Java

What is Java?

Java is a programming language and a platform. Java is a high level, robust, secured and object oriented programming language.

The History of Java:

The Java is created by Sun microsystems, began a project called “The Green project” in 1991.The goal of the project was to create a new portable programming language. The original name of the language is oak. Later it renamed to java. Java was first release to public in 1995.Java is very popular by its write once and run anywhere method.

The principles of Java:

Simple: It’s very simple and easy to understand because syntax is based on C++.

Object-oriented: It means we organize our software in the form of different types of objects that incorporates both data and behavior. Basic OOPS concepts are: objects, classes, Inheritance, Polymorphism, Abstraction and Encapsulation.

Robust: Robust means Strong. Java uses Strong memory management.

Secure: Java is secured because

We are not using any pointers.

Program runs inside virtual machine sandbox.

Portable: we can carry the java byte code to anywhere.

Multi-threaded: A thread is like separate program, executing concurrently.

Dynamic: we can change the data type in runtime as long as those data types are compatible with each other.

Simple Hello Word:

public class helloword

{

public static void main(String args[])

{

System.out.println(“Hello World”);

}

}

class: keyword is used to declare a class in java.

public: It is an access specifier which represents visibility, it means it is visible to all.

static: if we declare any method as static, it is known as static methods. The core advantage of static method is that there is no need to create object to invoke the static method.

void: Void is the return type of the method, it says it’s not return anything.

main: represents the startup of a program.

System.out.println: used to print a statement to the screen.

Java Virtual Machine:

JVM is an abstract machine. It is specification that provides runtime environment in which java byte code can be executed. It also performs various tasks like

-Loads code.

-Verifies code.

-Executes code.

-Provides runtime environment.

Java Runtime Environment:

It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that jvm uses at runtime.

Garbage Collector in Java:

In java we do not have to allocate and de-allocate memory. It will manage memory automatically for you. The garbage collector is a part of the JVM. There are 2 types of memory, Stack and Heap. If we create an object in java, its referenced are created in a special memory called heap memory.

Stack-The stack memory is faster than heap. Primitive variables depending on their context they might be stored in either stack or heap.

Heap-The heap memory is more dynamic than stack. Complex objects are always stored in heap.

As long as any variable references an object, it’s retained and when all references expire, the object is eligible to be garbage collected.

Types of variables:

Java uses variables to stored data in memory. There are 2 types of variables primitive and complex.

**Primitive variables** are store in the fastest stored memory but there is only small number of data types to store in it. Primitive datatypes includes numeric (which contains integers and floating point decimals), single characters and Boolean values (true/false).

**Complex objects** includes String (any String value longer than a single character must be stored as a complex objects), Dates.

Declaring a primitive variable:

Variables are declared with 3 parts

Data type required. (Once set it can’t change)

Variable name. (Always start with a lower case, which can be either alphabetic or an underscore (\_) but can’t be numeric).

Initial value (optional).

Eg: int aa=10;

Declaring a Complex variable:

Initialization uses new keyword and class constructor.

Eg: Date newDate=new Date();

Where Date-datatype.

newDate-variable name.

Date();-Initial value from constructor.

We can create either local variables or global variables. Variables declared inside a function are called local variables and those declare outside the function are called Global variables.

Numeric Primitive datatypes:

byte -1 Bytes

short -2 Bytes

int -4 Bytes

long -8 Bytes

float -4bytes

double – 8 Bytes

Converting primitive Datatypes:

**Converting upwards:**

Byte->short->int->long->float->double

Eg:

int aa=120;

double ab=aa;

Upward numeric conversion is implicit and reliable.

**Converting downwards:**

Double->float->long->int->short->Byte

Eg:

Double aa=3.99;

int ab=aa;//compile-time error!!

Downward conversion is explicit and can change data.

Types of Operators:

**Assignment Operator:**

int value=10 ;

**Mathematical:**

int newvalue=value+5;(15)

int newvalue=value-5;(5)

int newvalue=value\*5;(50)

int newvalue=value/5;(2)

int newvalue=value%5;(0)

**Incrementing/Decrementing operator:**

intvalue ++;(11)

intvalue --;(9)

**Postfix:**

System.out.println(intvalue ++);

Output:10,New Value:11

**Prefix:**

System.out.println(++ intvalue);

Output:11,New Value:11

**Equality or Relational:**

String s1=”Hello”;

String s2=”Hello”;

If(s1==s2){

System.out.println (“they are match”);

}

else {

System.out.println (“no match”);

}

Output: no match

String s1=”Hello”;

String s2=”Hello”;

If(s1.equals(s2)){

System.out.println (“they are match”);

}

else {

System.out.println (“no match”);

}

Output: they are match

**Conditional:**

Conditional AND:

If(condition1&&condition2){

System.out.println(“true”);

}

It returns true only if both conditions satisfied.

Conditional OR:

If(condition1||condition2){

System.out.println(“true”);

}

It returns true either one of the condition satisfies.

Conditional Statements:

**If-else:**

The simplest conditional statement in java. Returns either a true or false condition.

Syntax:

if(condition)

{

Set of codes

}

else(condition)

{

Set of codes

}

**switch**:

Used to execute the code in multiple conditions.

Syntax:

switch(condition){

case value1:

//code to be executed

break;

case value2:

//code to be executed

break;

default

}

**for:**

for loop iterates the code until the condition is false. There are 3 parts in for loop.

1). Initialization

2). Condition

3) Increment/Decrement

Syntax:

for (initialization; condition; increment/decrement)

{

Set of codes

}

**do-while:**

It iterates the code until condition is false. Here, condition is given after the code.so atleast once the code is executed whether the condition is true or false.

Syntax:

do

{

Code to be executed

}while(condition);

Method Overloading:

If a class has multiple methods by same name but different parameter, it is known as method overloading. Its increases the readability of the program. There are 2 ways to overload the methods in java.

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

Changing the number of arguments:

class calculation{

void sum(int a, int b)

{

System.out.println (a+b);}

void sum(int a,int b,int c)

{

System.out.println (a+b+c);

}

public static void main(String args[])

{

Calculation obj=new Calculation();

obj.sum (10, 10, 10);

obj.sum (20, 20);

}

}

Output:30,40

Changing the data types:

class calculation{

void sum(int a, int b)

{

System.out.println (a+b);}

void sum(double a, double b)

{

System.out.println (a+b);

}

public static void main(String args[])

{

Calculation obj=new Calculation();

obj.sum (10, 10);

obj.sum (20.5, 20.5);

}

}

Output: 20, 41

**Passing by copy or reference:**

Passing by copy:

In this method when a variable is passed to a function, a copy is made.

Passing by reference:

In this method when a variable is passed to a function, the code in the function operates on the original variable.

In java, variables are always passed by copy.

Strings in Java:

Strings which are widely used in java programming are a sequence of characters. In java Strings are objects. Strings are immutable, once you set the value you cannot change it unless you need to use special classes like String Builder class.

**Creating Strings:**

The most direct way to create a string is

Eg: String greeting=”Hello world”;

Eg: char[] helloarray={‘h’,’e’,’l’,’l’,’o’};

String helloString=new String (helloString);

Eg: String

The String class is immutable, so that once it is created a String object cannot be changed.

**String Length:**

It returns the number of characters contained in the string object.

Eg: String data=”Manchester United”;

int length=data. length();

System.out.println (length);

Output:17

**String concatenation:**

It is used to concatenate 2 strings. We can either use concat method or plus(+) sign to perform the operation.

Eg: ”my Name is”.concat(“Appu”);

“Hello,”+”World”

**String comparison:**

It is a method used to compare strings. There are 3 ways to compare strings in java.

1). By equals() method.

2). By == operator.

3). By compareTo() method.

By equals():

It compares the original content of the string.it compares values of strings for quality.

Eg: String s1=”Sachin”;

String s2=”Sachin”;

String s3=new String(“Sachin”);

String s4=”Sourav”;

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

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

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

By == operator:

The == operator compares the references not values.

Eg: : String s1=”Sachin”;

String s2=”Sachin”;

String s3=new String(“Sachin”);

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

System.out.println (s2==s3);//false, because s3 refers to instance created in the nonpool.

By compareTo() method:

compareTo() method compares the values and returns an int which tells, if the values compare less than, equal or greater than. Suppose s1 and s2 are two strings

\*) s1==s2:0

\*) s1>s2:positive value

\*) s1<s2:negative value

**String Builder:**

The StringBuilder class is used to create mutable (modifiable) string. The StringBuilder class is same as StringBuffer class except that it is non-synchronized.

Commonly used StringBuilder constructors are:

1. StringBuilder (): creates an empty string Builder with the initial capacity of 16.
2. StringBuilder (String str): creates a string Builder with the specified string.
3. StringBuilder (int length): creates an empty string Builder with the specified capacity as length.

class A{

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

}

}

**String Parsing:**

String s1=”welcome to California”;

System.out.println (“length of the string: “+s1.length ());//22

Int pos=s1.indexOf (“California”);

System.out.println (“length of the string: “+pos ());//11

String sub=s1.substring(11);

System.out.println (sub); California!

String s2=”Welcome! “;

int len1=s2.length();

System.out.println (len1);//15

String s3=s2.trim();

System.out.println (s3.length());//8

**Exception Handling:**

The exception handling in java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained. Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

An Exception is an abnormal condition. It disrupts the normal flow of a program.

Types of Exceptions:

There are 3 types of exceptions, Checked Exception, Unchecked Exception and Error.

Checked Exception: Checked at compiler time.(I/O Exceptions,SQLExceptions)

Unchecked Exception: Checked at runtime.

(Arithmetic Exception, NullPointerException,ArrayIndexOutOfBoundException)

Errors: It is irrecoverable.(Virtual Memory)

Exception handlers:

There are 5 keywords used in exception handling.

1. try
2. catch
3. throw
4. throws
5. finally

**try block:**

Java try block is used to enclose the code that might throw an exception. It must be used within the method. Java try block must be followed by either catch or finally block.

Syntax: try

{

//code that might thrown an exception

}catch

{

//print the caught exception

}

*Problem without Exception handling:*

public class Testtrycatch1{

public static void main(String args[]){

int data=50/0;//may throw exception

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

}

}

O/p: Exception in thread main java.lang.ArithmeticException:/ by zero

*Problem with Exception handling:*

public class Testtrycatch2{

public static void main(String args[]){

try{

int data=50/0;

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

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

}

}

**Finally:**

Java finally block is a block that is used *to execute important code* such as closing connection, stream etc. Java finally block is always executed whether exception is handled or not. Java finally block must be followed by try or catch block.

Note: If there is any Exception occurs, the compiler will first checks for the finally block, if it founds it will execute that block first.

Case 1:

Let's see the java finally example where **exception doesn't occur**.

class TestFinallyBlock{

public static void main(String args[]){

try{

int data=25/5;

System.out.println(data);

}

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

finally{System.out.println("finally block is always executed");}

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

}

}

Output:5

finally block is always executed

rest of the code...

Case 2:

Let's see the java finally example where **exception occurs and not handled**.

class TestFinallyBlock1{

public static void main(String args[]){

try{

int data=25/0;

System.out.println(data);

}

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

finally{System.out.println("finally block is always executed");}

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

}

}

Output: finally block is always executed

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

Case 3

Let's see the java finally example where **exception occurs and handled**.

public class TestFinallyBlock2{

public static void main(String args[]){

try{

int data=25/0;

System.out.println(data);

}

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

finally {System.out.println("finally block is always executed");}

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

}

}

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

finally block is always executed

rest of the code...

**Throw keyword:**

The Java throw keyword is used to explicitly throw an exception. We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.

Syntax: throw exception

public class TestThrow1{

static void validate(int age){

if(age<18)

throw new ArithmeticException("not valid");

else

System.out.println ("welcome to vote");

}

public static void main(String args[]){

validate (13);

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

}

}

Output: Exception in thread main java.lang.ArithmeticException: not valid

**Throws keyword:**

The Java throws keyword is used to declare an exception. It gives information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

import java.io.\*;

class M{

void method()throws IOException{

throw new IOException("device error");

}

}

class Testthrows4{

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

M m=new M();

m.method();

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

}

}

Output: Runtime Exception

Java Array:

Array is a collection of similar type of elements that have contiguous memory location.

Advantage of Java Array

Code Optimization: It makes the code optimized; we can retrieve or sort the data easily.

Random access: We can get any data located at any index position.

Disadvantage of Java Array

Size Limit: We can store only 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.

**Single Dimensional Array:**

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

class Testarray{

public static void main(String args[]){

int a[]=new int[5];//declaration and instantiation

a[0]=10;//initialization

a[1]=20;

a[2]=70;

a[3]=40;

a[4]=50;

//printing array

for(int i=0;i<a.length;i++)//length is the property of array

System.out.println(a[i]);

}}

Output:10,20,70,40,50

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

int a[]={33,3,4,5};//declaration, instantiation and initialization

class Testarray1{

public static void main(String args[]){

int a[]={33,3,4,5};//declaration, instantiation and initialization

//printing array

for(int i=0;i<a.length;i++)//length is the property of array

System.out.println (a[i]);

}}

Output:33,3,4,5

**Multi-Dimensional Array:**

Syntax to Declare Multidimensional Array in java

dataType[][] dat; (or)

dataType [][]dat; (or)

dataType dat[][]; (or)

dataType []dat[];

Example to instantiate Multidimensional Array in java

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

Example to initialize Multidimensional Array in java

arr[0][0]=1;

arr[0][1]=2;

arr[0][2]=3;

arr[1][0]=4;

arr[1][1]=5;

arr[1][2]=6;

arr[2][0]=7;

arr[2][1]=8;

arr[2][2]=9;

Example of Multidimensional java array

Let's see the simple example to declare instantiate, initialize and print the 2Dimensional array.

class Testarray3{

public static void main(String args[]){

//declaring and initializing 2D array

int arr[][]={{1,2,3},{2,4,5},{4,4,5}};

//printing 2D array

for(int i=0;i<3;i++){

for(int j=0;j<3;j++){

System.out.println (arr[i][j]+" ");

}

System.out.println ();

}

}}

Output:

Output: 1 2 3

2 4 5

4 4 5

# Java ArrayList class:

One main disadvantage of an array is it is not dynamic.ie; we need to define the size of the array in advance. Once declared we can’t change it.Another is we cannot insert the elements in between the array elements. ArrayList come with a solution to this problem. Not only this, there are some other advantages also

Java ArrayList class uses a dynamic array for storing the elements. It extends AbstractList class and implements List interface.

Java ArrayList class can contain duplicate elements.

Java ArrayList class maintains insertion order.

Java ArrayList class is non-synchronized.

Java ArrayList allows random access because array works at the index basis.

In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Example of ArrayList:

public static void main(String args[])

{

 ArrayList<String> al=new ArrayList<String>();//creating arraylist

al.add("Ravi");//adding object in ArrayList

al.add("Vijay");

al.add("Ravi");

al.add("Ajay");

System.out.println(al);

}

Output: [Ravi,Vijay,ravi,Ajay]

HashMap:

Map is an interface; it stores the elements in the form of key-value pair. It extends the abstract MAP & implements the MAP interface.

Syntax: HashMap<K, V> map=new HashMap<K, V>;

Where K=key and V=Value

Methods in HashMap:

1). Value put(key, value);

This method stores the Key-value pair into the HashMap.

2).Value get(obj key);

This method returns the corresponding values per the key.

3).Value remove(object key);

This method removes the key and the corresponding values from the Hashmap.

**ListIterator Class:**

ListIterator Interface is used to traverse the element in backward and forward direction.

Commonly used methods of ListIterator Interface:

1. public boolean hasNext();
2. public Object next();
3. public boolean hasPrevious();
4. public Object previous();

public static void main(String args[]){

ArrayList<String> al=new ArrayList<String>();

al.add ("Amit");

al.add ("Vijay");

al.add ("Kumar");

al.add (1,"Sachin");

System.out.println ("element at 2nd position: "+al.get (2));

ListIterator<String> itr=al.listIterator ();

System.out.println ("traversing elements in forward direction...");

while (itr.hasNext()){

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

}

System.out.println ("traversing elements in backward direction...");

while (itr.hasPrevious()){

System.out.println (itr.previous ());

}

}

}

Output: element at 2nd position: Vijay

Traversing elements in forward direction...

Amit

Sachin

Vijay

Kumar

Traversing elements in backward direction...

Kumar

Vijay

Sachin

Amit

Packages:

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

Syntax: package package\_name;

# Constructor in Java

**Constructor in java** is a special type of method that is used to initialize the object.Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

### Rules for creating java constructor

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

### Types of java constructors

There are two types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

## Java Default Constructor:

|  |
| --- |
| A constructor that has no parameter is known as default constructor. |

### Syntax of default constructor:

<class\_name>(){}

## Example of default constructor

|  |
| --- |
|  |

### What is the purpose of default constructor?

Default constructor provides the default values to the object like 0, null etc. depending on the type.

class Bike1{

Bike1(){System.out.println("Bike is created");}

public static void main(String args[]){

Bike1 b=new Bike1();

}

}

Output: 0 null

0 null

**Java parameterized constructor:**

|  |
| --- |
| Constructors that have parameters are known as parameterized constructor. |

Why use parameterized constructor?

|  |
| --- |
| Parameterized constructor is used to provide different values to the distinct objects. |

Example of parameterized constructor

|  |
| --- |
|  |

class Student4{

    int id;

    String name;

    Student4(int i,String n){

    id = i;

    name = n;

    }

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

    public static void main(String args[]){

    Student4 s1 = new Student4(111,"Karan");

    Student4 s2 = new Student4(222,"Aryan");

    s1.display();

    s2.display();

   }

}

Output:111 karan

222 Aryan

|  |
| --- |
| Java Copy Constructor: There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.  Eg: |

class Student6{

    int id;

    String name;

    Student6(int i,String n){

    id = i;

    name = n;

    }

    Student6(Student6 s){

    id = s.id;

    name =s.name;

    }

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

    public static void main(String args[]){

    Student6 s1 = new Student6(111,"Karan");

    Student6 s2 = new Student6(s1);

    s1.display();

    s2.display();

   }

}

Inheritance:

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

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the **IS-A relationship**, also known as parent-child relationship.

### Why use inheritance in java

* For Method Overriding (so runtime polymorphism can be achieved).
* For Code Reusability.

### Syntax of Java Inheritance

**class** Subclass-name **extends** Superclass-name

{

   //methods and fields

}

The **extends keyword** indicates that you are making a new class that derives from an existing class.

In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

# Method Overriding in Java:

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**. In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

* Method overriding is used to provide specific implementation of a method that is already provided by its super class.
* Method overriding is used for runtime polymorphism

Eg:

class Vehicle{

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

}

class Bike2 extends Vehicle{

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

public static void main(String args[]){

Bike2 obj = new Bike2();

obj.run();

}

Output: Bike is running safely

Super keyword:

The **super** keyword in java is a reference variable that is used to refer immediate parent class object.

## Usage of java super Keyword

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

## 1) super is used to refer immediate parent class instance variable.

***Problem without super keyword***

class Vehicle{

int speed=50;

}

class Bike3 extends Vehicle{

  int speed=100;

  void display(){

   System.out.println(speed);//will print speed of Bike

  }

  public static void main(String args[]){

   Bike3 b=new Bike3();

   b.display();

}

}

Output:100

***Solution by super keyword***

//example of super keyword

class Vehicle{

  int speed=50;

}

class Bike4 extends Vehicle{

  int speed=100;

  void display(){

   System.out.println(super.speed);//will print speed of Vehicle now

  }

  public static void main(String args[]){

   Bike4 b=new Bike4();

   b.display();

}

}

Output:50

## 2) super is used to invoke parent class constructor.

|  |
| --- |
| The super keyword can also be used to invoke the parent class constructor as given below: |

class Vehicle{

  Vehicle(){System.out.println("Vehicle is created");}

}

class Bike5 extends Vehicle{

  Bike5(){

   super();//will invoke parent class constructor

   System.out.println("Bike is created");

  }

  public static void main(String args[]){

   Bike5 b=new Bike5();

}

}

Output: Vehicle is created

Bike is created

## 3) super can be used to invoke parent class method.

|  |
| --- |
| The super keyword can also be used to invoke parent class method. It should be used in case subclass contains the same method as parent class as in the example given below: |

class Person{

void message(){System.out.println("welcome");}

}

class Student16 extends Person{

void message(){System.out.println("welcome to java");}

void display(){

message();//will invoke current class message() method

super.message();//will invoke parent class message() method

}

public static void main(String args[]){

Student16 s=new Student16();

s.display();

}

}

Output: welcome to java

welcome

Typecasting:

It is possible to convert one primitive datatype to another nothing but typecasting.

1. Widening in primitive data types:

Syntax: <RDT><var>=<RDT>(S\_DT\_variable);

Where RDT=required datatype

var=variable

S\_DT\_variable=Source data type variable

Eg: int ii=(int)ch;//char to int

1. Narrowing in primitive datat types:

Syntax:<RDT><variable>=(RDT)(R.variable);

Eg: double d=120.88;

Byte b=(byte)d;

# Interface:

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

The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritances in Java.

## Why use Java interface?

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

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

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();

}

}

# Java I/O Tutorial:

**Java I/O** (Input and Output) is used to process the input and produce the output based on the input.

**FileOutputStream:**

Java FileOutputStream is an output stream for writing data to a file.

Eg:

import java.io.\*;

class Test{

  public static void main(String args[]){

   try{

     FileOutputstream fout=new FileOutputStream("abc.txt");

     String s="Sachin Tendulkar is my favourite player";

 byte b[]=s.getBytes();//converting string into byte array

     fout.write(b);

     fout.close();

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

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

  }

}

**FileInputStream:**

Java FileInputStream class obtains input bytes from a file. It is used for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader.

Eg:

import java.io.\*;

class SimpleRead{

 public static void main(String args[]){

  try{

    FileInputStream fin=new FileInputStream("abc.txt");

    int i=0;

    while((i=fin.read())!=-1){

     System.out.println((char)i);

    }

    fin.close();

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

 }

}