Student(222, “kapil”,25 );

S1.display();

S2.display();

}

4.5 Constructor Calling

1. Inheritance

4.1 What is Inheritance-

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object.

### Terms used in Inheritance

* **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
* **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
* **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
* Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Syntax –

Class Subclass-name extends Superclass-name

{

//method and field

}

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

4.2 Advantage of Inheritance-

* The biggest advantage of inheritance is code reusability, since the fields and methods of parent class get's inherited in child class, the child class won't have to create it again. It can access those features from parent class, that is what the code reusability is.
* It also help's to reduce code duplicacy. If inheritance is not used, multiple classes may need to write similar functions/logic in their body.
* Another advantages of inheritance is extensibility, you can add new features or change the existing features easily in subclasses.
* Using inheritance we can achieve runtime polymorphism(method overriding).
* Inheritance makes easy to maintain the code, as the common codes are written at one place.





4.3 Single Level Inheritance-

When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

Example-

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class TestInheritance{

public static void main(String args[]){

Dog d=new Dog();

d.bark();

d.eat();

}}

Output-

Barking….

Eating…..

4.4 Multi Level Inheritance-

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

Example-

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class BabyDog extends Dog{

void weep(){System.out.println("weeping...");}

}

class TestInheritance2{

public static void main(String args[]){

BabyDog d=new BabyDog();

d.weep();

d.bark();

d.eat();

}}

Output-

Weeping…

Barking….

Eating…

4.5 Hierarchical Inheritance-

When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

Example –

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class Cat extends Animal{

void meow(){System.out.println("meowing...");}

}

class TestInheritance3{

public static void main(String args[]){

Cat c=new Cat();

c.meow();

c.eat();

//c.bark();//C.T.Error

}}

Output-

Meowing…

Eating….

4..6 Multiple Inheritance-

Multiple inheritance a feature of some object-oriented programming languages in which a class or an object inherits characteristics and properties from more than one parent class or object. This is contrary to the single inheritance property, which allows an object or class to inherit from one specific object or class.

Example-

// first inheritance class

Class Event

{

public void start();

}

// second inheritance class

Class Sport

{

public void play();

}

// inheritance class

Class Hockey extends sports ,play

{

public void show();

}

Public class Tester

{

Psvm()

{

Hockey hockey=new Hockey()

{

public void start()

{

sout(“start event”)

}

public void play()

{

sout(“start sports”)

}

public void show()

{

sout(“start hockey”)

}

};

hockey.start();

hockey.play();

hockey.show();

output-

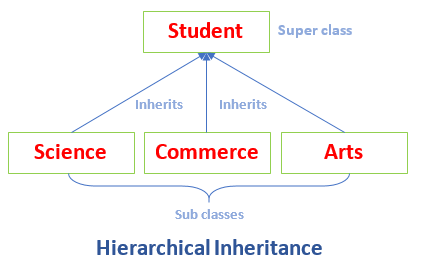
start event

play sports

show hockey

4.7 Hybrid Inheritance-

If a number of classes are derived from a single base class, it is called **hierarchical inheritance**.



Example-

//parent class

class Student

{

public void methodStudent()

{

System.out.println("The method of the class Student invoked.");

}

}

class Science extends Student

{

public void methodScience()

{

System.out.println("The method of the class Science invoked.");

}

}

class Commerce extends Student

{

public void methodCommerce()

{

System.out.println("The method of the class Commerce invoked.");

}

}

class Arts extends Student

{

public void methodArts()

{

System.out.println("The method of the class Arts invoked.");

}

}

public class HierarchicalInheritanceExample

{

public static void main(String args[])

{

Science sci = new Science();

Commerce comm = new Commerce();

Arts art = new Arts();

//all the sub classes can access the method of super class

sci.methodStudent();

comm.methodStudent();

art.methodStudent();

}

}

Output-

The method of the class student invoked.

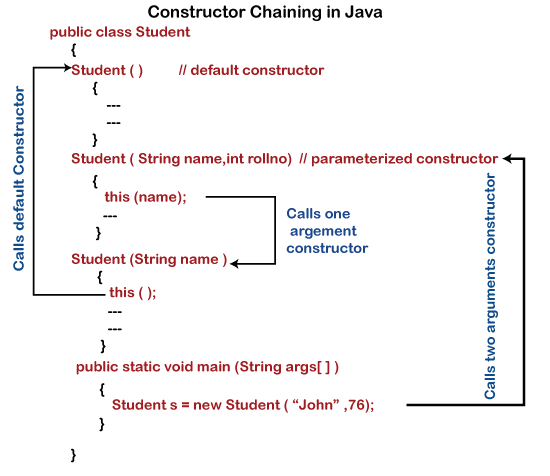
The method of the class student invoked.

The method of the class student invoked.

4.8 Constructor Chaining-

In Java, **constructor chaining** is a sequence of invoking [constructors](https://www.javatpoint.com/java-constructor) upon initializing an object. It is used when we want to invoke a number of constructors, one after another by using only an instance. In this section, we will discuss **constructor chaining in Java in detail with proper examples.** Let's have a quick look at **what is a constructor in Java.**

In constructor chain, a constructor is called from another constructor in the same class this process is known as **constructor chaining.** It occurs through inheritance. When we create an instance of a derived class, all the constructors of the inherited class (base class) are first invoked, after that the constructor of the calling class (derived class) is invoked.



### Rules of Constructor Chaining

* An expression that uses **this** keyword must be the first line of the constructor.
* **Order** does not matter in constructor chaining.
* There must exist at least one constructor that does not use **this**

### Constructor Calling form another Constructor

The calling of the constructor can be done in two ways:

* **By using this() keyword:** It is used when we want to call the current class constructor within the same class.
* **By using super() keyword:** It is used when we want to call the superclass constructor from the base class.

**Example-**

public class ConstructorChain

{

//default constructor

ConstructorChain()

{

this("Javatpoint");

System.out.println("Default constructor called.");

}

//parameterized constructor

ConstructorChain(String str)

{

System.out.println("Parameterized constructor called");

}

//main method

public static void main(String args[])

{

//initializes the instance of example class

ConstructorChain cc = new ConstructorChain();

}

}

Output-

Parameterized constructor called

Default constructor called

1. Access Modifiers

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

4.1 Private-

The private access modifier is accessible only within the class.

Example-

class A{

private int data=40;

private void msg(){System.out.println("Hello java");}

}

public class Simple{

 public static void main(String args[]){

   A obj=new A();

   System.out.println(obj.data);//Compile Time Error

   obj.msg();//Compile Time Error

   }

}

4.2 Public-

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Example-

// save by A.java

package pack;

public class A{

public void msg(){System.out.println("Hello");}

}

//save by B.java

package mypack;

import pack.\*;

class B{

  public static void main(String args[]){

   A obj = new A();

   obj.msg();

  }

}

Output-

Hello

4.3 default-

If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

Example-

//save by A.java

package pack;

class A{

  void msg(){System.out.println("Hello");}

}

//save by B.java

package mypack;

import pack.\*;

class B{

  public static void main(String args[]){

   A obj = new A();//Compile Time Error

   obj.msg();//Compile Time Error

  }

}

4.4 Protected -

The protected access modifier is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifer.

Example-

//save by A.java

package pack;

public class A{

protected void msg(){System.out.println("Hello");}

}

//save by B.java

package mypack;

import pack.\*;

class B extends A{

  public static void main(String args[]){

   B obj = new B();

   obj.msg();

  }

}

Output-

Hello

1. Abstraction

4.1 What is Abstraction-

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

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

4.2 Advantage of Abstraction-

1. It reduces the complexity of viewing the things.
2. Avoids code duplication and increases reusability.
3. Helps to increase the security of an application or program as only important details are provided to the user.

4.3 What is Abstract Class-

A class which is declared with the abstract keyword is known as an abstract class in [Java](https://www.javatpoint.com/java-tutorial). It can have abstract and non-abstract methods (method with the body).

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have [constructors](https://www.javatpoint.com/java-constructor) and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

4.4 Implementation of Abstract Class-

abstract class Bike{

  abstract void run();

}

class Honda4 extends Bike{

void run(){System.out.println("running safely");}

public static void main(String args[]){

 Bike obj = new Honda4();

 obj.run();

}

}

Output-

Running safely.

4.5 Defining of Interface-

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

There are mainly three reasons to use interface. They are given below.

* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.

Example-

interface printable{

void print();

}

class A6 implements printable{

public void print(){System.out.println("Hello");}

public static void main(String args[]){

A6 obj = new A6();

obj.print();

 }

}

Example-

Hello

1. Polymorphism

3.1 What is polymorphism-

**Polymorphism in Java** is a concept by which we can perform a single action in different ways. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

3.2 Advantages of Polymorphism-

1. It allows programmers to reuse ,evaluate and execute the program ,modules, forms written once. In certain aspects ,they can be repeated .
2. You may use the odd variables name to stock variables of different types of data ,such as int ,float,etc..
3. Polymorphism tends to reduce the pairing of multiple functionality.
4. Method overloading can be extended to builders that allow multiple ways of initializing class objects. it helps you to identify several builders for managing various form of initializations.
5. Method overriding functions along with inheritance without the need for re-compilation to allow code reuse of existing groups.

3.3 Runtime Polymorphism-

**Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Example-

class Bike{

  void run(){System.out.println("running");}

}

class Splendor extends Bike{

  void run()

{

System.out.println("running safely with 60km");

}

  public static void main(String args[]){

    Bike b = new Splendor();//upcasting

    b.run();

  }

}

Output-

Running safetly with 60km

3.3 Compile Time Polymorphism-

Compile time polymorphism or static method dispatch is a process in which a call to an overloading method is resolved at compile time rather than at run time. In this process, we done overloading of methods is called through the reference variable of a class here no need to superclass.

Example-

Public void meth(int value)

{

Sout(“ int values: ”+ values;

)

Public void meth(String name)

{

Sout(“ string value :”+ name);

}

Public void meth(int value,String name)

{

Sout(“ name with value :”+value+ “ ”+name );

}

Psvm()

{

Compile c1=new Compile();

C1.meth(10);

C1.meth(“hello java ”)

C1.meth(20, “java world”)

}

}

Output-

Int value: 10;

String value : hello java

Name with value : 20 java world

* 1. Application of Polymorphism-

Advantages-

It allows programmers to reuse ,evaluate and execute the program ,modules, forms written once. In certain aspects ,they can be repeated .

* 1. You may use the odd variables name to stock variables of different types of data ,such as int ,float,etc..
  2. Polymorphism tends to reduce the pairing of multiple functionality.
  3. Method overloading can be extended to builders that allow multiple ways of initializing class objects. it helps you to identify several builders for managing various form of initializations.
  4. Method overriding functions along with inheritance without the need for re-compilation to allow code reuse of existing groups.

Disadvantages –

* 1. One of the key drawbacks of polymorphisms is that the implementations of polymorphism in code is complicated for developers.
  2. Polymorphisms in run time will lead to the performance problem where the system has to determine which process or variables to invoke so that the performance is effectively diminished as decisions are taken at a time.
  3. The readability of the program is diminished by polymorphisms. In order to define real execution time, one has to recognise the program’s runtime actions.

1. Inner Class

Java inner class or nested class is a class that is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place to be more readable and maintainable.

Additionally, it can access all the members of the outer class, including private data members and methods.

Syntax of inner class

class Java\_Outer\_class{

 //code

 class Java\_Inner\_class{

  //code

 }

}

Advantages –

1. Nested classes represent a particular type of relationship that is it can access all the members (data members and methods) of the outer class, including private.
2. Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.
3. Code Optimization: It requires less code to write.

Need of inner class-

Sometimes users need to program a class in such a way so that no other class can access it. Therefore, it would be better if you include it within other classes.

If all the class objects are a part of the outer object then it is easier to nest that class inside the outer class. That way all the outer class can access all the objects of the inner class.

3.1 Nested Inner class-

3.2 Method Local inner classes

3.3 Anonymous inner class-

Java anonymous inner class is an inner class without a name and for which only a single object is created. An anonymous inner class can be useful when making an instance of an object with certain "extras" such as overloading methods of a class or interface, without having to actually subclass a class.

In simple words, a class that has no name is known as an anonymous inner class in Java. It should be used if you have to override a method of class or interface. Java Anonymous inner class can be created in two ways:

1. Class (may be abstract or concrete).
2. Interface

Example-

abstract class Person{

  abstract void eat();

}

class TestAnonymousInner{

 public static void main(String args[]){

  Person p=new Person(){

  void eat(){System.out.println("nice fruits");}

  };

  p.eat();

 }

}

Output-

Nice fruits

3.3 Static nested classes-

A static inner class is a nested class which is a static member of the outer class. It can be accessed without instantiating the outer class, using other static members. Just like static members, a static nested class does not have access to the instance variables and methods of the outer class.

Example –

Public class outer

{

Static class Nested\_demo

{

Public void my\_method()

{

System.out.println(“ this is my nested class”);

}

}

Public static void main(String[] args )

{

Outer.Nested\_demo nested =new Outer.Nested\_demo();

Nested.my\_method();

}

}

Output-

This is my nested class

1. Other Oops Concepts

3.1 Aggragation-

3.2 Scope of Variable

3.2.1 Global

3.2.2 Local

3.2.3 final, static

3.3 Wrapper Class-

The **wrapper class in Java** provides the mechanism to convert primitive into object and object into primitive.

3.4 Java Recursion-

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

It makes the code compact but complex to understand.

Syntax-

returntype methodname()

{

//code to be executed

methodname(); //calling same method

}

Example-

public class RecursionExample2 {

static int count=0;

static void p(){

count++;

if(count<=5){

System.out.println("hello "+count);

p();

}

}

public static void main(String[] args) {

p();

}

}

Output-

Hello 1

Hello 2

Hello 3

Hello 4

Hello 5

3.5 Conversion-

3.6 Enum Classes-

The Enum in Java is a data type which contains a fixed set of constants.

The enum can be defined within or outside the class because it is similar to a class. The semicolon (;) at the end of the enum constants are optional.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY), directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colours (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.

Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change). The Java enum constants are static and final implicitly. It is available since JDK 1.5.

Enums are used to create our own data type like classes. The enum data type (also known as Enumerated Data Type) is used to define an enum in Java. Unlike C/C++, enum in Java is more *powerful*. Here, we can define an enum either inside the class or outside the class.

Java Enum internally inherits the *Enum class*, so it cannot inherit any other class, but it can implement many interfaces. We can have fields, constructors, methods, and main methods in Java enum.

Points to remember for Java Enum

* Enum improves type safety
* Enum can be easily used in switch
* Enum can be traversed
* Enum can have fields, constructors and methods
* Enum may implement many interfaces but cannot extend any class because it internally extends Enum class.

Example –

class EnumExample1{

//defining enum within class

public enum Season { WINTER, SPRING, SUMMER, FALL }

//creating the main method

public static void main(String[] args) {

//printing all enum

for (Season s : Season.values()){

System.out.println(s);

}

System.out.println("Value of WINTER is: "+Season.valueOf("WINTER"));

ystem.out.println("Index of WINTER is: "+Season.valueOf("WINTER").ordinal());

System.out.println("Index of SUMMER is: "+Season.valueOf("SUMMER").ordinl());

}}

Output-

WINTER

SPRING

SUMMER

FALL

Value of WINTER is : WINTER

Index of WINTER is : 0

Index of SUMMER is : 2

1. Multi-Threading

3.1 What is multithreading-

**Multithreading in**[Java](https://www.javatpoint.com/java-tutorial) is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

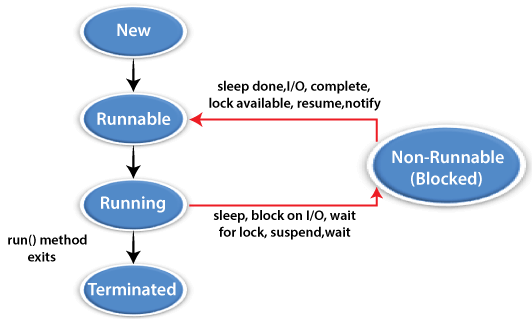
### Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

3.2 Life cycle of thread-



-1) New (Ready to run)

A thread is in New when it gets CPU time.

2) Running

A thread is in a Running state when it is under execution.

3) Suspended

A thread is in the Suspended state when it is temporarily inactive or under execution.

4) Blocked

A thread is in the Blocked state when it is waiting for resources.

5) Terminated

A thread comes in this state when at any given time, it halts its execution immediately.

3.3 Sleeping a thread-

( stop the code execution for limited amount of time…..)

The Java Thread class provides the two variant of the sleep() method. First one accepts only an arguments, whereas the other variant accepts two arguments. The method sleep() is being used to halt the working of a thread for a given amount of time. The time up to which the thread remains in the sleeping state is known as the sleeping time of the thread. After the sleeping time is over, the thread starts its execution from where it has left.

Example –

class TestSleepMetod1 extends Thread{

 public void run(){

  for(int i=1;i<5;i++){

  // the thread will sleep for the 500 milli seconds

    try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}

    System.out.println(i);

  }

 }

 public static void main(String args[]){

  TestSleepMethod1 t1=new TestSleepMethod1();

  TestSleepMethod1 t2=new TestSleepMethod1();

  t1.start();

  t2.start();

 }

}

Output-

1

1

2

2

3

3

4

4

3.4 Naming a thread-

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name, i.e. thread-0, thread-1 and so on. By we can change the name of the thread by using the setName() method.

The syntax of setName() and getName() methods are given below:

public String getName(): is used to return the name of a thread.

public void setName(String name): is used to change the name of a thread.

Example –

class TestMultiNaming1 extends Thread{

  public void run(){

   System.out.println("running...");

  }

 public static void main(String args[]){

  TestMultiNaming1 t1=new TestMultiNaming1();

  TestMultiNaming1 t2=new TestMultiNaming1();

  System.out.println("Name of t1:"+t1.getName());

  System.out.println("Name of t2:"+t2.getName());

  t1.start();

  t2.start();

  t1.setName("hello");

  System.out.println("After changing name of t1:"+t1.getName());

 }

}

Output-

Name of t1: thread-0

Name of t2:thread-1

After changing name of t1: hello

Running…

Running…..

3.5 Thread Priority

3.6 Garbage Collection-

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

### Advantage of Garbage Collection

* It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
* It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

Example –

public class TestGarbage1{

 public void finalize(){System.out.println("object is garbage collected");}

 public static void main(String args[]){

  TestGarbage1 s1=new TestGarbage1();

  TestGarbage1 s2=new TestGarbage1();

  s1=null;

  s2=null;

  System.gc();

 }

}

Output –

Object is garbage collected

Object is garbage collected

3.7 Thread Synchronization-

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

example of java synchronized method-

class Table{

 synchronized void printTable(int n){//synchronized method

   for(int i=1;i<=5;i++){

     System.out.println(n\*i);

     try{

      Thread.sleep(400);

     }catch(Exception e){System.out.println(e);}

   }

 }

}

class MyThread1 extends Thread{

Table t;

MyThread1(Table t){

this.t=t;

}

public void run(){

t.printTable(5);

}

}

class MyThread2 extends Thread{

Table t;

MyThread2(Table t){

this.t=t;

}

public void run(){

t.printTable(100);

}

}

public class TestSynchronization2{

public static void main(String args[]){

Table obj = new Table();//only one object

MyThread1 t1=new MyThread1(obj);

MyThread2 t2=new MyThread2(obj);

t1.start();

t2.start();

}

}

Output-

5

10

15

20

25

100

200

300

400

500

3.8 Inter thread communication-

Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of Object class:

* wait()
* notify()
* notifyAll()

1. Exceptional Handling

3.1 What is Exception-

**Dictionary Meaning:** Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

20.7M

458

C++ vs Java

Next

Stay

## What is Exception Handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

### Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. Checked Exception- The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.
2. Unchecked Exception- The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.
3. Error- Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Example-

public class JavaExceptionExample{

  public static void main(String args[]){

 try{

       //code that may raise exception

 int data=100/0;

  }

catch(ArithmeticException e){System.out.println(e);}

    //rest code of the program

  System.out.println("rest of the code...");

  }

}

Output-

Exception in thread main java.lang.ArithmeticException :/ by zero rest of the code…….

3.2 Try-catch block-

try block-Java try block is used to enclose the code that might throw an exception. It must be used within the method.

If an exception occurs at the particular statement in the try block, the rest of the block code will not execute. So, it is recommended not to keep the code in try block that will not throw an exception.

Java try block must be followed by either catch or finally block.

Syntax –

try{

//code that may throw an exception

}

catch(Exception\_class\_Name ref){}

catch block - Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception ( i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.

The catch block must be used after the try block only. You can use multiple catch block with a single try block.

Example –

Exception occurs

public class TryCatchExample1

{

    public static void main(String[] args)

 {

          int data=50/0; //may throw exception

          System.out.println("rest of the code");

       }

 }

Output

Exception in thread “main”java.lang.ArithmeticException :/ by zero

Example –

Exception handling with try catch block –

public class TryCatchExample2 {

    public static void main(String[] args) {

        try

        {

        int data=50/0; //may throw exception

        }

            //handling the exception

        catch(ArithmeticException e)

        {

            System.out.println(e);

        }

        System.out.println("rest of the code");

    }

}

Output-

Java.lang.ArithmeticException : / by zero

Rest of the code

3.3 Multiple Catch Block-

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

* At a time only one exception occurs and at a time only one catch block is executed.
* All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.



Example –

public class MultipleCatchBlock1

 {

  public static void main(String[] args)

 {

          Try

{

                int a[]=new int[5];

                 a[5]=30/0;

               }

               catch(ArithmeticException e)

                  {

                   System.out.println("Arithmetic Exception occurs");

                  }

               catch(ArrayIndexOutOfBoundsException e)

                  {

                   System.out.println("ArrayIndexOutOfBounds Exception occurs");

                  }

               catch(Exception e)

                  {

                   System.out.println("Parent Exception occurs");

                  }

               System.out.println("rest of the code");

    }

}

Output-

Arithmetic Exception occurs

Rest of the code

3.4 Nested try catch block-

In Java, using a try block inside another try block is permitted. It is called as nested try block. Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.

For example, the **inner try block** can be used to handle **ArrayIndexOutOfBoundsException** while the **outer try block** can handle the **ArithemeticException** (division by zero).

### Why use nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Example –

public class NestedTryBlock{

 public static void main(String args[]){

 //outer try block

  try{

  //inner try block 1

    try{

     System.out.println("going to divide by 0");

     int b =39/0;

   }

    //catch block of inner try block 1

    catch(ArithmeticException e)

    {

      System.out.println(e);

    }

    //inner try block 2

    try{

    int a[]=new int[5];

    //assigning the value out of array bounds

     a[5]=4;

     }

    //catch block of inner try block 2

    catch(ArrayIndexOutOfBoundsException e)

    {

       System.out.println(e);

    }

    System.out.println("other statement");

  }

  //catch block of outer try block

  catch(Exception e)

  {

    System.out.println("handled the exception (outer catch)");

  }

  System.out.println("normal flow..");

 }

}

Output –

Going to divide by 0

Java.lang.ArithmeticException :/ by zero

Java.lang.ArrayIndexOutOfBoundsExcepton: index 5 out of bounds for length 5

Other statement

Normal flow..

3.5 Throw, Throws and Finally-

Throw-

The throw keyword is used to throw an exception explicitly. Only object of Throwable class or its sub classes can be thrown. Program execution stops on encountering throw statement, and the closest catch statement is checked for matching type of exception.

Example

Class test

{

Static void avg()

{

Try

{

Throw new AeithmeticException(“exception”);

}

Catch(ArithmeticException e)

{

System.out.println(“exception caught”);

}

}

public static void main(String[] args)

{

Avg();

}

}

Throws-

The throws keyword is used to declare the list of exception that a method may throw during execution of program. Any method that is capable of causing exceptions must list all the exceptions possible during its execution, so that anyone calling that method gets a prior knowledge about which exceptions are to be handled. A method can do so by using the throws keyword.

Example-

Class test

{

Static void check() throws ArithmeticException

{

System.out.println(“inside check function”);

Throw new ArithmeticException(“demo”);

}

public static void main(String[] args)

{

Try

{

Check();

}

Catch(ArithmeticException e)

{

System.out.println(“exception caught”+e);

}

}

}

}

Output-

Inside check function

Caughtjava.lang.ArithmeticException:demo

Difference between throw and throws -

|  |  |
| --- | --- |
| throw | throws |
| throw keyword is used to throw an exception explicitly. | throws keyword is used to declare an exception possible during its execution. |
| throw keyword is followed by an instance of Throwable class or one of its sub-classes. | throws keyword is followed by one or more Exception class names separated by commas. |
| throw keyword is declared inside a method body. | throws keyword is used with method signature (method declaration). |
| We cannot throw multiple exceptions using throw keyword. | We can declare multiple exceptions (separated by commas) using throws keyword. |

Final-

In this example, we are using finally block along with try block. This program throws an exception and due to exception, program terminates its execution but see code written inside the finally block executed. It is because of nature of finally block that guarantees to execute the code.

Example –

Class Exceptiontest

{

Psvm()

{

Int a[]=new int[2];

System.out.println(“out of try ”);

Try

{

Sout(“access invalid element”+a[3]);

}

Finally

{

Sout(“finally is always executed”);

}

}

}

Output-

Out of try

Finally is always executed

Exception in thread main.java.lang.exception array out of bound exception.

3.6 Exception Propagation

1. Collections

3.1 Array List

Java **ArrayList** class uses a dynamic [*array*](https://www.javatpoint.com/array-in-java) for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the java.util package.

Important points to remember-

* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non [synchronized](https://www.javatpoint.com/synchronization-in-java).
* Java ArrayList allows random access because array works at the index basis.
* In ArrayList, manipulation is little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.

Example-

import java.util.\*;

 public class ArrayListExample1{

 public static void main(String args[]){

  ArrayList<String> list=new ArrayList<String>();//Creating arraylist

      list.add("Mango");//Adding object in arraylist

      list.add("Apple");

      list.add("Banana");

      list.add("Grapes");

      //Printing the arraylist object

      System.out.println(list);

 }

}

Output-

[Mango, Apple, Banana, Grapes]

3.3 Hash Map-

Java **HashMap** class implements the Map interface which allows us to store key and value pair, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy to perform operations using the key index like updation, deletion, etc. HashMap class is found in the java.util package.

Points to remember-

* Java HashMap contains values based on the key.
* Java HashMap contains only unique keys.
* Java HashMap may have one null key and multiple null values.
* Java HashMap is non synchronized.
* Java HashMap maintains no order.
* The initial default capacity of Java HashMap class is 16 with a load factor of 0.75

Example-

import java.util.\*;

public class HashMapExample1{

 public static void main(String args[]){

   HashMap<Integer,String> map=new HashMap<Integer,String>();//Creating HashMap

   map.put(1,"Mango");  //Put elements in Map

   map.put(2,"Apple");

   map.put(3,"Banana");

   map.put(4,"Grapes");

   System.out.println("Iterating Hashmap...");

   for(Map.Entry m : map.entrySet()){

    System.out.println(m.getKey()+" "+m.getValue());

   }

}

}

Output-

1 Mango

1. Apple
2. Banana
3. Grapes

3.4 Sets,HashSets-

Sets-The **set** is an interface available in the **java.util** package. The **set** interface extends the Collection interface. An unordered collection or list in which duplicates are not allowed is referred to as a **collection interface**. The set interface is used to create the mathematical set. The set interface use collection interface's methods to avoid the insertion of the same elements

Example-

import java.util.\*;

public class setExample{

    public static void main(String[] args)

    {

        // creating LinkedHashSet using the Set

        Set<String> data = new LinkedHashSet<String>();

        data.add("JavaTpoint");

        data.add("Set");

        data.add("Example");

        data.add("Set");

        System.out.println(data);

    }

}

Output-

[Javatpoint, Set, Example]

Hashsets-

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

* HashSet stores the elements by using a mechanism called hashing.
* HashSet contains unique elements only.
* HashSet allows null value.
* HashSet class is non synchronized.
* HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
* HashSet is the best approach for search operations.
* The initial default capacity of HashSet is 16, and the load factor is 0.75.

Example-

import java.util.\*;

class HashSet1{

 public static void main(String args[]){

  //Creating HashSet and adding elements

     HashSet<String> set=new HashSet();

           set.add("One");

           set.add("Two");

           set.add("Three");

           set.add("Four");

           set.add("Five");

           Iterator<String> i=set.iterator();

           while(i.hasNext())

           {

           System.out.println(i.next());

           }

 }

}

Output-

Five

One

Four

Two

Three