**Java Notes**

CPU

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| |

|-----> | CU |---> sends the command signal to

| |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | every connected peripheral

Input | | | Output

[ ] | |-->| ALU |---| |---> [ ]

| | | |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| | |

| | |----| |<--| |

| |---------| IM | |

scanf() |----------------> |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|---------------| printf()

cin ^ | cout

\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_V\_\_\_\_\_\_

| |

| EM |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

**---------- Java Programming -------------**

A computer is a two-state, Multipurpose, programmable, electronic device, that takes input from a user, stores it, processes it, and gives the output in a desired format.

SIMULA

COBOL |-- Sun Microsystem --> Java

Assembly --> Fortran ---> ALGOL60 --> CPL --> BCPL----> B ----> C ---> C++ ==|-- Microsoft corpo.--> .Net

RPG (1960) (1963) (1967) (1970) (1972) (1983-84) |-- .......

BASIC |

Pascal

....

Instructions Prog\_nm

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

|

| Debug

source code \*.c/\*.cpp <----------------------------|

| ^

| |

compile ---------------------------->| compile-time error

| | (syntax Errors)

| |

backup file \*.bak |

| |

Linker --------->|-------------------------------->| Linker errors

| |

object code file \*.obj |

| |

| ^

executable file \*.exe |

| |

Run ------------------===------>| Runtime errors

|

Output

Any hardware or software environment in which a program runs is a platform.

Platform = Processor architecture + OS

In C and CPP, the machine code is generated on the machine on which the code is compiled, due to that the application will run only on machines having the same config. (platform)which is known as platform-dependent. This is the limitation of C and C++.

Now we want the platform-independent code.

\*.java

|

|

Compile

(javac)

|

|

Bytecode

(\*.class)

Compile |

================ |

Interpretation -----------------------------------------

| | | ...

JVM JVM JVM 🡨----- JVM is different for every platform

(Interpreter - java) | | | means it is platform dependent

Windows Linux Mac

| | |

Native Code Native Code Native Code

| | |

Run Run Run

| | |

Output Output Output

Java Editors and IDE's

==========================================================

Editor: Notepad, Editplus, Notepad++ ...

IDE: Best Java IDEs

Eclipse. Platform – Linux/macOS/Solaris/Windows. ...

NetBeans. Platform – Linux/macOS/Solaris/Windows. ...

IntelliJ IDEA. Platform – Linux/macOS/Windows. ...

BlueJ. Platform – Linux/macOS/Windows. ...

(Oracle) JDeveloper. Platform – Linux/macOS/Windows.

Now let's see, how to write a code:

1. Use any editor/ide

2. Install jdk/jre (https://www.oracle.com/in/java/technologies/javase/javase8-archive-downloads.html)

3. Use Notepad as an editor and write code as

class <cls\_nm>

{

public static void main(String []args)

{

------------------;

------------------;

program\_body ;

------------------;

------------------;

}

}

class Demo

{

public static void main(String []args)

{

System.out.print("Welcome to Java Programming");

}

}

4. Save the code in C:\Program Files\Java\jdk-17.0.1\bin As <class\_nm>.java

5. win+r --> cmd --> enter (attend the folder where the source file, compiler, and interpreter are present) as

C:\Users\hp>cd\

C:\>cd "Program Files\Java\jdk-17.0.1\bin"

C:\Program Files\Java\jdk-17.0.1\bin>javac Demo.java (compilation where you get the bytecode (\*.class) )

C:\Program Files\Java\jdk-17.0.1\bin>java Demo (Byte code interpretation)

Welcome to Java Programming

C:\Program Files\Java\jdk-17.0.1\bin>

//------------------------------------------------------------------------------------------------------------------

How to run, the same code when the source file (\*.java) is in a different folder

C:\Users\hp>e:

E:\>cd myjavafiles

E:\myjavafiles>javac First.java

'javac' is not recognized as an internal or external command,

operable program or batch file.

**E:\myjavafiles>set path=C:\Program Files\Java\jdk-17.0.1\bin**

E:\myjavafiles>javac First.java

E:\myjavafiles>java First

Welcome to Java Programming-First

E:\myjavafiles>

Note that the path is applicable till the current session of the command prompt.

to set the path permanently set the path in Environment variable

Setting the environment variable: this pc --> rh+ click ---> properties --> adv. system settings

--> Advanced tab --> environment variable --> user variable path

-if already path is there -> edit --> new-> paste path (C:\Program Files\Java\jdk-17.0.1\bin otherwise user variable path--> new and write

variable name --> path

variable value --> C:\Program Files\Java\jdk-17.0.1\bin) --> ok....

//-------------------------------------------------------------------------------------------------------------

Youtube Link: https://youtu.be/RBxum7M3B94?si=jepmNZAtetZfJKFp

//-------------------------------------------------------------------------------------------------------------

**Details of the welcome program:**

===========================================================================

class WelcomeProg

{

public static void main(String []args)

{

System.out.print("Welcome to Java");

}

}

Line 1: class WelcomeProg:

class: It is a keyword that allows you to create your own data type.

WelcomeProg: this is the name of UDT, it must be a valid identifier. Internally in Java lib, they have chosen, the first letter of the class name in uppercase and all others in lowercase if it is made from one word, if multiple words then the first character of each word is in ucase and all others in lcase.

e.g. First, Demo, FirstProgram, ExampleDemoWelcome

It is recommended, not compulsory

Line 3: public static void main(String []args)

public: it is used to define the visibility of method main(), coz Javac and Java are not members of the class as an outsider they must have access to class members therefore visibility is public.

static: The static members gain the memory space when the class is loaded into memory, no need for object creation. therefore, the method main() decl. as static

void: It is returning type of method main(), it is void coz the java program does not return any value to OS

main(): It is the method name, and as it is main(), it is considered as a starting point of your program

String []args: String is a Built-in class from java.lang package is a language support package, which is imported by default. []args is an array of arguments, which is passed automatically at the time of execution from the command line, in the absence null is collected. Simply it is an array of objects.

Line 5: System.out.print("Welcome to Java");

"Welcome to Java": It is data, to be displayed

print() is a method from the PrintStream class used to display the data on screen.

out is a predefined object of the PrintStream class, declared as static in the System class

The system is a class from java.lang package.

//-----------------------------------------------------------------------------------------------------------------------

class FirstProgram

{

public static void main(String []args)

{

System.out.println("Welcome to Java Programming");

System.out.println(args[0]);

System.out.println(args[1]);

System.out.println(args[2]);

System.out.println("Bye Bye...!!");

}

}

***Note that, the IO in Java is in the form of a string only.***

Now we need to proceed using the path followed in the C and C++

i.e.

constant

character set -----> keyword ------> Instructions ----> program ----> module ---> software

variable

**Java Character set:**

- ASCII(American Standard Code for Information Interchange): Provides the binary string to all symbols present in US English, which are used in different electronic devices.

- ASCII used to code in English, but Java supports different human-understandable languages for coding. means Java has a rich character set as compared to c/c++.

-The Standard Code system names the **Unicode system** used in Java which provides the 16-bit binary string to each symbol for different languages.

**Constants:**

These are the elements in the program having a fixed value.

Java Constants

|

---------------------------

| |

Numeric non-numeric

| |

----------------- -----------------

| | | |

Integer floating-point character String

-6,4,56 -5.3,7.0,78.6755 'a', '$' "A", "Hello"

'H' '4' "a123"

**Keyword:**

These are *reserved words*, whose meaning is already known to the compiler.

abstract continue for new switch

assert\*\*\* default goto\* package synchronized

boolean do if private this

break double implements protected throw

byte else import public throws

case enum\*\*\*\* instanceof return transient

catch extends int short try

char final interface static void

class finally long strictfp\*\* volatile

const\* float native super while

\* not used

\*\* added in 1.2

\*\*\* added in 1.4

\*\*\*\* added in 5.0

(https://docs.oracle.com/javase/tutorial/java/nutsandbolts/\_keywords.html)

**Java Data Types:**

Tool used for the memory allocation.

***- Primitive data types***: These are provided by the language itself. e.g. int, char, byte...

***- Non-Primitive data types***: These are defined by the programmers according to the need. e.g. class, interface

- The default integer value is considered as an int and the default fractional value is considered as double.

- When you declare the variable, java demands for the init. of variables, otherwise it will generate the error message,

"Variable xxx might not have been initialized"

- When the value of a variable having a higher type is assigned to a variable of lower type, then it will generate the error message

e.g. a=c; gives the following error, when a is a byte variable and c is an int variable

"Possible lossy conversion from int to byte"

In such case, where you want to convert the value from higher type to lower type, go for

the type casting; i.e. a=(byte)c;

Note carefully that, lower type to higher type is promoted automatically.

- The long constant is represented using 'l' or 'L' as a prefix and for the float 'f' or 'F' is used.

- Java allows you to decl. the variables anywhere in the program, just decl before using it.

Java Data Types

|

-----------------------------------------

| |

Primitive Data Types Non-Primitive Data Types

| |

--------------------------------- - string

| | - Array

Numeric boolean - Vector

| (1 bit)

-----------------------

| |

Integrals character

| |

------------------------- char(2)

| |

integer floating point

| |

|- byte (1) |- float(4)

|- short (2) |- double(8)

|- int(4)

|- long(8)

Note: bracket values tell about the size of the data type in bytes

class IntData

{

public static void main(String []args)

{

byte b=10; // If the literal it will be considered as a byte value

System.out.println("value of b: "+b);

int x=100;

// b=x; //If downcast then it will generate an error

b=(byte)x;

System.out.println("value of b: "+b);

long t;

t=x;

System.out.println("value of t: "+t);

}

}

//-----------------------------------------------------------------------------------------------------------------------------

**/// Different ways of input in Java**

**// 1. init and display**

class InitID

{

public static void main(String []args)

{

byte b=10;

short s=23;

int i=50;

long no=456L;

System.out.println("\n b="+b+"\t s="+s+"\t i="+i+"\t no="+no);

float ft=4.5F;

double db=56.233;

System.out.println("\n ft="+ft+"\t db="+db);

boolean ans=true;

System.out.println("\n ans is: "+ans);

}

}

//---------------------------------------------------------------------------------------------------

**//2. Data input using commandline**

class CmdlnID

{

public static void main(String []args)

{

int a,b;

a=args[0];

b=args[1];

System.out.println("\n a="+a+"\t b="+b);

}

}

output on compile

E:\jprodyp>javac CmdlnID.java

CmdlnID.java:6: error: incompatible types: String cannot be converted to int

a=args[0];

^

CmdlnID.java:7: error: incompatible types: String cannot be converted to int

b=args[1];

^

2 errors

//------------------------------------------------------------------------------

class CmdlnID

{

public static void main(String []args)

{

int a=0,b=0;

a=Integer.parseInt(args[0]);

b=Integer.parseInt(args[1]);

System.out.println("\n a="+a+"\t b="+b);

System.out.println("\n Sum: "+(a+b));

}

}

E:\jprodyp>javac CmdlnID.java

E:\jprodyp>java CmdlnID

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0 at CmdlnID.main(CmdlnID.java:6)

E:\jprodyp>java CmdlnID 11 22

a=11 b=22

Sum: 33

//------------------------------------------------------------------------------------------------

***// Dealing with the exception***

class CmdlnID

{

public static void main(String []args)

{

int a=0,b=0;

try

{

a=Integer.parseInt(args[0]);

b=Integer.parseInt(args[1]);

}

catch(Exception e)

{

System.out.println("\n Please Pass the arguments through commandline");

}

System.out.println("\n a="+a+"\t b="+b);

System.out.println("\n Sum: "+(a+b));

}

}

output:

E:\jprodyp>javac CmdlnID.java

E:\jprodyp>java CmdlnID

Please Pass the arguments through the commandline

a=0 b=0

Sum: 0

//================================================================================

**//3. Using BufferedReader and InputStreamReader**

import java.io.InputStreamReader;

import java.io.BufferedReader;

class BrIsrID

{

public static void main(String []args)

{

InputStreamReader isr=new InputStreamReader(System.in);

BufferedReader br=new BufferedReader(isr);

//or BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

int x=0;

double y=0.0;

System.out.println("Enter the int value: ");

x=br.readLine();

System.out.println("Enter the double value: ");

y=br.readLine();

System.out.println("\n x="+x+"\t y="+y);

}

}

output on compile

E:\jprodyp>javac BrIsrID.java

BrIsrID.java:15: error: incompatible types: String cannot be converted to int

x=br.readLine();

^

BrIsrID.java:17: error: incompatible types: String cannot be converted to double

y=br.readLine();

^

2 errors

//-------------------------------------------------------------------------------------------------------------------

import java.io.InputStreamReader;

import java.io.BufferedReader;

class BrIsrID

{

public static void main(String []args)

{

InputStreamReader isr=new InputStreamReader(System.in);

BufferedReader br=new BufferedReader(isr);

int x=0;

double y=0.0;

System.out.println("Enter the int value: ");

x=Integer.parseInt(br.readLine());

System.out.println("Enter the double value: ");

y=Double.parseDouble(br.readLine());

System.out.println("\n x="+x+"\t y="+y);

}

}

output on Compile

E:\jprodyp>javac BrIsrID.java

BrIsrID.java:15: error: unreported exception IOException; must be caught or declared to be thrown

x=Integer.parseInt(br.readLine());

^

BrIsrID.java:17: error: unreported exception IOException; must be caught or declared to be thrown

y=Double.parseDouble(br.readLine());

^

2 errors

//------------------------------------------------------------------------------------------------------------------------

There are two different ways to eliminate these Exceptions

a) use try catch

b) use throws clause

import java.io.InputStreamReader;

import java.io.BufferedReader;

class BrIsrID

{

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

{

InputStreamReader isr=new InputStreamReader(System.in);

BufferedReader br=new BufferedReader(isr);

int x=0;

double y=0.0;

System.out.println("Enter the int value: ");

x=Integer.parseInt(br.readLine());

System.out.println("Enter the double value: ");

y=Double.parseDouble(br.readLine());

System.out.println("\n x="+x+"\t y="+y);

}

}

output

E:\jprodyp>javac BrIsrID.java

E:\jprodyp>java BrIsrID

Enter the int value:

12

Enter the double value:

67.34

x=12 y=67.34

OR

import java.io.InputStreamReader;

import java.io.BufferedReader;

class BrIsrID

{

public static void main(String []args)

{

InputStreamReader isr=new InputStreamReader(System.in);

BufferedReader br=new BufferedReader(isr);

int x=0;

double y=0.0;

***try***

{

System.out.println("Enter the int value: ");

x=Integer.parseInt(br.readLine());

System.out.println("Enter the double value: ");

y=Double.parseDouble(br.readLine());

}

***catch***(Exception e){}

System.out.println("\n x="+x+"\t y="+y);

}

}

output:

E:\jprodyp>javac BrIsrID.java

E:\jprodyp>java BrIsrID

Enter the int value:

23

Enter the double value:

45.78

x=23 y=45.78

//================================================================================

**4. Input using Scanner class**

import java.util.Scanner;

class ScannerID

{

public static void main(String []args)

{

Scanner sc=new Scanner(System.in);

int x=0;

double y=0.0;

System.out.println("Enter the int value: ");

x=sc.nextInt();

System.out.println("Enter the double value: ");

y=sc.nextDouble();

System.out.println("\n x="+x+"\t y="+y);

}

}

output:

E:\jprodyp>javac ScannerID.java

E:\jprodyp>java ScannerID

Enter the int value:

11

Enter the double value:

45.67

x=11 y=45.67

//================================================================================

**5. Input using javax.swing.JOptionPane.showInputDialog()**

import javax.swing.JOptionPane;

class InDialogID

{

public static void main(String []args)

{

int x=0;

double y=0.0;

x=Integer.parseInt(JOptionPane.showInputDialog("Enter the int value: "));

y=Double.parseDouble(JOptionPane.showInputDialog("Enter the double value: "));

System.out.println("\n x="+x+"\t y="+y);

}

}

//----------------------------------------------------------------------------------------------------------------------------

**// Operators in Java:**

Operators are used to process the data. There are following operators present in the java.

- Assignment Operators (= and short-hand operators)

- Unary Operators ( - ++ -- (type) )

- Arithmetic Operators ( + - \* / % )

- Relational Operators ( < <= > >= == != )

- Logical Operators (&& || !)

- conditional Operator ( ? : )

- Bitwise Operators (& | ^ >> << >>>)

- special Operators (. and instanceof )

**- Assignment Operators** (= and short-hand operators): will assigns constant value at its rh+, value of variable at its rh+ or answer of expression at its rh+ to variable at left.

e.g. int x=10; int y=x; int z=x+y;

shorthand expression: suppose, x=x+10 can be written as x+=10;

x=x/10 --> x/=10

....

class DemoAssignment

{

public static void main(String []args)

{

int x=10;

int y=x;

int z=x+y;

System.out.println("\n x="+x+"\t y="+y+"\t z="+z);

x+=100;

y\*=2;

System.out.println("\n x="+x+"\t y="+y+"\t z="+z);

}

}

**- Unary Operators** ( - ++ -- (type) )

- will gives opposite value

++ incr by 1

-- decr by 1

inc/dec

|

--------------------------------

| |

pre post

(++x, --x) (x++, x--)

++x <-----> x=x+1 <------> x++

--x <-----> x=x-1 <------> x--

when these operators are used in the expression,

***pre --> expr --> post***

suppose x=5, and y=9

z = ++x + y-- ;

- find the basic expr

- operate all pre operators

- calc. the basic expr. with current values

- operator all post operators

- x becomes 6

- assigned 15 to z

-y becomes 8

import java.util.Scanner;

class DemoUnary

{

public static void main(String []args)

{

int x=0;

int y=0;

int z=0;

Scanner sc= new Scanner(System.in);

System.out.println("\n Enter the values of x and y: ");

x=sc.nextInt();

y=sc.nextInt();

z=-x;

x++;

--y;

System.out.println("\n x="+x+"\t y="+y+"\t z="+z);

z=++x+y--;

System.out.println("\n x="+x+"\t y="+y+"\t z="+z);

}

}

//------------------------------------------------------------------------------------------------------------------------------

(type): It refers to typecasting, which means changing the data type of variable only at the time of calc.

import java.util.Scanner;

class DemoCasting

{

public static void main(String []args)

{

int x=0,y=0;

double z=0;

Scanner sc= new Scanner(System.in);

System.out.println("\n Enter the values of x and y: ");

x=sc.nextInt();

y=sc.nextInt(); // 13, 5

z=x/y;

System.out.println("\n x="+x+"\t y="+y+"\t z="+z);

z=(double)x/y;

System.out.println("\n x="+x+"\t y="+y+"\t z="+z);

z=x/(double)y;

System.out.println("\n x="+x+"\t y="+y+"\t z="+z);

z=(double)x/(double)y;

System.out.println("\n x="+x+"\t y="+y+"\t z="+z);

}

}

output:

E:\jprodyp>javac DemoCasting.java

E:\jprodyp>java DemoCasting

Enter the values of x and y:

13

5

x=13 y=5 z=2.0

x=13 y=5 z=2.6

x=13 y=5 z=2.6

x=13 y=5 z=2.6

//----------------------------------------------------------------------------------------------------------------------------------

**- Arithmetic Operators** ( + - \* / % )

// program to calculate the simple interest.

import java.io.InputStreamReader;

import java.io.BufferedReader;

class DemoArith

{

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

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

int p=0,n=0;

double r=0.0,si=0.0;

System.out.println("Enter the value of p: ");

p=Integer.parseInt(br.readLine());

System.out.println("Enter the value of r: ");

r=Double.parseDouble(br.readLine());

System.out.println("Enter the value of n: ");

n=Integer.parseInt(br.readLine());

si=(p\*r\*n)/100;

System.out.println("Simple Interst is: "+si);

}

}

//Using % and / operator

let's see a simple example, we have to calculate 13/5

2 <------------- (13/5)

\_\_\_\_\_\_\_\_

5 ) 13

- 10

---------

3 <----------- (13%5)

let's see some examples, observe the result, and write the conclusion

13/5=2 13%5=3

27/7=3 27%7=6

67/9=7 67%9=4

123/10=12 123%10=3

459/10=45 459%10=9

3857/10=385 3857%10=7

7/10=0 7%10=7

- Div by 10 eliminates the last digit from the number.

and mod by 10 gives the last digit.

- In N/D, when N<D then div is 0 and rem is N

// Enter any 3-digit number from the keyboard and find the addition of its all digits.

// no=285 then ans = 5+8+2 => 15

import java.util.Scanner;

class DemoDivMod

{

public static void main(String []args)

{

int no=0,rem=0,tot=0;

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter any 3-digit number: ");

no=sc.nextInt(); //285

rem=no%10; //5

tot=tot+rem; // 0+5=5

no=no/10; //28

rem=no%10; //8

tot=tot+rem; // 5+8=13

no=no/10; //2

rem=no%10; //2

tot=tot+rem; // 13+2=15

no=no/10; //0

System.out.println("\n Total is "+tot);

}

}

Unlike C/C++, here in Java, you can operate the % operator on fractional and -ve values,

when you operate the % operator on -ve values the sign of ans is taken as the sign of N form N/D.

//--------------------------------------------------------------------------------------------------------------------------

**// Relational Operators:** (<, <=, >, >= ==, !=):

These operators are used to find the relation between two operands. It will form the condition that is useful in the conditional control statements.

The ans of condition is a boolean value true when it is true and false when false.

suppose x=23 y=5;

x>y ----> true means if we write z=x>y then true is assigned to z.

x!=y ----> true

y<1 ----> false

x%10==0 ----> false

100%y==0 ---> true

class DemoRel

{

public static void main(String []args)

{

int x=23,y=5;

boolean b;

System.out.println("\n x is: "+x+"\t y is: "+y);

b=x>y;

System.out.println("\n (x>y) is: "+b);

b=x!=y;

System.out.println("\n (x!=y) is: "+b);

b=y<1;

System.out.println("\n (y<1) is: "+b);

b=x%10==0;

System.out.println("\n (x%10==0) is: "+b);

b=100%y==0;

System.out.println("\n (100%y==0) is: "+b);

}

}

output

E:\jprodyp>javac DemoRel.java

E:\jprodyp>java DemoRel

x is: 23y is: 5

(x>y) is: true

(x!=y) is: true

(y<1) is: false

(x%10==0) is: false

(100%y==0) is: true

//-----------------------------------------------------------------------------------------------

**/// Logical Operator ( && || !):**

These operators are used to join two or more conditions

when the conditions are joined by

- && --> gives true only when both true otherwise false

- || --> gives false only when both false and otherwise true

- ! --> gives

!(true) ---> false

!(false) ---> true

suppose x=23 y=5;

(x>y)&&(y<100) ----> true

(x>y)&&(y>100) ----> false

(x>y)||(y>100) ----> true

(x<y)||(y>100) ----> false

!(x!=y) ----> false

!(y<1) ----> true

class DemoLogical

{

public static void main(String []args)

{

int x=23,y=5;

boolean b;

System.out.println("\n x is: "+x+"\t y is: "+y);

b=(x>y)&&(y<100);

System.out.println("\n ((x>y)&&(y<100)) is: "+b);

b=(x>y)&&(y>100);

System.out.println("\n ((x>y)&&(y>100)) is: "+b);

b=(x>y)||(y>100);

System.out.println("\n ((x>y)||(y>100)) is: "+b);

b=(x<y)||(y>100);

System.out.println("\n ((x<y)||(y>100)) is: "+b);

b=!(x!=y);

System.out.println("\n (!(x!=y)) is: "+b);

b=!(y<1);

System.out.println("\n (!(y<1)) is: "+b);

}

}

output:

E:\jprodyp>javac DemoLogical.java

E:\jprodyp>java DemoLogical

x is: 23 y is: 5

((x>y)&&(y<100)) is: true

((x>y)&&(y>100)) is: false

((x>y)||(y>100)) is: true

((x<y)||(y>100)) is: false

(!(x!=y)) is: false

(!(y<1)) is: true

///-------------------------------------------------------------------------------------------------------------------------------

**// Conditional operator or ternary operator or if-then-else operator(?:):**

This is the only operator which has decision ability.

syntax:

<condition> ? <options>;

<condition> ? <true\_part> : <false\_part> ;

// WAP to find the max from 2 nos

import java.util.Scanner;

class DemoConditionalOperator

{

public static void main(String []args)

{

Scanner sc=new Scanner(System.in);

int x=0,y=0;

System.out.println("\n Enter any two nos: ");

x=sc.nextInt();

y=sc.nextInt();

int z = (x>y) ? x : y ;

System.out.println("\n Max no: "+z);

}

}

Output:

E:\jprodyp>javac DemoConditionalOperator.java

E:\jprodyp>java DemoConditionalOperator

Enter any two nos:

45

78

Max no: 78

E:\jprodyp>java DemoConditionalOperator

Enter any two nos:

90

23

Max no: 90

///------- Nesting of conditional operators

// WAP to find the max from 3 nos

import java.util.Scanner;

class DemoConditionalOperator1

{

public static void main(String []args)

{

Scanner sc=new Scanner(System.in);

int x=0,y=0,z=0;

System.out.println("\n Enter any three nos: ");

x=sc.nextInt();

y=sc.nextInt();

z=sc.nextInt();

int max = (x>y) ? (x>z?x:z) : (y>z?y:z) ;

System.out.println("\n Max no: "+max);

}

}

output:

E:\jprodyp>javac DemoConditionalOperator1.java

E:\jprodyp>java DemoConditionalOperator1

Enter any three nos:

11

22

33

Max no: 33

//-----------------------------------------------------------------------------------------------------------------------

**/// Bitwise Operator:** [ & | ^ >> << >>> ]

These operators are used in the bit-level operations.

& ==> 1 & 1 -> 1 otherwise 0

| ==> 0 | 0 -> 0 otherwise 1

0^0

^ --> ==> 0 otherwise 1

1^1

suppose x=10 y=12

(0000 1010) (0000 1100)

(x&y) (x|y) (x^y)

1010 1010 1010

&1100 |1100 ^1100

====== ===== ======

1000 1110 0110

(8) (14) (6)

x=10 (0000 1010) y=12 (0000 1100)

z=x<<2 z=y>>2

(0010 1000) (0000 0011)

==>40 ==> 3

class DemoBitwise

{

public static void main(String []args)

{

int x=10,y=12;

System.out.println("\n (x&y) is"+ (x&y) );

System.out.println("\n (x|y) is"+ (x|y) );

System.out.println("\n (x^y) is"+ (x^y) );

System.out.println("\n (x<<2) is"+ (x<<2) );

System.out.println("\n (y>>2) is"+ (y>>2) );

}

}

//---------------------------------------------------------------------------------------------------------------------------------

**// Control Statements in Java:**

Control Statements

|

---------------------------------------------

| |

Conditional Un-Conditional

| |

------------------------------------ -------------------------------------------

| | | | | | |

Decision Loop case break continue lbl.break return

| | | and

|- if() |- for() switch lbl. continue

|- if() else |- while()

|- nesting |- do..while()

|- ladder

**// Decision Conditional Control statement:**

Using if(): used to decide, whether to execute the block of code or not. That block is mentioned in the program as

syntax:

if(<condi>)

{

-------------;

block of code;

-------------;

}

- Block of code will be executed only when the condition is true otherwise it will be skipped

import java.io.InputStreamReader;

import java.io.BufferedReader;

class DemoIf

{

public static void main(String []args)

{

InputStreamReader isr=new InputStreamReader(System.in);

BufferedReader br=new BufferedReader(isr);

int a=0;

try

{

System.out.println("Enter the value of a: ");

a=Integer.parseInt(br.readLine());

}

catch(Exception e){}

if(a%7==0)

{

System.out.println("Entered no is div by 7 ");

}

if(a%7!=0)

{

System.out.println("Entered no is not div by 7 ");

}

}

}

//-----------------------------------------------------------------------------------

***// Using if() else***: It is used when you want to execute any one code block from two different blocks according to condition.

syntax:

if(<condi>)

{

-----------------;

---------------;

----------------;

}

else

{

-----------------;

---------------;

----------------;

}

when <condi> is TRUE --> will execute the if() block only

FALSE -> will execute the else block only

import java.io.InputStreamReader;

import java.io.BufferedReader;

class DemoIfElse

{

public static void main(String []args)

{

InputStreamReader isr=new InputStreamReader(System.in);

BufferedReader br=new BufferedReader(isr);

int a=0;

try

{

System.out.println("Enter the value of a: ");

a=Integer.parseInt(br.readLine());

}

catch(Exception e){}

if(a%7==0)

{

System.out.println("Entered no is div by 7 ");

}

else

{

System.out.println("Entered no is not div by 7 ");

}

}

}

***// Using Nesting of if() else:***

Nesting refers to using one control statement in to same or another control statement

class DemoIfElseNesting

{

public static void main(String []args)

{

int a=Integer.parseInt(args[0]);

int b=Integer.parseInt(args[1]);

int c=Integer.parseInt(args[2]);

if(a>b)

{

if(a>c)

{

System.out.println("\n a is max");

}

else

{

System.out.println("\n c is max");

}

}

else

{

if(b>c)

{

System.out.println("\n b is max");

}

else

{

System.out.println("\n c is max");

}

}

}

}

//---------------------------------------------------------------------------------------------------------------------

***// Using if() else Ladder***

syntax:

if(<>)

{

---------;

---------;

}

else if(<>)

{

---------;

---------;

}

else if(<>)

{

---------;

---------;

}

else if(<>)

{

---------;

---------;

}

[<else>]

{

---------;

---------;

}

// Enter the co-ordinates of point in 2D system, and display the exact location of that point.

y axis

^

II QD | I QD

|

-+ | ++

<-------------|------------> x axis

|(0,0)

|

-- | +-

III QD V IV QD

There are 7 different possibilities.

import java.util.Scanner;

class DemoIfElseLadder

{

public static void main(String []args)

{

int x=0,y=0;

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the x cord: ");

x=sc.nextInt();

System.out.println("\n Enter the y cord: ");

y=sc.nextInt();

if(x>0&&y>0)

{

System.out.println("\n Point present in I st qd");

}

else if(x<0&&y>0)

{

System.out.println("\n point present in II nd qd");

}

else if(x<0&&y<0)

{

System.out.println("\n Point is present in 3 rd qd");

}

else if(x>0&&y<0)

{

System.out.println("\n Point is present in 4 th qd");

}

else if(x!=0&&y==0)

{

System.out.println("\n Point is present on x axis");

}

else if(x==0&&y!=0)

{

System.out.println("\n Point is present on y axis");

}

else

{

System.out.println("\n Point present at org");

}

}

}

------------------------------------------------------------------------------------------------------------------------------------

// Using the Loops in Java: Loops are used to avoid the continue repetition of code in the program.

There are three different loops in C.

1. For() loop 2. While() loop 3. do…While() loop

**1. For() loop:**

syntax:

for( [<init>] ; <condi> ; [<inc/dec/stat/expr>] )

{

--------------;

--------------;

}

import javax.swing.JOptionPane;

class DemoFor

{

public static void main(String []args)

{

int no=Integer.parseInt(JOptionPane.showInputDialog("Enter any number:"));

int t=0,tot=0;

for(t=no;no!=0;no=no/10)

{

tot=tot+(no%10);

}

System.out.println("\n Addition of all digits from "+t+" is "+tot");

}

}

///-------------------------------------------------------------------------------------------------------------------------

**// 2. Using while loop:**

Again, the aim is same i.e. used to avoid the code rep.

syntax:

while(<cond>)

{

----------------;

----------------;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;

}

- It will execute the body of loop, till the condition is true.

- <init> block is absent in while() but you have init. the iterator before starting of loop.

- <inc/dec> block is absent, but you have to add at least one statement which will make the <cond> false after some iterations. otherwise, it will attend the infinite looping

/// WAP to display 1 to 15 nos using while loop

class DemoWhile

{

public static void main(String []args)

{

int i;

i=1;

while(i<=15)

{

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

i++;

}

}

}

//-------------------------------------------------------------------------------------------

/// WAP to display list of odd nos from 1 to 50 using while loop.

class DemoWhile

{

public static void main(String []args)

{

int i;

i=1;

while(i<=50)

{

if(i%2!=0)

{

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

}

i++;

}

}

}

//----------------------------------------------------------------------------------------------------------------------------------

// WAP to find the entered number is prime

import java.util.\*;

class DemoWhile

{

public static void main(String []args)

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter any no: ");

int no=sc.nextInt();

int d=2;

int flg=0;

while(d<=(no/2))

{

if(no%d==0)

{

flg=1;

break;

}

d++;

}

if(flg==0)

{

System.out.println("\n Entered no is prime ");

}

else

{

System.out.println("\n Entered no is not prime ");

}

}

}

//----------------------------------------------------------------------------------------------------------------------------------

***//3. using do while()***

syntax:

do

{

----------;

----------;

}while(<cond>);

// Display list of prime nos from given range

import java.util.\*;

class DemoWhile

{

public static void main(String []args)

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the range starts from: ");

int n1=sc.nextInt();

System.out.println("\n Enter the range ends to: ");

int n2=sc.nextInt();

int d=2,flg=0;

System.out.println("\n List of prime nos: ");

for(no=n1;no<=n2;no++)

{

d=2;

flg=0;

while(d<=(no/2))

{

if(no%d==0)

{

flg=1;

break;

}

d++;

}

if(flg==0)

System.out.println(" "+no);

}

}

//------------------------------------------------------------------------------------------------------------------------------

// WAP to display \*

class StarPattern

{

public static void main(String []args)

{

System.out.print("\*")

}

}

//--------------------------------------------------------------------------------------------------------------------------------

WAP to print

\*\*\*\*\*

class StarPattern

{

public static void main(String []args)

{

int j=0;

for(j=0;j<5;j++)

{

System.out.print("\*");

}

}

}

//----------------------------------------------------------------------------------------------------------------------------------

WAP to print

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

class StarPattern

{

public static void main(String []args)

{

int i, j=0;

for(i=0;i<5;i++)

{

for(j=0;j<5;j++)

{

System.out.print("\*");

}

System.out.print("\n");

}

}

}

//----------------------------------------------------------------------------------------------------------------------------------

WAP to print

j 01234

\*\*\*\*\* i=0

\* \* i=1

\* \* i=2

\* \* i=3

\*\*\*\*\* i=4

class StarPattern

{

public static void main(String []args)

{

int i, j=0;

for(i=0;i<5;i++) // No of lines

{

for(j=0;j<5;j++) // no of cols

{

if(i==0 || i==4)

{

System.out.print("\*");

}

else if(j==0 || j==4)

{

System.out.print("\*");

}

else

{

System.out.print(" ");

}

}

System.out.print("\n");

}

}

}

//-------------------------------------------------------------------------------------------------------------------------------

WAP to print

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

class StarPattern

{

public static void main(String []args)

{

int i, j=0;

for(i=0;i<5;i++)

{

for(j=0;j<5;j++)

{

if(j<=i)

{

System.out.print("\*");

}

}

System.out.print("\n");

}

}

}

//---------------------------------------------------------------------------------------------------------------------------------

WAP to print

\*\*\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

class StarPattern

{

public static void main(String []args)

{

int i, j=0;

for(i=0;i<5;i++)

{

for(j=0;j<5;j++)

{

if(j<(5-i))

{

System.out.print("\*");

}

}

System.out.print("\n");

}

}

}

//--------------------------------------------------------------------------------------------------------------------------------

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*

class StarPattern

{

public static void main(String []args)

{

int i, j=0;

for(i=0;i<6;i++)

{

for(j=0;j<(6+i);j++)

{

if(j<(5-i))

{

System.out.print(" ");

}

else

{

System.out.print("\*");

}

}

System.out.print("\n");

}

}

}

//------------------------------------------------------------------------------------------------------------------

1

222

33333

4444444

555555555

66666666666

class StarPattern

{

public static void main(String []args)

{

int i, j=0;

for(i=0;i<6;i++)

{

for(j=0;j<(6+i);j++)

{

if(j<(5-i))

{

System.out.print(" ");

}

else

{

System.out.print(i+1);

}

}

System.out.print("\n");

}

}

}

//--------------------------------------------------------------------------------------------------------------------------

A

ABC

ABCDE

ABCDEFG

ABCDEFGHI

ABCDEFGHIJK

class StarPattern

{

public static void main(String []args)

{

int i, j=0;

char ch;

for(i=0;i<6;i++)

{

ch='A';

for(j=0;j<(6+i);j++)

{

if(j<(5-i))

{

System.out.print(" ");

}

else

{

System.out.print(ch);

ch++;

}

}

System.out.print("\n");

}

}

}

//---------------------------------------------------------------------------------------------------------------------------------

**switch():** It is used in the menu driven programming.

syntax:

switch(<opt>)

{

case <CC>:

---------;

---------;

break;

case <CC>:

---------;

---------;

break;

case <CC>:

---------;

---------;

break;

case <CC>:

---------;

---------;

break;

[<default>]:

-----------;

----------;

}

------------------;

The switch case is used when there are more possibilities, and from which we have to

choose any one according to users’ choice.

import java.util.Scanner;

class DemoSwitch

{

public static void main(String []args)

{

Scanner sc=new Scanner(System.in);

int opt=0;

double a=0.0, b=0.0, ans=0.0;

System.out.println("\n\*\*\* Menu \*\*\*\*");

System.out.println("1.add \n 2.sub \n 3.multi \n 4.div ");

System.out.println("select your option: ");

opt=sc.nextInt();

System.out.println("Enter any two nos: ");

a=sc.nextDouble();

b=sc.nextDouble();

switch(opt)

{

case 1:

ans=a+b;

break;

case 2:

ans=a-b;

break;

case 3:

ans=a\*b;

break;

case 4:

ans=a/b;

break;

default:

System.out.println("\n Incorrect Option");

}

System.out.println("\n Ans is: "+ans);

}

}

//-------------------------------------------------------------------------------------------------------------------------------

// Using the character as an option (both lcase and ucase for a single case)

import java.util.Scanner;

class DemoSwitch

{

public static void main(String []args)

{

Scanner sc=new Scanner(System.in);

char opt=0;

double a=0.0, b=0.0, ans=0.0;

System.out.println("\n\*\*\* Menu \*\*\*\*");

System.out.println(" a.add \n b.sub \n c.multi \n d.div ");

System.out.println("select your option: ");

opt=sc.nextLine().charAt(0);

System.out.println("Enter any two nos: ");

a=sc.nextDouble();

b=sc.nextDouble();

switch(opt)

{

case 'A':

case 'a':

ans=a+b;

break;

case 'B':

case 'b':

ans=a-b;

break;

case 'C':

case 'c':

ans=a\*b;

break;

case 'D':

case 'd':

ans=a/b;

break;

default:

System.out.println("\n Incorrect Option");

}

System.out.println("\n Ans is: "+ans);

}

}

//-----------------------------------------------------------------------------------------------------------------------------------

import java.util.Scanner;

class DemoSwitch

{

public static void main(String []args)

{

Scanner sc=new Scanner(System.in);

int i=0, opt=0;

double a=0.0, b=0.0, ans=0.0;

while(i<3)

{

i++;

System.out.println("\n\*\*\* Menu \*\*\*\*");

System.out.println("1.add \n 2.sub \n 3.multi \n 4.div ");

System.out.println("select your option: ");

opt=sc.nextInt();

System.out.println("Enter any two nos: ");

a=sc.nextDouble();

b=sc.nextDouble();

switch(opt)

{

case 1:

ans=a+b;

break;

case 2:

ans=a-b;

break;

case 3:

ans=a\*b;

break;

case 4:

ans=a/b;

break;

default:

System.out.println("\n Incorrect Option");

}

System.out.println("\n Ans is: "+ans);

}

}

}

//---------------------------------------------------------------------------------------------------------------------------------

***// Using the switch within infinite loop (Termination using break)***

import java.util.Scanner;

class DemoSwitch

{

public static void main(String []args)

{

Scanner sc=new Scanner(System.in);

int opt=0;

double a=0.0, b=0.0, ans=0.0;

while(true)

{

System.out.println("\n\*\*\* Menu \*\*\*\*");

System.out.println("1.add \n 2.sub \n 3.multi \n 4.div \n 5.stop");

System.out.println("select your option: ");

opt=sc.nextInt();

if(opt==5)

break;

System.out.println("Enter any two nos: ");

a=sc.nextDouble();

b=sc.nextDouble();

switch(opt)

{

case 1:

ans=a+b;

break;

case 2:

ans=a-b;

break;

case 3:

ans=a\*b;

break;

case 4:

ans=a/b;

break;

default:

System.out.println("\n Incorrect Option");

}

System.out.println("\n Ans is: "+ans);

}

}

}

//----------------------------------------------------------------------------------------------------------------------------------

// Using the switch within infinite loop (Termination using System.exit(0) )

import java.util.Scanner;

class DemoSwitch

{

public static void main(String []args)

{

Scanner sc=new Scanner(System.in);

int opt=0;

double a=0.0, b=0.0, ans=0.0;

while(true)

{

System.out.println("\n\*\*\* Menu \*\*\*\*");

System.out.println("1.add \n 2.sub \n 3.multi \n 4.div \n 5.stop");

System.out.println("select your option: ");

opt=sc.nextInt();

System.out.println("Enter any two nos: ");

a=sc.nextDouble();

b=sc.nextDouble();

switch(opt)

{

case 1:

ans=a+b;

break;

case 2:

ans=a-b;

break;

case 3:

ans=a\*b;

break;

case 4:

ans=a/b;

break;

case 5:

System.exit(0);

default:

System.out.println("\n Incorrect Option");

}

System.out.println("\n Ans is: "+ans);

}

}

}

//--------------------------------------------------------------------------------------------------------------------------------

**Unconditional control statements**: The control statement does not need any condition.

NOTE: In Java goto is not present.

- using the continue: It will keep the enclosing loop in the running condition without

considering the remaining body of loop.

class DemoContinue

{

public static void main(String []args)

{

int i,j;

for(i=0;i<10;i++)

{

System.out.print("-");

for(j=0;j<10;j++)

{

if(j>i)

{

continue;

}

System.out.print("\*");

}

System.out.println();

}

}

}

//-----------------------------------------------------------------------------------------------------------------------------------

// **Using Labelled Continue**: It will keep loop in running condition, not only

enclosing, but outer loops using label(tag)

class DemoLabelledContinue

{

public static void main(String []args)

{

int i,j;

outer:for(i=0;i<10;i++)

{

System.out.print("-");

for(j=0;j<10;j++)

{

if(i>5)

{

continue outer;

}

if(j>i)

{

continue;

}

System.out.print("\*");

}

System.out.println();

}

}

}

//----------------------------------------------------------------------------------------------------------------

**// Using the break ans labelled break**

class DemoBreak

{

public static void main(String []args)

{

int i,j;

for(i=0;i<10;i++)

{

System.out.print("-");

for(j=0;j<10;j++)

{

if(j>i)

{

break;

}

System.out.print(""+j);

}

System.out.println();

}

}

}

//-----------------------------------------------------------------------------------------------------------------------------------

class DemoLabelledBreak

{

public static void main(String []args)

{

int i,j;

outer:for(i=0;i<10;i++)

{

System.out.print("-");

for(j=0;j<10;j++)

{

if(i>5)

{

break outer;

}

if(j>i)

{

break;

}

System.out.print(""+j);

}

System.out.println();

}

}

}

//--------------------------------------------------------------------------------------------------------------------------------

**// Using return**: pass back the value from called method to calling function method.

//--------------------------------------------------------------------------------------------------------------------------------

**/// Array in Java Programming**

// Def: It is collection of elements having same data type which are conti. arranged in the memory.

// Decl. of array in java:

we have seen the array decl. in c/c++ as,

<data\_type> <ar\_nm>[<index>];

int x[5]; --> // in C/C++ it will allocate the memory

x

[] [] [] [] []

but in java, if we decl array as,

int x[] or int []x;

then it will create the reference variable only.

x

[ ]

this location same as pointer variable, it able to refer towards the

location where data is or will be stored.

Note carefully that, array gains the memory space dynamically and for that we have to use the new keyword as shown below.

syntax:

<data\_type> []<ar\_nm> = new <data\_type>[<index>]; OR

<data\_type> <ar\_nm>[] = new <data\_type>[<index>];

e.g. int []x=new int[6];

x

[] -------> [][][][][][]

class DemoArrayInit

{

public static void main(String []args)

{

int []x={11,67,89,45,2};

System.out.println("\n Array elements are: ");

for(i=0;i<5;i++)

{

System.out.print(" "+x[i]);

}

}

}

// Note that there is one property named length in array, which will gives the number of elements in the array. so, it can be used as

class DemoArrayInit

{

public static void main(String []args)

{

int []x={11,67,89,45,2};

System.out.println("\n Array elements are: ");

for(int i=0;i<x.length;i++)

{

System.out.print(" "+x[i]);

}

}

}

//----------------------------------------------------------------------------------------------------------------------------------

/// Array input and display - using commandline

x

[ ] ------> [][][][][][]

class DemoArrayInCmdln

{

public static void main(String []args)

{

// decl.

int []x=new int[6];

System.out.println("\n Enter any 5 nos: ");

for(int i=0;i<x.length;i++)

{

x[i]=Integer.parseInt(args[i]);

}

// display

System.out.println("\n Array elements are: ");

for(int i=0;i<x.length;i++)

{

System.out.print(" "+x[i]);

}

}

}

//-------------------------------------------------------------------------------------------------------------------------------

/// Array input and display - BufferedReader and InputStreamReader

import java.io.BufferedReader;

import java.io.InputStreamReader;

class DemoArrayInBr

{

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

{

// decl.

int []x=new int[6];

BufferedReader br=new BufferedReader (new InputStreamReader(System.in));

System.out.println("\n Enter nos: ");

for(int i=0;i<x.length;i++)

{

x[i]=Integer.parseInt(br.readLine());

}

// display

System.out.println("\n Array elements are: ");

for(int i=0;i<x.length;i++)

{

System.out.print(" "+x[i]);

}

}

}

//------------------------------------------------------------------------------------------------------------------------------

/// Array input and display - Scanner

import java.util.Scanner;

class DemoArrayInsc

{

public static void main(String []args)

{

// decl.

int []x=new int[6];

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter nos: ");

for(int i=0;i<x.length;i++)

{

x[i]=sc.nextInt();

}

// display

System.out.println("\n Array elements are: ");

for(int i=0;i<x.length;i++)

{

System.out.print(" "+x[i]);

}

}

}

//-------------------------------------------------------------------------------------------------------------------------------------

/// input Array and display all prime nos from it

import java.util.Scanner;

class DemoArrayInsc

{

public static void main(String []args)

{

// decl.

int d=0,flg=0;

int []x=new int[6];

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter nos: ");

for(int i=0;i<x.length;i++)

{

x[i]=sc.nextInt();

}

// process - display all primes

System.out.println("\n Prime elements are: ");

for(int i=0;i<x.length;i++)

{

d=2;

flg=0;

while(d<=(x[i]/2) )

{

if(x[i]%d==0)

{

flg=1;

break

}

d++;

}

if(flg==0)

System.out.print(" "+x[i]);

}

// display

System.out.println("\n Array elements are: ");

for(int i=0;i<x.length;i++)

{

System.out.print(" "+x[i]);

}

}

}

//---------------------------------------------------------------------------------------------------------------------------------

**// Using the 2D array:**

Decl syntax:

<data\_type> <ar\_nm>[][]; // It will create the reference variable only

Inti. of array:

<data\_type> <ar\_nm>[][]= { {},{},{}};

[][][]

[][][]

[][][]

class Demo2DArray

{

public static void main(String []args)

{

int a[][]={{11,22,33},{44,55,66},{77,88,99}};

int i,j;

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

System.out.print(" "+a[i][j]);

}

System.out.println();

}

}

}

OR

class Demo2DArray

{

public static void main(String []args)

{

int a[][]={{11,22,33},{44,55,66},{77,88,99}};

int i,j;

for(i=0;i<a.length;i++)

{

for(j=0;j<a[i].length;j++)

{

System.out.print(" "+a[i][j]);

}

System.out.println();

}

}

}

//---------------------------------------------------------------------------------------------------------------------

**decl. of array:**

<data\_type> <ar\_nm>[][]=new <data\_type>[<rows>][<cols>];

import java.util.Scanner;

class Demo2DArrayIO

{

public static void main(String []args)

{

Scanner sc = new Scanner(System.in);

int a[][]=new int [3][4];

int i,j;

System.out.println(" Enter the array of 3x4: ");

for(i=0;i<3;i++)

{

for(j=0;j<4;j++)

{

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

}

}

System.out.println("\n the array of 3x4: ");

for(i=0;i<a.length;i++)

{

for(j=0;j<a[i].length;j++)

{

System.out.print(" "+a[i][j]);

}

System.out.println();

}

}

}

//-----------------------------------------------------------------------------------------------------------------------------------

**// Variable size array (Jagged Array)**

This the array with number of rows mentioned in the decl. but the number of cols

are decided at runtime.

The data structure of array looks like as...

x

[] ---------------> [] -------------> [][][][][]

[] -------------> [][][][]

[] -------------> [][][][][][][][]

[] -------------> [][][]

[] -------------> [][][][][][]

such array can be decl. as...

int x[][]=new int[5][];

x[0]=new int[5];

x[1]=new int[4];

x[2]=new int[8];

x[3]=new int[3];

x[4]=new int[6];

// WAP using array to input and display the marks of N students in m subjects.

import java.util.\*;

class JaggedDemo

{

public static void main(String []args)

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the student count: ");

int nos=sc.nextInt();

int x[][]=new int[nos][];

int j=0,scnt=0;

System.out.println("\n Enter the Marks of "+nos+" students: ");

for(int i=0;i<nos;i++)

{

System.out.println("\n Enter the subject count: ");

scnt=sc.nextInt();

x[i]=new int[scnt];

System.out.println("\n Enter the marks in "+scnt+" subjects: ");

for(j=0;j<scnt;j++)

{

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

}

}

System.out.println("\n Marks of "+nos+" students ");

for(int i=0;i<x.length;i++)

{

System.out.print("\n Student "+(i+1)+": ");

for(j=0;j<x[i].length;j++)

{

System.out.print(" "+x[i][j]);

}

}

}

}

//-----------------------------------------------------------------------------------------------------------------------------

**OOPS in Java:**

In the oop, we must know the following things.

- class

- Object

- data hiding

- encapsulation

- abstraction

- Instance variables and methods

- constructors and destructors

- static variable and static method

- method overloading and method overriding

- operator overloading

- Inheritance

- polymorphism

- virtual and abstract

All these things are mentioned with ref to C++ background.

// Let’s consider a simple program in C++

void abc(int a1)

{

........;

}

int pqr(int a1, int a2)

{

..........;

return --;

}

int main()

{

int x, y, z;

.......

abc(x);

.......

.......

pqr(y, z);

.......

return 0;

}

// You can provide the security to data using class as shown below.

ob

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|private: |

| x |

| [ ] |

| |

| |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_ |

|public: |

| |

| void in() {...} |

| void out() {...} |

| |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_|

class Number

{

private:

int x;

public:

void in()

{

}

void out()

{

}

};

int main()

{

Number ob;

ob.x=10; // here we can’t use x because it is a private in Number class

ob.in();

return 0;

}

**Using class in Java:** It allows you to create your data type. In the Java Standard Library (JSL) all

the methods are provided as a member method of a class, and in many cases there

are different classes that are used to some existing types in their object form.

syntax:

class <cls\_nm>

{

<instance\_variables>;

<Member\_methods>;

.

.

}

\* Note: There are 4 different visibility modifiers for java classes.

- private: Not accessible directly using (dot .) operator

- public: can be accessed using the (dot .) operator directly

- protected:...

- default: can be accessed using the (dot .) operator directly

\* By default, the class members are **default** in nature.

\* Here in Java, you have to mention the visibility of every class member using public, private, or protected. No keyword for default.

\* There is one unwritten rule. CHOOSE THE FIRST CHARACTER OF THE CLASS NAME IN UPPERCASE AND ALL OTHER IN LOWER IF IT IS MADE FROM SINGLE WORD. IF IT IS MADE FROM MORE WORDS THE FIRST CHARACTER OF EACH WORD IN UCASE AND ALL OTHERS IN LCASE, WITH NO SPACE IN BETWEEN.

//--------------------------------------------------------------------------------------

- Creating the Object in Java: In simple words the object is a variable of class type. Or it is an instance of class.

class Demo

{

...

}

If I used the statement like

Demo d;

d [ ] (reference variable)

It is called a reference variable (It is the same as a pointer variable), which means it can hold

the reference(address) where the data is stored.

This means we have to make some additional efforts to allocate the memory for objects and for that we need to use the new keyword as ...

<class\_nm> <Obj\_nm>=new <cls\_nm>();

i.e.

Demo d=new Demo();

d

[ ]-----------> [ ]

[ ] SPACE ALLOCATED TO

[ ] STORE THE DATA

[ ]

Note that the java object is dynamic in nature.

class Number

{

private int x;

private double y;

public void in()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the val of x: ");

x=sc.nextInt();

System.out.println("Enter the val of y: ");

y=sc.nextDouble();

}

public void out()

{

System.out.println("val of x: "+x);

System.out.println("val of y: "+y);

}

}

// ob1

// [] -----> [ ]

class MainNumber

{

public static void main(String []args)

{

Number ob1; // Only reference variable is created in java

ob1=new Number(); // as java object gains the memory space dynamically, new is used

ob1.in();

ob1.out();

new Number().in(); // Anonymous Object i.e. we can call method by using this way

}

}

//--------------------------------------------------------------------------------------------

// Defining main() method in a same class

import java.util.Scanner;

class Number

{

private int x;

private double y;

public void in()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the val of x: ");

x=sc.nextInt();

System.out.println("Enter the val of y: ");

y=sc.nextDouble();

}

public void out()

{

System.out.println("val of x: "+x);

System.out.println("val of y: "+y);

}

public static void main(String []args)

{

Number ob=new Number();

ob.in();

ob.out();

}

}

//-------------------------------------------------------------------------------------------------------

/// A class containing methods with arguments

class Student

{

private String name;

private int id;

private double per;

public void setData(String a1, int a2, double a3)

{

name=a1;

id=a2;

per=a3;

}

public void showData()

{

System.out.println("Student Information: Name:"+name+"\t ID: "+id+"\t Percentage: "+per);

}

}

class DemoStudent

{

public static void main(String []args)

{

String nm="Amol";

int i=45;

double p=90.56;

Student s1=new Student();

s1.setData(nm, i, p);

s1.showData();

}

}

//-------------------------------------------------------------------------

// creating another object

import java.util.Scanner;

class Student

{

private String name;

private int id;

private double per;

public void setData(String a1, int a2, double a3)

{

name=a1;

id=a2;

per=a3;

}

public void showData()

{

System.out.println("Student Information: Name:"+name+"\t ID: "+id+"\t Percentage: "+per);

}

}

class DemoStudent

{

public static void main(String []args)

{

String nm="Amol";

int i=45;

double p=90.56;

Student s1=new Student();

s1.setData(nm, i, p);

s1.showData();

//--------------------------------------------

String b1;

int b2;

double b3;

Scanner sc=new Scanner(System.in);

System.out.println("Enter the student’s name: ");

b1=sc.nextLine();

System.out.println("Enter the student id: ");

b2=sc.nextInt();

System.out.println("Enter the Percentage: ");

b3=sc.nextDouble();

Student s2=new Student();

s2.setData(b1,b2,b3);

s2.showData();

}

}

//-----------------------------------------------------------------------------------------------

**/// Method overloading**

import java.util.Scanner;

class Student

{

private String name;

private int id;

private double per;

public void setData()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the student’s name: ");

name=sc.nextLine();

System.out.println("Enter the student id: ");

id=sc.nextInt();

System.out.println("Enter the Percentage: ");

per=sc.nextDouble();

}

public void setData(String a1, int a2, double a3)

{

name=a1;

id=a2;

per=a3;

}

public void showData()

{

System.out.println("Student Information: Name:"+name+"\t ID: "+id+"\t Percentage: "+per);

}

}

class DemoStudent

{

public static void main(String []args)

{

String nm="Amol";

int i=45;

double p=90.56;

Student s1=new Student();

s1.setData(nm, i, p);

s1.showData();

//--------------------------------------------

String b1;

int b2;

double b3;

Scanner sc=new Scanner(System.in);

System.out.println("Enter the student’s name: ");

b1=sc.nextLine();

System.out.println("Enter the student id: ");

b2=sc.nextInt();

System.out.println("Enter the Percentage: ");

b3=sc.nextDouble();

Student s2=new Student();

s2.setData(b1,b2,b3);

s2.showData();

//-----------------------------------------------------------------

Student s3=new Student();

s3.setData();

s3.showData();

}

}

//-----------------------------------------------------------------------------------------------------

/// Defining the Constructors in class

**- default constructor**

import java.util.Scanner;

class Student

{

private String name;

private int id;

private double per;

public Student() // Default Constructor no return type and same name as class name

{

name="NA";

id=0;

per=0.0;

}

public void setData()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the student’s name: ");

name=sc.nextLine();

System.out.println("Enter the student id: ");

id=sc.nextInt();

System.out.println("Enter the Percentage: ");

per=sc.nextDouble();

}

public void setData(String a1, int a2, double a3)

{

name=a1;

id=a2;

per=a3;

}

public void showData()

{

System.out.println("Student Information: Name:"+name+"\t ID: "+id+"\t Percentage: "+per);

}

}

class DemoStudent

{

public static void main(String []args)

{

Student s1=new Student();

s1.showData();

}

}

//-------------------------------------------------------------------------------------------------------

**// Parameterized and Copy Constructor**

import java.util.Scanner;

class Student

{

private String name;

private int id;

private double per;

public Student() // Default Constructor

{

name="NA";

id=0;

per=0.0;

}

public Student(String a1, int a2, double a3) // Parame. Constructor

{

name=a1;

id=a2;

per=a3;

}

public Student(Student a) // copy Constructor

{

name=a.name;

id=a.id;

per=a.per;

}

public void setData()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the student name: ");

name=sc.nextLine();

System.out.println("Enter the student id: ");

id=sc.nextInt();

System.out.println("Enter the Percentage: ");

per=sc.nextDouble();

}

public void setData(String a1, int a2, double a3)

{

name=a1;

id=a2;

per=a3;

}

public void showData()

{

System.out.println("Student Information: Name:"+name+"\t ID: "+id+"\t Percentage: "+per);

}

}

class DemoStudentConstructor

{

public static void main(String []args)

{

Student s1=new Student(); // default constructor will get called

s1.showData();

Student s2=new Student("Kiran",12,98.34); //para constructor will get called

s2.showData();

Student s3=new Student(s2); // copy constructor get called

s3.showData();

}

}

//-----------------------------------------------------------------------------------------------------------

**/// Array Of Objects**

s 0 1 2

[] -------------> [ ] [ ] [ ]

import java.util.Scanner;

class Student

{

private String name;

private int id;

private double per;

public Student() // Default Constructor

{

name="NA";

id=0;

per=0.0;

}

public Student(String a1, int a2, double a3) // Parame. Constructor

{

name=a1;

id=a2;

per=a3;

}

public Student(Student a) // copy Constructor

{

name=a.name;

id=a.id;

per=a.per;

}

public void setData()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the student’s name: ");

name=sc.nextLine();

System.out.println("Enter the student id: ");

id=sc.nextInt();

System.out.println("Enter the Percentage: ");

per=sc.nextDouble();

}

public void setData(String a1, int a2, double a3)

{

name=a1;

id=a2;

per=a3;

}

public void showData()

{

System.out.println("Student Information: Name:"+name+"\t ID: "+id+"\t Percentage: "+per);

}

}

class DemoStudentArray s

{ [] -----> [ ][ ][ ]

public static void main(String []args) | | |

{ V V V

Student []s=new Student[3]; //array of references {} {} {}

System.out.println("Enter the information of 3 student: ");

for(int i=0;i<3;i++)

{

s[i]=new Student(); //object created

s[i].setData();

}

System.out.println(" information of 3 student: ");

for(int i=0;i<3;i++)

{

s[i].showData();

}

}

}

//----------------------------------------------------------------------------------------------

**// static member in a class**

Memory allocated for all objects of a single class

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|Common |

| void setData() {} |

| void setData(...) {} |

| void showData() {} |

| double getfees() {} |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

|individual |

| s1 s2 ...... |

| [ ] name [ ] name |

| [ ] id [ ] id |

| [ ] fees [ ] fees |

| [ ] per [ ] per |

| |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

import java.util.Scanner;

class Student

{

private String name;

private int id;

private double fees;

private double per;

public Student() // Default Constructor

{

name="NA";

id=0;

per=0.0;

}

public void setData()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the student’s name: ");

name=sc.nextLine();

System.out.println("Enter the student id: ");

id=sc.nextInt();

System.out.println("Enter the fees paid: ");

fees=sc.nextDouble();

System.out.println("Enter the Percentage: ");

per=sc.nextDouble();

}

public void setData(String a1, int a2, double a3)

{

name=a1;

id=a2;

per=a3;

}

public double getFees()

{ return fees; }

public void showData()

{

System.out.println("Student Information: Name:"+name+"\t ID: "+id+"\t Fees Paid: "+fees+"\t Percentage: "+per);

}

}

class DemoStudentStatic1

{

public static void main(String []args)

{

Student []s=new Student[3];

System.out.println("Enter the information of 3 student: ");

for(int i=0;i<3;i++)

{

s[i]=new Student();

s[i].setData();

}

System.out.println(" information of 3 student: ");

for(int i=0;i<3;i++)

{

s[i].showData();

}

double Total\_fees\_Collected=0.0;

for(int i=0;i<3;i++)

{

Total\_fees\_Collected+=s[i].getFees();

}

System.out.println("\n Amount Received: "+Total\_fees\_Collected);

}

}

//-------------------------------------------------------------------------------------------------------------

// static data member

Memory allocated for all objects of a single class

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|Common |

| void setData() {} |

| void setData(...) {} tot\_amount |

| void showData() {} [ 0.0 ] |

| double getfees() {} |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

|individual |

| s1 s2 ...... |

| [ ] name [ ] name |

| [ ] id [ ] id |

| [ ] fees [ ] fees |

| [ ] per [ ] per |

| |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

import java.util.Scanner;

class Student

{

private String name;

private int id;

private double fees;

private double per;

static private double tot\_amount;

public Student() // Default Constructor

{

name="NA";

id=0;

per=0.0;

}

public void setData()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the student name: ");

name=sc.nextLine();

System.out.println("Enter the student id: ");

id=sc.nextInt();

System.out.println("Enter the fees paid: ");

fees=sc.nextDouble();

tot\_amount=tot\_amount+fees;

System.out.println("Enter the Percentage: ");

per=sc.nextDouble();

}

public void showData()

{

System.out.println("Student Information: Name:"+name+"\t ID: "+id+"\t Fees Paid: "+fees+"\t Percentage: "+per);

}

public void showTotalAmount()

{

System.out.println("Total Fees Collected: "+tot\_amount);

}

}

class DemoStudentStatic1

{

public static void main(String []args)

{

Student []s=new Student[3];

System.out.println("Enter the information of 3 student: ");

for(int i=0;i<3;i++)

{

s[i]=new Student();

s[i].setData();

}

System.out.println(" information of 3 student: ");

for(int i=0;i<3;i++)

{

s[i].showData();

}

s[0].showTotalAmount();

}

}

//---------------------------------------------------------------------------------------------------------

**// Static method and static block**

import java.util.Scanner;

class Student

{

private String name;

private int id;

private double fees;

private double per;

static private double tot\_amount;

static

{

tot\_amount=1000;

}

public Student() // Default Constructor

{

name="NA";

id=0;

per=0.0;

}

public void setData()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the student’s name: ");

name=sc.nextLine();

System.out.println("Enter the student id: ");

id=sc.nextInt();

System.out.println("Enter the fees paid: ");

fees=sc.nextDouble();

tot\_amount=tot\_amount+fees;

System.out.println("Enter the Percentage: ");

per=sc.nextDouble();

}

public void showData()

{

System.out.println("Student Information: Name:"+name+"\t ID: "+id+"\t Fees Paid: "+fees+"\t Percentage: "+per);

}

public static void showTotalAmount()

{

System.out.println("Total Fees Collected: "+tot\_amount);

}

}

class DemoStudentStatic3

{

public static void main(String []args)

{

Student.showTotalAmount();

// (as showTotalAmount is the static method we call it directly by class name without creating an object)

Student []s=new Student[3];

System.out.println("Enter the information of 3 student: ");

for(int i=0;i<3;i++)

{

s[i]=new Student();

s[i].setData();

}

System.out.println(" information of 3 student: ");

for(int i=0;i<3;i++)

{

s[i].showData();

}

Student.showTotalAmount();

}

}

//----------------------------------------------------------------------------------------------

**// Garbage Collector**: Java doesn't have destructors, then the dereferenced memory locations are released automatically by a Garbage Collector.

[][][][][][][][][][]

[][][][s][][][][][][] ob1 dereferenced

[][][][][][][][][][] [] -----X---> [][][][]

[][][][][][][][][][] ob1=ob2

[][][][][][][s][][][] ob2

[][][][][][][][][][] [] --------> [][][][]

[][][][][][][][][][] ^

[][][][][][][][][][] |

[][][][][][][][][][] | ob1

[ ]

void in() protected void finalize()

{ {

int a;

...... }

}

System.gc() //calling garbage collector

//------------------------------------------------------------------------------------------------------------

**forming the relationship**

**Inheritance**   **Association**

IS-A HAS-A

person Car

| |

--------------------- ---------------------

| | | | |

Trainer Student Driver Engine Music Player

Composition Agrégation

CPP Code just for example

class A class A

{ {

}; };

Class B class B

{ {

}; };

class Z:public A, public B class Z

{ {

}; A ob1;

B ob2;

};

// Inheritance in Java: It is the art of defining a new class using the pre-defined classes.

consider the example, in a normal case,

class employee class student class Teacher class Driver

{ { { {

char name[50]; char name[50]; char name[50]; char name[50];

int age; int age; int age; int age;

-----------; -----------; -----------; -----------;

-----------; -----------; -----------; -----------;

-----------; -----------; -----------; -----------;

-----------; -----------; -----------; -----------;

}; }; }; };

/// Syntax of inheritance

class <base/super/parent\_class>

{

........;

};

class <derived/sub/child\_class> **extends** <base/super/parent\_class>

{

--------;

--------;

};

let’s see with the example

class Person

{

char name[50];

int age;

-----------;

}

class employee extends Person class student extends Person class Teacher extends Person class Driver extends Person

{ { { {

-----------; -----------; -----------; -----------;

-----------; -----------; -----------; -----------;

-----------; -----------; -----------; -----------;

}; }; }; };

**/// Types of Inheritance in C++**

single multi-level hierarchical Multiple Hybrid

Inheritance Inheritance Inheritance Inheritance Inheritance

[ ] A [ ] A [ ] A A B

| | | [ ] [ ]

| [ ] B -------------- |\_\_\_\_\_\_\_|

V | | | |

[ ] B [ ] C [ ]B [ ]c [ ] C

- Single inheritance: One-to-one relationship, two layers

- Multi-level Inheritance: one-to-one relationship, more than two layers

- hierarchical Inheritance: One to many relationship

- Multiple Inheritance: many-to-one relationship

- Hybrid Inheritance: combinations of any two or more inheritance

To avoid ambiguity errors java eliminated the Multiple and Hybrid inheritance. To achieve the plus

points of these inheritances’ java introduced the **"interface."**

So, Java having only 3 inheritances

- single Inheritance: One to one relationship, two layers

- Multi-level Inheritance: one to one relationship, more than two layers

- hierarchical Inheritance: One to many relationship

- single Inheritance:

[ ] Person [ ] Person

| |

| or |

[ ] Student [ ] Employee

- if we include both in single program, it become hierarchical Inheritance.

import java.util.Scanner;

class Person

{

private String nm="";

private int age;

public void in()

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the name of person: ");

nm=sc.nextLine();

System.out.println("\n Enter the age of person: ");

age=sc.nextInt();

}

public void out()

{

System.out.println("\n Name: "+nm+"\t age: "+age);

}

}

class Student extends Person

{

private int rno;

private double per;

public void input()

{

in();

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the rno of Student: ");

rno=sc.nextInt();

System.out.println("\n Enter the percentage of Student: ");

per=sc.nextDouble();

}

public void output()

{

out();

System.out.println("\n RNO: "+rno+"\t Percentage: "+per);

}

}

class MainStudent

{

public static void main(String []args)

{

Student s=new Student();

s.input();

s.output();

}

}

//--------------------------------------------------------------------------------

Or can be called as

import java.util.Scanner;

class Person

{

private String nm="";

private int age;

public void in()

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the name of person: ");

nm=sc.nextLine();

System.out.println("\n Enter the age of person: ");

age=sc.nextInt();

}

public void out()

{

System.out.println("\n Name: "+nm+"\t age: "+age);

}

}

class Student extends Person

{

private int rno;

private double per;

public void input()

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the rno of Student: ");

rno=sc.nextInt();

System.out.println("\n Enter the percentage of Student: ");

per=sc.nextDouble();

}

public void output()

{

System.out.println("\n RNO: "+rno+"\t Percentage: "+per);

}

}

class MainStudent

{

public static void main(String []args)

{

Student s=new Student();

s.in();

s.input();

s.out();

s.output();

}

}

As the in() and out() from Person class decl. as public, visibility remains as it is

in derived class. so, both can be called from main() or from methods of child class.

**- Multilevel Inheritance:**

[ ] Person in() and out()

|

|

[ ] Student input() and output()

|

|

[ ] Sport set() and show()

import java.util.Scanner;

class Person

{

private String nm="";

private int age;

public void in()

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the name of person: ");

nm=sc.nextLine();

System.out.println("\n Enter the age of person: ");

age=sc.nextInt();

}

public void out()

{

System.out.println("\n Name: "+nm+"\t age: "+age);

}

}

class Student extends Person

{

private int rno;

private double per;

public void input()

{

in();

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the rno of Student: ");

rno=sc.nextInt();

System.out.println("\n Enter the percentage of Student: ");

per=sc.nextDouble();

}

public void output()

{

out();

System.out.println("\n RNO: "+rno+"\t Percentage: "+per);

}

}

class Sport extends Student

{

private int points;

public void set()

{

input();

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the grade points of Student: ");

points=sc.nextInt();

}

public void show()

{

output();

System.out.println("\n Points: "+points);

}

}

class MainStudentMultilevel

{

public static void main(String []args)

{

Sport s=new Sport();

s.set();

s.show();

}

}

//--------------------------------------------------------------------------------

/// Using the constructors in inheritance

class Base

{

private int a;

public Base() //constructor

{

System.out.println("\n In Base default");

a=10;

}

public Base(int x)

{

System.out.println("\n In Base para");

a=x;

}

public void showBase()

{

System.out.println("\n a="+a);

}

}

class ImdBase extends Base

{

private int b;

public ImdBase()

{

System.out.println("\n In ImdBase default");

b=20;

}

public ImdBase(int x, int y)

{

super(x);

// (super keyword is used to call the base class’s para constructor, and it should be called first in the method)

System.out.println("\n In ImdBase para");

b=y;

}

public void showImdBase()

{

showBase();

System.out.println("\n b="+b);

}

}

class Derived extends ImdBase

{

private int c;

public Derived()

{

System.out.println("\n In Derived default");

c=30;

}

public Derived(int x, int y, int z)

{

super(x,y);

System.out.println("\n In Derived para");

c=z;

}

public void showDerived()

{

showImdBase();

System.out.println("\n c="+c);

}

}

class MainPassParamInh

{

public static void main(String []args)

{

Derived ob1=new Derived();

ob1.showDerived();

Derived ob2=new Derived(100,200,300);

ob2.showDerived();

}

}

///=========================================================================================================================

**// Hierarchical Inheritance**

[ ] Person

|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| | |

[ ] student [ ] Employee [ ] Driver

| | |

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

[ ] sport ------------------- | |

| | | | [ ] Utility [ ] Heavy

| [ ] [ ] [ ]

[ ] Result Worker Director CEO

///=========================================================================================================================

We have two more inheritances multiple and hierarchical, these inheritances are present in C++

but as both create the Ambiguous condition both are eliminated from Java.

NOTE THAT TO GAIN A GOOD PART OF THESE INHERITANCES JAVA INTRODUCED THE INTERFACE.

Let's see, how the ambiguous condition is created by the both

// Multiple Inheritance:

student sport

get() [ ] [ ] in()

put() | | put()

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

|

|

[ ]

Result

get() in()

put() put() //two methods of same name

input()

output()

// Hybrid Inheritance:

in() Student

out() [ ] ---|

| |

\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | -----------> // Hierarchical Inheritance

in() | | in()

out() | | out()|

Show() Set() Exam [] sport [] setdata() showdata()

| |

|\_\_\_\_\_\_\_\_o\_\_\_\_\_\_\_\_\_\_\_\_\_\_| ----------> // Multiple Inheritance

| |

[ ] ----

in() out() Result in() out()

Set() Show() setdata() showdata()

///==========================================================================================================================

**/// Method Overriding**

When the base class function is redefined in the child class, we call the same function

using the object of the child class. then the native copy of the child class will be executed

and the inherited copy from a base class will be overruled by the native copy of function. which

is called as function overriding.

class A [ ] show(){..}

|

|

class B [ ] show(){...} --> Own copy of B

show(){..} --> Inhe. from A

suppose we create an object of class B,

i.e. B ob=new B();

ob.show(); --> the owned copy of B will be executed.

which overrides the inh. copy of show()

/\*

import java.util.Scanner;

class Person

{

private String nm="";

private int age;

public void in()

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the name of person: ");

nm=sc.nextLine();

System.out.println("\n Enter the age of person: ");

age=sc.nextInt();

}

public void out()

{

System.out.println("\n Name: "+nm+"\t age: "+age);

}

}

class Student extends Person

{

private int rno;

private double per;

public void input()

{

in();

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the rno of Student: ");

rno=sc.nextInt();

System.out.println("\n Enter the percentage of Student: ");

per=sc.nextDouble();

}

public void out()

{

// out(); // recursive call i.e. it will call its own copy of out()

super.out(); // it will call base class method out()

System.out.println("\n RNO: "+rno+"\t Percentage: "+per);

}

}

class MainStudentOverriding

{

public static void main(String []args)

{

Student s=new Student();

s.input();

s.out();

}

}

//------------------------------------------------------------------

**// Accessing the inherited instance variable using the super keyword**

class Base

{

public int a;

Base()

{

a=10;

}

}

class Derived extends Base

{

public int a;

Derived()

{

a=100;

}

public void out()

{

System.out.println("\n In Derived Inh a="+super.a+" and a="+a);

}

}

class DemoInitBlockAndSuper\_InstanceVariable

{

public static void main(String []args)

{

Derived ob=new Derived();

ob.out();

}

}

//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------

**/// Using the final keyword**

1. Final Variables:

When a variable is declared as final, its value cannot be changed after initialization.

This is often used for constants Decl.

/\*

public class FinalVariableExample {

public static void main(String[] args) {

final int MAX\_VALUE = 100;

// MAX\_VALUE = 200; // Compilation error, as MAX\_VALUE is final

System.out.println("Maximum value: " + MAX\_VALUE);

}

}

class DemoFinal

{

final public int x;

public DemoFinal()

{

x=10;

}

public void show()

{

System.out.println("\n x="+x);

}

}

class MainFinal

{

public static void main(String []args)

{

DemoFinal ob=new DemoFinal();

// ob.x=20;

ob.show();

}

}

- Note that we have to assign the value to final variable either using

= operator directly or using initilizer block or by using default

constructor.

- when we not assign any value, in that case it will generate an error message as

"MainFinal.java:16: error: variable x might not have been initialized"

- If we try to assign any other value we will face error as

"MainFinal.java:27: error: cannot assign a value to final variable x"

//-----------------

2. Final Methods: When a method is declared as final, it cannot be

overridden by subclasses.

class Parent {

final void display() {

System.out.println("This is a final method.");

}

}

class Child extends Parent {

// Compilation error: Cannot override the final method from Parent

void display() { }

}

class MainFinalMethodExample

{

public static void main(String []args)

{

Child ob=new Child();

ob.display();

}

}

Output: (Compile time error)

MainFinalMethodExample.java:56: error: display() in Child cannot override display() in Parent

void display() { }

^

overridden method is final

1 error

Press any key to continue . . .

//--------------------

3.Final Classes:

- When a class is declared as final, it cannot be extended

by other classes.

final class Person

{

}

class Student extends Person

{

}

class mainDemoFinalClass

{

public static void main(String []args)

{

Student ob=new Student();

}

}

output: (Comile time error)

MainFinalMethodExample.java:83: error: cannot inherit from final Person

class Student extends Person

^

1 error

Press any key to continue . . .

//----------------------------------------

4. Final Parameters:

When a parameter is declared as final in a method, it means that

the value of the parameter cannot be changed within the method.

\*/

class Demo

{

private int x;

public void in(final int t)

{

t=78;

x=t;

}

public void show()

{

System.out.println("\n x="+x);

}

}

class DemoFinalParameter

{

public static void main(String []args)

{

Demo ob=new Demo();

ob.in(100);

ob.show();

}

}

Output (Error on Compie)

DemoFinalParameter.java:114: error: final parameter t may not be assigned

t=78;

^

1 error

Press any key to continue . . .

//------------------------------------------------------------------------------

// Polymorphism in Java: It refers one name many forms. there are two

different types as discussed below.

the word polymorphism forms as shown

- Poly: many or more

- morphism: forms or copies

Polymorphism

|

-----------------------

| |

Runtime / Compile-time/

dynamic binding / static binding/

dynamic polymorphism / static polymorphism /

late binding early binding

| |

Overriding Overloading

- Function Overloading

We have seen, the method overloading and Constructor overloading these

are the examples of static/compile time polymorphism.

lets see Runtime polymorphism now,

shape

[ ]

|

|

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

| | |

[ ] [ ] [ ]

Circle Rect Triangle

area() area() area()

class Shape

{

}

class Circle extends Shape

{

public void area()

{

System.out.println("In Circle area() method");

}

}

class Rect extends Shape

{

public void area()

{

System.out.println("In Rect area() method");

}

}

class Triangle extends Shape

{

public void area()

{

System.out.println("In Triangle area() method");

}

}

class DemoRuntimePoly

{

public static void main(String []args)

{

Shape s=null;

s=new Rect();

s.area();

}

}

output (on Compile)

DemoRuntimePoly.java:35: error: cannot find symbol

s.area();

^

symbol: method area()

location: variable s of type Shape

1 error

Press any key to continue . . .

//---------------------------------------------------

class Shape

{

}

class Circle extends Shape

{

public void area()

{

System.out.println("In Circle area() method");

}

}

class Rect extends Shape

{

public void area()

{

System.out.println("In Rect area() method");

}

}

class Triangle extends Shape

{

public void area()

{

System.out.println("In Triangle area() method");

}

}

class DemoRuntimePoly

{

public static void main(String []args)

{

Shape s=null;

s=new Rect();

s.area();

s=new Circle();

s.area();

s=new Triangle();

s.area();

}

}

output: (On Compile)

DemoRuntimePoly.java:35: error: cannot find symbol

s.area();

^

symbol: method area()

location: variable s of type Shape

DemoRuntimePoly.java:38: error: cannot find symbol

s.area();

^

symbol: method area()

location: variable s of type Shape

DemoRuntimePoly.java:41: error: cannot find symbol

s.area();

^

symbol: method area()

location: variable s of type Shape

3 errors

Press any key to continue . . .

Note that it is searching the method area() in the base class Shape.

as it is not present over there, it is generating the error message.

let's try after defining the area() in the base class shape.

class Shape

{

public void area()

{

System.out.println("In Shape area() method");

}

}

class Circle extends Shape

{

public void area()

{

System.out.println("In Circle area() method");

}

}

class Rect extends Shape

{

public void area()

{

System.out.println("In Rect area() method");

}

}

class Triangle extends Shape

{

public void area()

{

System.out.println("In Triangle area() method");

}

}

class DemoRuntimePoly

{

public static void main(String []args)

{

Shape s=null;

s=new Rect();

s.area();

s=new Circle();

s.area();

s=new Triangle();

s.area();

}

}

output:

In Rect area() method

In Circle area() method

In Triangle area() method

Press any key to continue . . .

Just analyze the output, without using virtual keyword the methods of

class object where the reference of shape is pointing is executed

this inteligence due to fact which is known as "Dynamic method Dispatch"

and therefore virtual keyword not present is Java.

//------------------------------------------------

Note that it is not compulsory to all childs of Shape class to

override the area() method. So to make compulsory in C++ pure

virtual method is there in base but as virtual keyword is absent

in java, it is done with the help of abstract keyword.

shape

[ ]

|

|

----------|--------==================

| | | |

[ ] [ ] [ ] [ ]

Circle Rect Triangle Parabola

area() area() area()

abstract class Shape

{

abstract public void area();

}

class Circle extends Shape

{

public void area()

{

System.out.println("In Circle area() method");

}

}

class Rect extends Shape

{

public void area()

{

System.out.println("In Rect area() method");

}

}

class Triangle extends Shape

{

}

class DemoRuntimePoly

{

public static void main(String []args)

{

Shape s=null;

s=new Rect();

s.area();

s=new Circle();

s.area();

s=new Triangle();

s.area();

}

}

- Output on compile when only area() method in Shape decl. as abstract and tringle does not override area():

DemoRuntimePoly.java:139: error: Shape is not abstract and does not override abstract method area() in Shape

class Shape

^

1 error

Press any key to continue . . .

- Output on compile when Shape class and area() method from it decl. as abstract and tringle does not override area():

DemoRuntimePoly.java:159: error: Triangle is not abstract and does not override abstract method area() in Shape

class Triangle extends Shape

^

1 error

Press any key to continue . . .

//------------------------------------------------------

// After overriding area() in class Triangle and class shape and its area() as a abstract:

abstract class Shape

{

abstract public void area();

}

class Circle extends Shape

{

public void area()

{

System.out.println("In Circle area() method");

}

}

class Rect extends Shape

{

public void area()

{

System.out.println("In Rect area() method");

}

}

class Triangle extends Shape

{

public void area()

{

System.out.println("In Triangle area() method");

}

}

class DemoRuntimePoly

{

public static void main(String []args)

{

Shape s=null;

s=new Rect();

s.area();

s=new Circle();

s.area();

s=new Triangle();

s.area();

}

}

Output:

In Rect area() method

In Circle area() method

In Triangle area() method

Press any key to continue . . .

**Abstract Class and Abstract Methods:**

**Abstract Class:**

In Java, an abstract class is a class that cannot be instantiated on its own and is meant to be subclassed by other classes. It serves as a blueprint for other classes, providing common functionality and structure. Abstract classes can have both abstract and concrete methods.

Here are key characteristics of abstract classes in Java:

- It is decared as an abstract using abstract keyword.

abstract class Demo

{

.......

}

- Abstract class have atleast one abstract method, and child class must override the all abstract methods of base otherwise we need to declare the child class as abstract, and remember that we are ***not allowed to create the object of an abstract class.***

abstract class Demo

{

abstract void show();

.......

}

- Abstract classes can also have regular (concrete) methods with a complete implementation

abstract class Demo

{

abstract void show();

public void display()

{

}

.......

}

- Objects cannot be created from abstract classes using the new keyword. They are meant to be extended by subclasses.

- Abstract classes can have constructors, and they are called when a subclass object is created.

- Abstract classes can be used as a base class for other classes. Subclasses extend the abstract class and provide concrete implementations for the abstract methods.

- Abstract classes can have instance variables just like regular classes.

abstract class Demo

{

private int x=10;

public Demo()

{

System.out.println("\n In the Demo Class Constructor");

}

public void out()

{

System.out.println("\n In out() method - x is "+x);

}

abstract void show();

}

class Test extends Demo

{

int y;

public Test()

{

y=100;

System.out.println("\n In child Test class Constructor");

}

public void show()

{

System.out.println("\n Hi from show() Test and y="+y);

}

}

class DemoAbstractExample

{

public static void main(String[] args)

{

Test ob=new Test();

ob.show();

ob.out();

}

}

- The inner class can be declared abstract by declaring it local.

- A static method and constructor can be part of an abstract class.

- When the final keyword is used, the abstract keyword cannot be used.

- Abstract methods cannot be declared private.

- Abstract methods cannot be declared static.

- An abstract keyword cannot be used with variables or constructors.

//---------------------------------------------

**Abstract method:**

In Java, an abstract method is a method that is declared without an implementation in an abstract class. An abstract class is a class that cannot be instantiated on its own and may contain abstract methods, which are meant to be implemented by concrete (non-abstract) subclasses. Abstract methods serve as a blueprint for concrete classes to provide their implementation.

Here are the key points about abstract methods in Java:

- The abstract method decleared as abstract using abstract keyword and it don’t have body.

i.e. abstract void dispay();

- When any method decleared as abstract, the containing class must be decleared as abstract.

i.e.

abstract class Test

{

abstract void display();

....

}

- Subclasses that extend an abstract class must provide implementations for all the abstract methods declared in the superclass, otherwise, you have to declare that class as abstract.

abstract class Test

{

abstract void display();

....

}

class Demo extends Test

{

.....

public void dispay()

{

.....

}

....

}

- Abstract methods provide a way to define a common interface for a group of related classes, ensuring that each concrete subclass provides its implementation. This promotes code reusability and helps in creating a consistent interface for different classes.

//---------------------------------------------------------------------------------------------------

//---------------------------------------------------------------------------------------------------

**Interface in Java:**

It is introduced to achieve the advantages of multiple inheritance in Java.

In interfaces, all methods are ***implicitly abstract and public***. They don't use the abstract keyword.

*interface variables in Java are constants, and they are implicitly* ***public, static, and final.***

***We can’t create an object of an interface.***

Decl. Syntax:

interface <interface\_nm>

{

---------------;

}

e.g.

interface area

{

static final float pi=3.14F;

void compute(float x, float y); //abstract method

void show(); //abstract method

}

A class can extend only one class but it may implements one or more than one interface

In C++ In Java

class Exam class Sport class Exam interface Sport

show() [ ] show()[ ] show()[ ] [ ]show() -abstract

| | | |

|\_\_\_\_\_\_\_\_\_\_\_\_\_ | |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

| show() |

show() [ ] show() [ ] show(){}

class Result: public Exam, Public Sport class Result extends Exam implements Sport

Result rob; Result rob=new Result();

rob.show(); // ambiguous error rob.show(); // No Error

- if() result class having his copy of show() then there will not be an ambiguous error.

/\*

abstract class Test

{

abstract public void in();

}

\*/

interface Test

{

void in();

}

class Demo implements Test

{

public void in()

{

}

}

class DemoMain

{

public static void main(String []args)

{

Demo ob=new Demo();

}

}

//--------------------------------------------------------------------

**// class implements interface**

import java.util.Scanner;

interface Sample

{

void in();

void out();

}

class Test implements Sample

{

private int x;

private double y;

public Test()

{

x=23;

y=45.21;

}

public void in() // *visibility must be public*

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the values of x and y: ");

x=sc.nextInt();

y=sc.nextDouble();

}

public void out() // visibility must be public

{

System.out.println("\n x="+x+"\t y="+y);

}

}

class MainDemoInterfaceExample

{

public static void main(String []args)

{

Test ob=new Test();

ob.in();

ob.out();

}

}

///-----------------------------------------------------------------------------------------------

**// class extends the class and implements the interface**

import java.util.Scanner;

interface Sample

{

void in();

void out();

}

class Demo

{ private int a;

public Demo()

{

a=100;

}

public void show()

{

System.out.println("\n a="+a);

}

}

class Test extends Demo implements Sample

{

private int x;

private double y;

public Test()

{

x=23;

y=45.21;

}

public void in() // *visibility must be public*

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the values of x and y: ");

x=sc.nextInt();

y=sc.nextDouble();

}

public void out() // visibility must be public

{

show();

System.out.println("\n x="+x+"\t y="+y);

}

}

class MainDemoInterfaceExample

{

public static void main(String []args)

{

Test ob=new Test();

ob.in();

ob.out();

}

}

//------------------------------------------------------------------------------------------

**// A class extends the class and implements multiple interfaces**.

import java.util.Scanner;

interface SampleIn

{

void in();

}

interface SampleOut

{

void out();

}

class Demo

{ private int a;

public Demo()

{

a=100;

}

public void show()

{

System.out.println("\n a="+a);

}

}

class Test extends Demo implements SampleIn, SampleOut

{

private int x;

private double y;

public Test()

{

x=23;

y=45.21;

}

public void in() *// visibility must be public*

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the values of x and y: ");

x=sc.nextInt();

y=sc.nextDouble();

}

public void out() // visibility must be public

{

show();

System.out.println("\n x="+x+"\t y="+y);

}

}

class MainDemoInterfaceExample

{

public static void main(String []args)

{

Test ob=new Test();

ob.in();

ob.out();

}

}

//--------------------------------------------------------------------------------------------------------------------

**// An inteface extends another interface**

import java.util.Scanner;

interface SampleIn

{

void in();

}

interface SampleOut extends SampleIn

{

void out();

}

class Demo

{ private int a;

public Demo()

{

a=100;

}

public void show()

{

System.out.println("\n a="+a);

}

}

class Test extends Demo implements SampleOut

{

private int x;

private double y;

public Test()

{

x=23;

y=45.21;

}

public void in() // visibility must be public

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the values of x and y: ");

x=sc.nextInt();

y=sc.nextDouble();

}

public void out() // visibility must be public

{

show();

System.out.println("\n x="+x+"\t y="+y);

}

}

class MainDemoInterfaceExample

{

public static void main(String []args)

{

Test ob=new Test();

ob.in();

ob.out();

}

}

//----------------------------------------------------------------------------------------------------------------------

**// refering the base class object using interface reference**

import java.util.Scanner;

interface SampleOut

{

void out();

}

class Demo

{ private int a;

public Demo()

{

a=100;

}

public void show()

{

System.out.println("\n a="+a);

}

}

class Test extends Demo implements SampleOut

{

private int x;

private double y;

public Test()

{

x=23;

y=45.21;

}

public void in() // *visibility must be public*

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter the values of x and y: ");

x=sc.nextInt();

y=sc.nextDouble();

}

public void out() // *visibility must be public*

{

System.out.println("\n x="+x+"\t y="+y);

}

}

class MainDemoInterfaceExample

{

public static void main(String []args)

{

SampleOut ob=new Test();

ob.in();

ob.out();

}

}

output on compile:

MainDemoInterfaceExample.java:33: error: cannot find symbol

ob.in();

^

symbol: method in()

location: variable ob of type SampleOut

1 error

Press any key to continue . . .

* ***Note that an interface reference able to refer an object of its every child but using it we can access only those methods which are declared with that interface.***

//----------------------------------------------------------------------------------------------------------------------------------

In version 8

- Java interface can have default methods

- Java Interface may contain static methods

interface A

{

default void show()

{

}

static void xxx()

{

}

}

///-----------------------------------------------------------------------------------------

**Packages in Java:**

It is a Collection of classes and interfaces.

There are two types of packages ----> Built-in packages --> provided with JDK

|-> User defined packages -> defined by programmer

Java provides the standard library known as JSL or API, which is divided into different packages as shown (It is the same as that of the file folder system)

java

|

------------------------------------------------------------------

| | | | | | |

lang util io awt net applet sql

|

Scanner

It contains some other sub-packages, classes, and interfaces together with the Scanner class So it can be referred to as java.util.Scanner as this is a provided class, we import it from the JSL using the import statement as

***import java.util.Scanner***

You know the concept of path

we used the command >>set path=c:\program files\......\bin

which will gives the location of java compiler.

in same manner class path command is there, which will gives the location from

where we have to load the class when that class not present at current location or

not present at default path(i.e.in the JSL)

Defining our own package:

---------------------------

Java source file can be any (or all) of the following four internal parts:

- A single package statement (optional)

- Any number of import statements (optional)

- A single public class declaration (required)

- Any number of classes private

// Importing predefined class

// main class: E:\javaprogramsdacoe -> MainPackageDemoExample.java

import java.util.Scanner;

public class MainPackageDemoExample

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter any number: ");

int no=sc.nextInt();

}

}

//-----------------------------------------------------------------------------------------------------------------------------------

**// Using class from the same file (means the same package)**

// Using class from file - i.e. same package

***Note - All the members i.e. private, public, protected, and default are accessible within the class and the private member not accessible directly within same package***.

//ArithOperations class: E:\javaprogramsdacoe\MainPackageDemoExample.java -> ArithOperations.java (same file)

// same package means write above as E:\javaprogramsdacoe\ -> ArithOperations.java (Same package)

// (same file and same package are exactly same)

package javaprogramsdacoe;

import java.util.Scanner;

class ArithOperations

{

private int ans; // visibility - private

public int b;

protected int c;

int d;

public ArithOperations() // constructor

{

ans=0;

b=10;

c=100;

d=1000;

}

//.....

public void add10(int no) // visibility - public

{

ans=10+no;

}

int getAns() // visibility - default

{

return ans;

}

}

//main class: E:\javaprogramsdacoe -> MainPackageDemoExample.java

public class MainPackageDemoExample

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter any number: ");

int no=sc.nextInt();

ArithOperations ob=new ArithOperations();

ob.add10(no);

System.out.println("ans after adding 10: "+ob.getAns());

// ob.ans=11; //MainPackageDemoExample.java:44: error: ans has private access in ArithOperations

ob.b=171;

ob.c=1781;

ob.d=511;

}

}

//-------------------------------------------------------------------------------------------------------------------------------

**// Using class from the same package: (non-subclass)**

// same package means write above as E:\javaprogramsdacoe\ -> ArithOperations.java (Same package)

// (same file and same package are the same)

// In the same package non subclass cannot access only private all others are accessible.

package javaprogramsdacoe;

import java.util.Scanner;

class ArithOperations

{

private int ans; // visibility - private

public int b;

protected int c;

int d;

public ArithOperations() //

{

ans=0;

b=10;

c=100;

d=1000;

}

public void add10(int no) // visibility - public

{

ans=10+no;

}

int getAns() // visibility - default

{

return ans;

}

}

//main class: E:\javaprogramsdacoe -> MainPackageDemoExample.java

public class MainPackageDemoExample

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter any number: ");

int no=sc.nextInt();

ArithOperations ob=new ArithOperations();

ob.add10(no);

System.out.println("ans after adding 10: "+ob.getAns());

// ob.ans=11; //MainPackageDemoExample.java:44: error: ans has private access in ArithOperations

ob.b=171;

ob.c=1781;

ob.d=511;

}

}

compile and run option

E:\javaprogramsdacoe>javac ArithOperations.java

E:\javaprogramsdacoe>javac MainPackageDemoExample.java

E:\javaprogramsdacoe>cd..

E:\>java javaprogramsdacoe.MainPackageDemoExample

Enter any number:

11

ans after adding 10: 21

// ---------------------------------------------------------------------------------------------------------

/// Subclass in same package

// In same package subclass unable to access only private all others are accessible.

package javaprogramsdacoe;

import java.util.Scanner;

class MyWork extends ArithOperations

{

public void setData()

{

ans=11; //MainPackageDemoExample.java:44: error: ans has private access in ArithOperations

b=171;

c=1781;

d=511;

}

public void showData()

{

//System.out.println("\n a="+a);

System.out.println("\n b="+b+"\t c="+c+"\t d="+d);

}

}

//main class: E:\javaprogramsdacoe -> MainPackageDemoExample.java

public class MainPackageDemoExample

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter any number: ");

int no=sc.nextInt();

MyWork ob=new MyWork();

ob.setData();

ob.showData();

}

}

/\*

on compile before commenting a

E:\>cd javaprogramsdacoe

E:\javaprogramsdacoe>javac ArithOperations.java

E:\javaprogramsdacoe>javac MainPackageDemoExample.java

MainPackageDemoExample.java:9: error: ans has private access in ArithOperations

ans=11; //MainPackageDemoExample.java:44: error: ans has private access in ArithOperations

^

1 error

\*/

/\*

After commenting ans

E:\javaprogramsdacoe>javac ArithOperations.java

E:\javaprogramsdacoe>javac MainPackageDemoExample.java

E:\javaprogramsdacoe>cd..

E:\>java javaprogramsdacoe.MainPackageDemoExample

Enter any number:

12

b=171 c=1781 d=511

\*/

//-------------------------------------------------------------------------------------------------------------

/// Sub class and from different package

// private and default not accessible

// E:\pack1\demo\ArithOperations.java

package demo;

public class ArithOperations

{

private int ans; // visibility - private

public int b;

protected int c;

int d;

public ArithOperations() //

{ //

ans=0;

b=10;

c=100;

d=1000;

}

//.....

public void add10(int no) // visibility - public

{

ans=10+no;

}

int getAns() // visibility - default

{

return ans;

}

}

//main class: E:\javaprogramsdacoe -> MainPackageDemoExample.java

package javaprogramsdacoe;

import demo.ArithOperations;

import java.util.Scanner;

class MyWork extends ArithOperations

{

public void setData()

{

ans=11; //MainPackageDemoExample.java:44: error: ans has private access in ArithOperations

b=171;

c=1781;

d=511;

}

public void showData()

{

//System.out.println("\n a="+a);

System.out.println("\n b="+b+"\t c="+c+"\t d="+d);

}

}

public class MainPackageDemoExample

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter any number: ");

int no=sc.nextInt();

MyWork ob=new MyWork();

ob.setData();

ob.showData();

}

}

/\*

Errors while compile and execution

E:\javaprogramsdacoe>javac MainPackageDemoExample.java

MainPackageDemoExample.java:5: error: cannot find symbol

class MyWork extends ArithOperations

^

//------------------------------------------------------------

E:\javaprogramsdacoe>set classpath=.\*;e:\pack1;

E:\javaprogramsdacoe>javac e:\pack1\demo\ArithOperations.java

E:\javaprogramsdacoe>javac MainPackageDemoExample.java

MainPackageDemoExample.java:4: error: ArithOperations is not public in demo; cannot be accessed from outside package

import demo.ArithOperations;

^

MainPackageDemoExample.java:6: error: ArithOperations is not public in demo; cannot be accessed from outside package

class MyWork extends ArithOperations

//---------------------------------------------------------------------

E:\javaprogramsdacoe>javac e:\pack1\demo\ArithOperations.java

e:\pack1\demo\ArithOperations.java:3: error: modifier protected not allowed here

protected class ArithOperations

//--------------------------------------------------------------------------

E:\javaprogramsdacoe>javac e:\pack1\demo\ArithOperations.java

E:\javaprogramsdacoe>javac MainPackageDemoExample.java

MainPackageDemoExample.java:10: error: ans has private access in ArithOperations

ans=11; //MainPackageDemoExample.java:44: error: ans has private access in ArithOperations

^

MainPackageDemoExample.java:13: error: d is not public in ArithOperations; cannot be accessed from outside package

d=511;

^

MainPackageDemoExample.java:18: error: d is not public in ArithOperations; cannot be accessed from outside package

System.out.println("\n b="+b+"\t c="+c+"\t d="+d);

^

3 errors

Note that as ans and d are private and default, thay are not accessible

\*/

//----------------------------------------------------------------------------------------------------------

// class from different package non-subclass

// - having only access to public

// E:\pack1\demo\ArithOperations.java

package demo;

public class ArithOperations

{

private int ans; // visibility - private

public int b;

protected int c;

int d;

public ArithOperations() //

{ //

ans=0;

b=10;

c=100;

d=1000;

}

//.....

public void add10(int no) // visibility - public

{

ans=10+no;

}

int getAns() // visibility - default

{

return ans;

}

}

//main class: E:\javaprogramsdacoe -> MainPackageDemoExample.java

package javaprogramsdacoe;

import demo.ArithOperations;

import java.util.Scanner;

public class MainPackageDemoExample

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter any number: ");

int no=sc.nextInt();

ArithOperations ob=new ArithOperations();

ob.add10(no);

System.out.println("ans after adding 10: "+ob.getAns());

ob.ans=11; //MainPackageDemoExample.java:44: error: ans has private access in ArithOperations

ob.b=171;

ob.c=1781;

ob.d=511;

}

}

/\*

Errors while compile and execution

E:\javaprogramsdacoe>javac MainPackageDemoExample.java

MainPackageDemoExample.java:16: error: getAns() is not public in ArithOperations; cannot be accessed from outside package

System.out.println("ans after adding 10: "+ob.getAns());

^

MainPackageDemoExample.java:18: error: ans has private access in ArithOperations

ob.ans=11; //MainPackageDemoExample.java:44: error: ans has private access in ArithOperations

^

MainPackageDemoExample.java:20: error: c has protected access in ArithOperations

ob.c=1781;

^

MainPackageDemoExample.java:21: error: d is not public in ArithOperations; cannot be accessed from outside package

ob.d=511;

^

4 errors

note: for class from different package non-subclass having only access to public

\*/

//--------------------------------------------------------------------------------------------------------------------

Access Protection Chart

private default protected public

same class Yes Yes Yes Yes

same package subclass No Yes Yes Yes

same package non-subclass No Yes Yes Yes

Different package subclass No No Yes Yes

Different package non-subclass No No No Yes

//================================================================================

**// Exception Handing**

Java Exception hierarchy

Here is a simplified diagram of the exception hierarchy in Java.

Throwable

|

-----------------------------------

| |

Error Exception

The Throwable class is the root class in the hierarchy.

Note that the hierarchy splits into two branches: Error and Except

**Errors:**

An error is the mistake in developing program causes a unexpected output or we are unable to execute the program.

The error may of two types

- Compile time errors:

- Run time errors:

**Compile time errors:** these are the syntax errors which are displayed by java compiler and therefore these are known as Compile time errors.

e.g. use of undeclared variable

bad reference to object

missing semi comma

misspelling of identifier and keyword

missing/mismatching brackets in classes and methods

**Run time errors:** some time the program may compile but gives error at the time of execution due to some mistakes at run time such errors are known as Run time errors

e.g. divide an integer by zero

access an element that is out of bounds of an array

attempting to use a negative size for an array

**Exceptions**

Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception such as the name and description of the exception and state of the program when the exception occurred.

now focus on different types of exceptions in Java.

Java Exception Types

The exception hierarchy also has two branches: RuntimeException and IOException.

**1. RuntimeException**

A runtime exception happens due to a programming error. They are also known as ***unchecked exceptions.***

These exceptions are not checked at compile-time but at run-time.

Some of the common runtime exceptions are:

- Improper use of an API - IllegalArgumentException

- Null pointer access (missing the initialization of a variable) - NullPointerException

- Out-of-bounds array access - ArrayIndexOutOfBoundsException

- Dividing a number by 0 - ArithmeticException

You can think about it in this way. ***“If it is a runtime exception, it is your fault”***.

The NullPointerException would not have occurred if you had checked whether the variable was initialized or not before using it.

An ArrayIndexOutOfBoundsException would not have occurred if you tested the array index against the array bounds.

**2. IOException**

An IO exception is also known as a ***checked exception***. They are checked by the compiler at the ***compile-time*** and the programmer is prompted to handle these exceptions.

Some of the examples of checked exceptions are:

- Trying to open a file that doesn’t exist results in FileNotFoundException

- Trying to read past the end of a file

***Java Exception Handling*** - *Note that handling the exception means not eliminating it.*

We know that exceptions abnormally terminate the execution of a program, to avoid it, it is important to handle exceptions. And it will allow us to make some corrective efforts.

There are 5 different keywords provided to handle the exceptions. and these are

***try catch finally throw throws***

try try try

{ { {

} } }

catch(Exception e) catch(Exception e) catch(XException e)

{ { {

--------- } }

} catch(YException e) finally

{ {

} }

catch(Exception e)

{

}

try try

{ {

} }

catch (ExceptionType1 Ob) finally

{ {

}

}

catch (ExceptionType2 Ob)

{

}

finally

{

}

//-------------------------------------------------------------------------------------

class DemoErrorException

{

public static void main()

{

double y;

x=10;

y=12.45;

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

}

}

/\*

E:\jprodyp>javac DemoErrorExceprion.java

DemoErrorExceprion.java:8: error: cannot find symbol

x=10;

^

symbol: variable x

location: class DemoErrorExceprion

DemoErrorExceprion.java:11: error: cannot find symbol

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

^

symbol: variable x

location: class DemoErrorExceprion

2 errors

\*/

//----------------------------------------

class DemoErrorException

{

public static void main(String []args)

{

int x;

double y;

x=args[0];

y=args[1];

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

}

}

E:\jprodyp>javac DemoErrorException.java

E:\jprodyp>javac DemoErrorException.java

DemoErrorException.java:9: error: incompatible types: String cannot be converted to int

x=args[0];

^

DemoErrorException.java:10: error: incompatible types: String cannot be converted to double

y=args[1];

^

2 errors

//-------------------------------------------------------------------------------------

class DemoErrorException

{

public static void main(String []args)

{

String s1;

String s2;

s1=args[0];

s2=args[1];

System.out.println("s1="+s1+"\t s2="+s2);

}

}

E:\jprodyp>javac DemoErrorException.java

E:\jprodyp>java DemoErrorException

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0

at DemoErrorException.main(DemoErrorException.java:9)

//----------------------------------------------------------------------------------

**// Using try-catch block**

class DemoErrorException

{

public static void main(String []args)

{

String s1="";

String s2="";

try

{

s1=args[0];

s2=args[1];

}

catch(Exception e)

{

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

System.out.println("One: "+e.getMessage());

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

System.out.println("Two: "+e);

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

System.out.println("Three:");

e.printStackTrace();

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

System.out.println("Four: Index Issue");

}

System.out.println("s1="+s1+"\t s2="+s2);

}

}

/\*

--------------- Run Without arguments -------------------

E:\jprodyp>javac DemoErrorException.java

E:\jprodyp>java DemoErrorException

---------------------------------

One: Index 0 out of bounds for length 0

---------------------------------

Two: java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0

---------------------------------

Three:

java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0

at DemoErrorException.main(DemoErrorException.java:11)

---------------------------------

Four: Index Issue

s1= s2=

------------------------- Run with single argument --------------------------------

E:\jprodyp>java DemoErrorException hello

---------------------------------

One: Index 1 out of bounds for length 1

---------------------------------

Two: java.lang.ArrayIndexOutOfBoundsException: Index 1 out of bounds for length 1

---------------------------------

Three:

java.lang.ArrayIndexOutOfBoundsException: Index 1 out of bounds for length 1

at DemoErrorException.main(DemoErrorException.java:12)

---------------------------------

Four: Index Issue

s1=hello s2=

------------------------------- Run with 2 arguments -----------------

E:\jprodyp>java DemoErrorException hello all

s1=hello s2=all

\*/

//----------------------------------------------------------------------------------------------

//------------------------------------------------------------------------------------

**// Using try Multiple catch blocks**

try to run the program and handle all possible exceptions

import java.util.Scanner;

class ExampleUsingMultipleCatch

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter any count: ");

int cnt=sc.nextInt();

int []x=new int[cnt];

for(int i=0;i<5;i++)

{

x[i]=Integer.parseInt(args[i]);

}

System.out.println("\n Enter any position: ");

int pos=sc.nextInt();

System.out.println("\n Value at position: "+x[pos]);

}

}

//---------------------------------------------------------------------------------------------

import java.util.\*;

class ExampleUsingMultipleCatch

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter any count: ");

try

{ int cnt=sc.nextInt();

int []x=new int[cnt];

for(int i=0;i<5;i++)

{

x[i]=sc.nextInt();

}

System.out.println("\n Enter any position: ");

int pos=sc.nextInt();

System.out.println("\n Value at position: "+x[pos]);

}

catch(InputMismatchException ime)

{

System.out.println("\n Immproper input");

}

catch(ArrayIndexOutOfBoundsException ae)

{

System.out.println("\n Index Issue");

}

catch(Exception e)

{

System.out.println("\n Execution problem");

}

System.out.println("\n End of program");

}

}

//----------------------------------------------------------------------------------------------

**Java finally block:**

In Java, the finally block is always executed no matter whether there is an exception or not.

The finally block is optional. And, for each try block, there can be only one finally block.

The finally block is typically used to perform cleanup operations or release resources that need to be executed regardless of whether an exception occurred.

import java.util.\*;

class ExampleUsingMultipleCatch

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("\n Enter any count: ");

try

{ int cnt=sc.nextInt();

int []x=new int[cnt];

for(int i=0;i<5;i++)

{

x[i]=sc.nextInt();

}

System.out.println("\n Enter any position: ");

int pos=sc.nextInt();

System.out.println("\n Value at position: "+x[pos]);

}

catch(InputMismatchException ime)

{

System.out.println("\n Immproper input");

}

catch(ArrayIndexOutOfBoundsException ae)

{

System.out.println("\n Index Issue");

}

catch(Exception e)

{

System.out.println("\n Execution problem");

}

finally

{

System.out.println("\n in finally block");

}

System.out.println("\n End of program");

}

}

//-----------------------------------------------------------------------------------------

**// Using throw keyword**

In Java, the throw statement is used to explicitly throw an exception. This allows you to create and throw your exceptions that are caught in your code. The throw statement is typically used inside the try block of a try-catch statement or within a method that declares to throw an exception using the throws keyword.

import java.util.Scanner;

class FormFill

{

private String name;

private int age;

private double per;

public FormFill()

{

name="";

age=0;

per=0.0;

}

public void input()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the name: ");

name=sc.nextLine();

System.out.println("Enter the age: ");

age=sc.nextInt();

if(age<=20 || age>30)

{

throw new ArrayIndexOutOfBoundsException();

}

System.out.println("Enter the percentage: ");

per=sc.nextDouble();

}

public void output()

{

System.out.println("Student Details: Name: "+name+"\n Age: "+age+"\t Percentage: "+per);

}

}

class FormFillProcess

{

public static void main(String []args)

{

FormFill ob=new FormFill();

ob.input();

ob.output();

}

}

/\*

E:\jprodyp>java FormFillProcess

Enter the name:

amit

Enter the age:

23

Enter the percentage:

90.45

Student Details: Name: amit

Age: 23 Percentage: 90.45

//---------------

E:\jprodyp>java FormFillProcess

Enter the name:

sumit

Enter the age:

33

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException

at FormFill.input(FormFill.java:22)

at FormFillProcess.main(FormFill.java:38)

\*/

//------------------------------------------------------------------------------------------

**// handling exception**

import java.util.Scanner;

class FormFill

{

private String name;

private int age;

private double per;

public FormFill()

{

name="";

age=0;

per=0.0;

}

public void input()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the name: ");

name=sc.nextLine();

System.out.println("Enter the age: ");

age=sc.nextInt();

if(age<=20 || age>30)

{

throw new ArrayIndexOutOfBoundsException("Problem due to age is not within age window");

}

System.out.println("Enter the percentage: ");

per=sc.nextDouble();

}

public void output()

{

System.out.println("Student Details: Name: "+name+"\n Age: "+age+"\t Percentage: "+per);

}

}

class FormFillProcess

{

public static void main(String []args)

{

FormFill ob=new FormFill();

try

{

ob.input();

ob.output();

}

catch(Exception e)

{

System.out.println(e.getMessage());

}

}

}

//--------------------------------------------------------------------------------------------

**// Using throws keyword**

In Java, the throws clause is used in a method signature to declare that the method may throw certain types of exceptions. This provides information to the caller about the method about the potential exceptions that need to be handled. The throws clause is used to delegate the responsibility of exception handling to the calling code.

- You can declare multiple exceptions in the throws clause, separating them with commas.

- Checked exceptions (those that extend Exception but not RuntimeException) must be declared in the throws clause. Unchecked exceptions (those that extend RuntimeException) do not need to be declared.

- If a method declares that it throws an exception, means it is not handing it. It is just to

inform to the caller about the his responsibility.

- Methods are not required to declare unchecked (runtime) exceptions in the throws clause.

import java.util.Scanner;

import java.io.\*;

class FormFill

{

private String name;

private int age;

private double per;

public FormFill()

{

name="";

age=0;

per=0.0;

}

public void input() throws IOException

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the name: ");

name=sc.nextLine();

System.out.println("Enter the age: ");

age=sc.nextInt();

if(age<=20 || age>30)

{

throw new IOException("Problem due to age is not within age window");

}

System.out.println("Enter the percentage: ");

per=sc.nextDouble();

}

public void output()

{

System.out.println("Student Details: Name: "+name+"\n Age: "+age+"\t Percentage: "+per);

}

}

class FormFillProcess

{

public static void main(String []args)

{

FormFill ob=new FormFill();

try

{

ob.input();

ob.output();

}

catch(Exception e)

{

e.printStackTrace();

}

} }

//-----------------------------------------------------------------------------------------------------------------------

**// User-defined Exception**

import java.util.Scanner;

import java.io.\*;

***class AgeException extends Exception***

{

private String message;

public AgeException()

{

super("Age Issue");// it will call para contractor of base class i.e. Exception class

}

public AgeException(String msg)

{

super(msg);

message=msg;

}

public String toString()

{

return "The Exception Due To: "+message;

}

}

class FormFill

{

private String name;

private int age;

private double per;

public FormFill()

{

name="";

age=0;

per=0.0;

}

public void input() throws IOException, AgeException

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the name: ");

name=sc.nextLine();

System.out.println("Enter the age: ");

age=sc.nextInt();

if(age<=20 || age>30)

{

throw new AgeException("Problem due to age is not within age window");

}

System.out.println("Enter the percentage: ");

per=sc.nextDouble();

}

public void output()

{

System.out.println("Student Details: Name: "+name+"\n Age: "+age+"\t Percentage: "+per);

}

}

class FormFillProcessUsingUDE

{

public static void main(String []args)

{

FormFill ob=new FormFill();

try

{

ob.input();

ob.output();

}

catch(AgeException e)

{

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

System.out.println("One: "+e.getMessage());

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

System.out.println("Two: "+e);

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

System.out.println("Three:");

e.printStackTrace();

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

//================================================================================

**/// Multi-Threading**

Multithreading in Java

What is a Single Thread?

A single thread in Java is basically a lightweight and the smallest unit of processing. Java uses threads by using a ***“Thread Class***”.

There are two types of thread – ***user thread*** and ***daemon thread*** (daemon threads are used when we want to clean the application and are used in the background).

When an application first begins, a user thread is created. Post that, we can create many user threads and daemon threads.

class ThreadingExample

{

public static void main(String[] args)

{

System.out.println("Hello World!");

}

}

**Advantages of single thread:**

Reduces overhead in the application as a single thread executes in the system

Also, it reduces the maintenance cost of the application.

//----------------------------------------------------------------------------------------------------------------------------------

**What is Multithreading in Java?**

Multithreading in Java is a process of executing two or more threads simultaneously to maximize the utilization of the CPU.

Multithreaded applications execute two or more threads and run concurrently. Hence, it is also known as ***Concurrency*** in Java. Each thread runs parallel to each other.

Multiple threads don’t allocate separate memory areas; hence they save memory. Also, context switching between threads takes less time.

**Advantages of multithread:**

The users are not blocked because threads are independent, and we can perform multiple operations at times

As such the threads are independent, the other threads won’t get affected if one thread meets an exception.

**Thread Life Cycle in Java**

There are various stages of the life cycle of a thread:

New

Runnable

Running

Waiting

Dead

**New:** In this phase, the thread is created using the class “Thread class”. It remains in this state till the program starts the thread. It is also known as the born thread.

**Runnable:** In this phase, the instance of the thread is invoked with a start method. The thread control is given to the scheduler to finish the execution. It depends on the scheduler, whether to run the thread.

**Running:** When the thread starts executing, then the state is changed to the “running” state. The scheduler selects one thread from the thread pool, and it starts executing in the application.

**Waiting:** This is the state in which a thread has to wait. As multiple threads are running in the application, there is a need for synchronization between threads. Hence, one thread has to wait, till the other thread gets executed. Therefore, this state is referred to as a waiting state.

**Dead:** This is the state in which the thread is terminated. The thread is running and as soon as it completes processing it is in a “dead state”.

//-----------------------------------------------------------------------------------------------------------------------------------

**There are two different ways of creating the Thread.**

**- Using Thread class**

class MyThread extends Thread

{

public void run()

{

…………

}

}

MyThread t = new MyThread();

t.start();

**- Using Runnable Interface**

class MyThread [extends <cls>] implements Runnable

{

public void run()

{

}

}

Thread t = new Thread(new MyThread());

t.start();

//---------------------------------------------------------------------------------------------------------------------------

**Using Thread class to create the Thread:**

In Java, the Thread class is part of the java.lang package and is used for creating and managing threads. Here are some important methods of the Thread class:

**public void run():**

This method contains the code that constitutes the new thread. It is the entry point for the thread.

**public void start():**

This method is used to start the execution of the thread. It internally calls the run method.

**public static void sleep(long milliseconds) throws InterruptedException:**

Causes the currently executing thread to sleep (temporarily ease execution) for the specified number of milliseconds.

**public final void join() throws InterruptedException:**

Waits for this thread to die. This method causes the current thread to pause execution until the thread on which it is called is terminated.

**public final void setPriority(int priority):**

Sets the priority of this thread. The priority is a value between Thread.MIN\_PRIORITY (1) and Thread.MAX\_PRIORITY (10).

**public final void setName(String name):**

Changes the name of the thread.

**public final String getName():**

Returns the name of the thread.

**public static Thread currentThread():**

Returns a reference to the currently executing thread object.

**public final boolean isAlive():**

Test if this thread is alive. A thread is alive if it has been started and has not yet died.

**public static void yield():**

A hint to the scheduler that the current thread is willing to yield its current use of a processor.

**public final void setDaemon(boolean on):**

Marks this thread as either a daemon thread or a user thread. Daemon threads are terminated when all non-daemon threads have been completed.

**public final void interrupt():**

Interrupts this thread. It causes the InterruptedException to be thrown in the thread if it's in a blocked state.

These methods are essential for managing the lifecycle, behavior, and characteristics of threads in Java.

class MyThread extends Thread

{

public void run()

{

for ( int i = 1; i <= 10 ; i++)

{

System.out.println ( getName() + " " + i );

try { Thread.sleep(500);}catch(Exception e){}

}

}

}

class ThreadDemo

{

public static void main(String a[])

{

MyThread t = new MyThread();

t.setName("demo");

t.start();

MyThread t1 = new MyThread();

t1.setPriority(9);

t1.setName("xyz");

t1.start();

for ( int i = 1; i <= 10 ; i++)

{

System.out.println ( "Main " + i );

try { Thread.sleep(700);}catch(Exception e){}

}

}

}

//--------------------------------------------------------------------------------------------------------------------------

**// Using runnable interface**

class NewThread implements Runnable

{

public void run()

{

try {

for(int i = 5; i > 0; i--)

{

System.out.println("Child Thread: " + i);

Thread.sleep(500);

}

}catch (InterruptedException e){

System.out.println("Child interrupted.");

}

System.out.println("Exiting child thread.");

}

}

class ThreadDemoUsingRunnable

{

public static void main(String args[])

{

Thread t=new Thread(new NewThread());

t.start(); // Start the thread

Thread t1=new Thread(new NewThread());

t1.start(); // Start the thread

try

{

for(int i = 5; i > 0; i--)

{

System.out.println("Main Thread: " + i);

Thread.sleep(1000);

}

}

catch (InterruptedException e)

{

System.out.println("Main thread interrupted.");

}

System.out.println("Main thread exiting.");

}

}

//------------------------------------------------------------------------------------------

**// Alternative Way**

class Test implements Runnable

{

private Thread t;

public Test()

{

t=new Thread(this);

t.start();

}

public Test(String nm)

{

t=new Thread(this);

t.setName(nm);

t.start();

}

public void run()

{

try

{

for(int i = 5; i > 0; i--)

{

System.out.println(Thread.currentThread().getName() + i);

Thread.sleep(500);

}

}

catch (InterruptedException e)

{

System.out.println("Child interrupted.");

}

System.out.println("Exiting child thread.");

}

}

class DemoRunnableExample

{

public static void main(String[] args)

{

MyThread t1=new MyThread("One");

MyThread t2=new MyThread("Two");

try

{

for(int i = 5; i > 0; i--)

{

System.out.println("Main Thread: " + i);

Thread.sleep(1000);

}

}

catch (InterruptedException e)

{

System.out.println("Main thread interrupted.");

}

System.out.println("Main thread exiting.");

}

}

//-----------------------------------------------------------------------------------------------------

**// Using the isAlive() and join() methods:**

class MyThread extends Thread

{

public MyThread(String nm)

{

super(nm);

}

public void run()

{

for ( int i = 1; i <= 10 ; i++)

{

System.out.println ( getName() + " " + i );

try { Thread.sleep(500);}catch(Exception e){}

}

}

}

class ThreadMainIsAliveJoin

{

public static void main(String a[])

{

MyThread t1 = new MyThread("One-");

t1.setPriority(10);

t1.start();

MyThread t2 = new MyThread("Two-");

t2.setPriority(1);

t2.start();

MyThread t3 = new MyThread("Three-");

t3.setPriority(1);

t3.start();

MyThread t4 = new MyThread("Four-");

t4.setPriority(1);

t4.start();

System.out.println (" T1 is alive: "+t1.isAlive());

System.out.println (" T2 is alive: "+t2.isAlive());

System.out.println (" T3 is alive: "+t3.isAlive());

System.out.println (" T4 is alive: "+t4.isAlive());

try { t1.join();}catch(Exception e){}

for ( int i = 1; i <= 10 ; i++)

{

System.out.println ( "Main " + i );

try { Thread.sleep(700);}catch(Exception e){}

}

System.out.println (" T1 is alive: "+t1.isAlive());

System.out.println (" T2 is alive: "+t2.isAlive());

System.out.println (" T3 is alive: "+t3.isAlive());

System.out.println (" T4 is alive: "+t4.isAlive());

}

}

//------------------------------------------------------------------------------------------------------------------------

**- Thread Synchronization**

When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called synchronization. Java implements synchronization through use of the synchronized keyword.

class Demo

{

synchronized void show()

{

System.out.println("Thread Starts");

try

{ for(int i=0;i<5;i++)

{

System.out.println("i="+i);

Thread.sleep(2000);

}

}

catch(Exception e)

{

System.out.println("Thread stops");

}

}

}

class MyThread extends Thread

{

Demo t;

MyThread(Demo p)

{

t=p;

}

public void run()

{

t.show();

}

}

class SynchDemo

{

public static void main(String a[])

{

Demo d=new Demo();

MyThread t1=new MyThread(d);

MyThread t2=new MyThread(d);

MyThread t3=new MyThread(d);

t1.start();

t2.start();

t3.start();

}

}

//--------------------------------------------------------------------------------------------------------------------

**Deadlock:**

A special type of error that you need to avoid that relates specifically to multitasking is deadlock, which occurs when two threads have a circular dependency on a pair of synchronized objects.

//-------------------------------------------------------------------------------------------------------------------------------

What is Daemon Thread in Java?

Daemon thread in Java is a low-priority thread that performs background operations such as garbage collection, finalizer, Action Listeners, Signal dispatches, etc.

Daemon thread in Java is also a service provider thread that helps the user thread. Its life is at the mercy of user threads; when all user threads expire, JVM immediately terminates this thread.

In simple words, we can say that it provides services to user threads for background-supporting tasks. Daemon thread in Java has no role in life other than to serve user threads.

Properties of Daemon Thread in Java

It's a thread with the lowest priority possible.

They won't be able to stop the JVM from quitting once all of the user threads have completed their tasks.

When all user threads have completed their execution, the JVM terminates.

If JVM finds a running daemon thread, it terminates the thread and, after that, shutdown it.

The JVM is unconcerned about whether the Daemon thread is active or not.

The nature of a demon is passed down from parent to child. That is, if the parent is a Daemon, the child will be a Daemon as well, and if the parent is a non-daemon, the child will be a non-daemon as well.

Methods for Daemon Thread in Java by Thread Class

- public void setDaemon(boolean status)

This method marks whether the current thread as a daemon thread or a user thread.

- public final boolean isDaemon()

This method is used to determine whether or not the current thread is a daemon. If the thread is Daemon, it returns true.

Otherwise, false is returned.

public class DemoDaemonThread extends Thread

{

String s;

public DemoDaemonThread(String name){

s=name;

}

public void run()

{

// Checking whether the thread is Daemon or not

if(Thread.currentThread().isDaemon())

{

System.out.println(s + " is Daemon Thread");

}

else

{

System.out.println(s + " is User Thread");

}

}

public static void main(String[] args)

{

DemoDaemonThread thread1 = new DemoDaemonThread("thread1");

DemoDaemonThread thread2 = new DemoDaemonThread("thread2");

DemoDaemonThread thread3 = new DemoDaemonThread("thread3");

thread1.setDaemon(true);// set user thread1 to Daemon

thread1.start();// starting thread1

thread2.start();// starting thread2

thread3.setDaemon(true);// set user thread1 to Daemon

thread3.start();// starting thread3

}

}

Daemon Thread vs User Thread

learn more about the distinctions between Daemon and User threads:

Daemon Threads User Threads (Non-daemon)

- Low Priority threads - High priority threads

- The JVM does not wait for its execution - The JVM waits till the execution is finished.

to complete.

- Life is dependent on user threads - Life is independent

- Daemon threads are created by JVM - An application creates its own user threads.

- provides service to the user thread - Used for foreground tasks

which runs in the background

==================================================================================

**/// java.lang.Object:**

In Java, java.lang.Object is the root class of all classes. Every class in Java is a direct or indirect subclass of Object. It's located in the java.lang package, which is automatically imported into every Java program, so you don't need to explicitly import it.

The Object class defines several important methods that are available to all Java objects. Some of the most commonly used methods defined in Object are:

toString(): Returns a string representation of the object. By default, this method returns the class name followed by the "@" symbol and the hash code of the object.

equals(Object obj): Indicates whether some other object is "equal to" this one. The default implementation in Object compares object references.

hashCode(): Returns a hash code value for the object. The default implementation returns the internal memory address of the object in hexadecimal form.

getClass(): Returns the runtime class of this object. This method returns an instance of the Class class that provides information about the object's class.

notify(), notifyAll(), and wait(): These methods are used for inter-thread communication and synchronization.

///================================================================================

**Wrapper Classes:**

Wrapper classes in Java are used to represent primitive data types as objects. Each primitive data type (such as int, float, char, etc.) has a corresponding wrapper class in Java. The primary purpose of wrapper classes is to provide a way to treat primitive data types as objects. Additionally, wrapper classes provide utility methods for converting primitive data types to and from strings, as well as other helpful functionality.

Here are the wrapper classes for primitive data types in Java:

**Byte**: Represents a byte value.

**Short:** Represents a short value.

**Integer:** Represents an int value.

**Long:** Represents a long value.

**Float**: Represents a float value.

**Double**: Represents a double value.

**Character:** Represents a char value.

**Boolean:** Represents a boolean value.

Wrapper classes are typically used in scenarios where primitive data types cannot be used, such as in collections (like ArrayList or HashMap), generics, or when dealing with methods that expect objects rather than primitives. They are also useful when you need to perform operations that are only available for objects, like using methods from the Object class.

class DemoInteger

{

public static void main(String []args)

{

int x=15;

Integer iob=new Integer(x);

System.out.println("For Int No of Bytes: "+Integer.BYTES);

System.out.println("For Int No of Bits: "+Integer.SIZE);

System.out.println("Short Range: "+Short.MIN\_VALUE+" To "+Short.MAX\_VALUE);

System.out.println("Binary of 15: "+Integer.toBinaryString(15));

System.out.println("Octal of 15: "+Integer.toOctalString(15));

System.out.println("Hex of 15: "+Integer.toHexString(15));

}

}

///================================================================================

**java.util.Arrays:**

The java.util.Arrays class in Java provides various utility methods for working with arrays. It contains methods for sorting, searching, comparing, filling, and converting arrays. Below, I'll explain some of the commonly used methods in the Arrays class:

**Sorting Methods:**

sort(T[] a): Sorts the specified array of objects into ascending order, according to the natural ordering of its elements.

sort(T[] a, Comparator<? super T> c): Sorts the specified array of objects according to the order induced by the specified comparator.

**Searching Methods:**

binarySearch(T[] a, T key): Searches the specified array of objects for the specified value using the binary search algorithm.

binarySearch(T[] a, int fromIndex, int toIndex, T key): Searches a range of the specified array of objects for the specified value using the binary search algorithm.

**Comparison Methods:**

equals(T[] a, T[] a2): Returns true if the two specified arrays of objects are equal to one another.

deepEquals(Object[] a1, Object[] a2): Returns true if the two specified arrays are deeply equal to one another.

**Conversion Methods:**

asList(T... a): Returns a fixed-size list backed by the specified array.

toString(T[] a): Returns a string representation of the contents of the specified array.

**Filling Methods:**

fill(T[] a, T val): Assigns the specified value to each element of the specified array.

fill(T[] a, int fromIndex, int toIndex, T val): Assigns the specified value to each element of the specified array of objects within the specified range.

**Miscellaneous:**

copyOf(T[] original, int newLength): Copies the specified array, truncating or padding with nulls (if necessary) so the copy has the specified length.

copyOfRange(T[] original, int from, int to): Copies the specified range of the specified array into a new array.

import java.util.Arrays;

public class ArraysExample {

public static void main(String[] args) {

// Sorting

Integer[] arr = {5, 2, 8, 1, 9};

Arrays.sort(arr);

System.out.println("Sorted Array: " + Arrays.toString(arr));

// Searching

int index = Arrays.binarySearch(arr, 8);

System.out.println("Index of 8: " + index);

// Filling

Integer[] newArr = new Integer[5];

Arrays.fill(newArr, 10);

System.out.println("Filled Array: " + Arrays.toString(newArr));

// Copying

Integer[] copiedArr = Arrays.copyOf(arr, arr.length);

System.out.println("Copied Array: " + Arrays.toString(copiedArr));

// List conversion

Integer[] listArr = {1, 2, 3};

System.out.println("List from Array: " + Arrays.asList(listArr));

}

}

//----------------------------------------------------------------------------------------------------------------------------------

**// Using String, StringBuffer and StringBuilder**

**public final class String**

**extends Object**

**implements Serializable, Comparable<String>, CharSequence**

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable, they can be shared. For example:

String str = "abc";

is equivalent to:

char data[] = {'a', 'b', 'c'};

String str = new String(data);

Here are some more examples of how strings can be used:

System.out.println("abc");

String cde = "cde";

System.out.println("abc" + cde);

String c = "abc".substring(2,3);

String d = cde.substring(1, 2);

The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard version specified by the Character class.

//-----------------------------------------------------------------------------------------------------------------------------------

**public final class StringBuffer**

**extends Object**

**implements Serializable, CharSequence**

A thread-safe, mutable sequence of characters. A string buffer is like a String, but can be modified. At any point in time it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.

String buffers are safe for use by multiple threads. The methods are synchronized where necessary so that all the operations on any particular instance behave as if they occur in some serial order that is consistent with the order of the method calls made by each of the individual threads involved.

//----------------------------------------------------------------------------------------------------------------------------------

**public final class StringBuilder**

**extends Object**

**implements Serializable, CharSequence**

A mutable sequence of characters. This class provides an API compatible with StringBuffer, but with no guarantee of synchronization. This class is designed for use as a drop-in replacement for StringBuffer in places where the string buffer was being used by a single thread (as is generally the case). Where possible, it is recommended that this class be used in preference to StringBuffer as it will be faster under most implementations.

//================================================================================

**collection class hierarchy:** Collection classes in Java provide a way to group multiple elements into a single unit, allowing operations such as adding, removing, and retrieving elements. Java's Collection Framework, located in the java.util package, provides a rich set of interfaces and classes for working with collections. Let's explore some of the key collection classes and interfaces:

java.util.Collection

├── java.util.List

│ ├── java.util.ArrayList

│ ├── java.util.LinkedList

│ └── java.util.Vector

├── java.util.Queue

│ ├── java.util.Deque

│ │ ├── java.util.ArrayDeque

│ │ └── java.util.LinkedList

│ └── java.util.PriorityQueue

└── java.util.Set

├── java.util.HashSet

├── java.util.LinkedHashSet

└── java.util.TreeSet

**Interfaces:**

Collection: This is the root interface of the Java Collections Framework. It provides the basic methods for working with collections, such as add, remove, contains, size, etc.

List: This interface represents an ordered collection where elements can be inserted or accessed by their index. It allows duplicate elements.

Set: Set represents a collection that contains no duplicate elements. It doesn't maintain any specific order of elements.

Queue: Queue represents a collection designed for holding elements prior to processing. It follows the FIFO (First-In-First-Out) principle.

Deque: Deque is a double-ended queue. It allows insertion and removal of elements from both ends.

Map: This interface represents a collection of key-value pairs. It maps keys to values, where each key is unique.

Classes:

ArrayList: Implements the List interface using a dynamic array. It allows fast random access and is good for storing and accessing data sequentially.

LinkedList: Implements the List interface using a doubly-linked list. It provides efficient insertion and deletion operations, especially in the middle of the list.

HashSet: Implements the Set interface using a hash table. It provides constant-time performance for basic operations like add, remove, contains, etc., assuming the hash function disperses the elements properly.

LinkedHashSet: Extends HashSet to maintain a linked list of the entries in the set, in the order in which they were inserted.

TreeSet: Implements the Set interface using a self-balancing binary search tree (specifically, a Red-Black Tree). It provides sorted order of elements and allows efficient operations like add, remove, and contains.

HashMap: Implements the Map interface using a hash table. It provides constant-time performance for the basic operations like put, get, remove, and containsKey, assuming the hash function disperses the elements properly.

LinkedHashMap: Extends HashMap to maintain a doubly-linked list running through all of its entries. It provides predictable iteration order, which is the order in which the entries were inserted.

TreeMap: Implements the Map interface using a Red-Black tree. It provides sorted order of keys and efficient operations like put, get, remove, and containsKey.

import java.util.Vector;

public class VectorExample {

public static void main(String[] args) {

// Create a Vector

Vector<String> vector = new Vector<>();

// Adding elements

vector.add("Java");

vector.add("Python");

vector.add("C++");

// Displaying the Vector

System.out.println("Vector: " + vector);

// Accessing elements by index

String element = vector.get(0);

System.out.println("Element at index 0: " + element);

// Removing elements

vector.remove("Python");

System.out.println("Vector after removing 'Python': " + vector);

// Checking if an element exists

boolean containsCPlusPlus = vector.contains("C++");

System.out.println("Does Vector contain 'C++'? " + containsCPlusPlus);

// Getting the size of the Vector

int size = vector.size();

System.out.println("Size of Vector: " + size);

// Clearing the Vector

vector.clear();

System.out.println("Vector after clearing: " + vector);

}

}

//--------------------------------------------------------------------------------------------

import java.util.ArrayList;

public class ArrayListExample {

public static void main(String[] args) {

// Creating an ArrayList

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

// Adding elements

arrayList.add("Apple");

arrayList.add("Banana");

arrayList.add("Orange");

// Displaying the ArrayList

System.out.println("ArrayList: " + arrayList);

// Accessing elements by index

String element = arrayList.get(0);

System.out.println("Element at index 0: " + element);

// Removing elements

arrayList.remove("Banana");

System.out.println("ArrayList after removing 'Banana': " + arrayList);

// Checking if an element exists

boolean containsOrange = arrayList.contains("Orange");

System.out.println("Does ArrayList contain 'Orange'? " + containsOrange);

// Getting the size of the ArrayList

int size = arrayList.size();

System.out.println("Size of ArrayList: " + size);

// Clearing the ArrayList

arrayList.clear();

System.out.println("ArrayList after clearing: " + arrayList);

}

}

//----------------------------------------------------------------------------

import java.util.LinkedList;

public class LinkedListExample {

public static void main(String[] args) {

// Creating a LinkedList

LinkedList<String> linkedList = new LinkedList<>();

// Adding elements

linkedList.add("Apple");

linkedList.add("Banana");

linkedList.add("Orange");

// Displaying the LinkedList

System.out.println("LinkedList: " + linkedList);

// Adding elements at specific positions

linkedList.addFirst("Grapes");

linkedList.addLast("Pineapple");

System.out.println("LinkedList after adding 'Grapes' at first and 'Pineapple' at last: " + linkedList);

// Accessing elements by index

String element = linkedList.get(2);

System.out.println("Element at index 2: " + element);

// Removing elements

linkedList.remove("Banana");

System.out.println("LinkedList after removing 'Banana': " + linkedList);

// Checking if an element exists

boolean containsOrange = linkedList.contains("Orange");

System.out.println("Does LinkedList contain 'Orange'? " + containsOrange);

// Getting the size of the LinkedList

int size = linkedList.size();

System.out.println("Size of LinkedList: " + size);

// Clearing the LinkedList

linkedList.clear();

System.out.println("LinkedList after clearing: " + linkedList);

}

}

//-----------------------------------------------------------------------------------------------------

// Java IO

In Java, IO (Input/Output) refers to the process of reading data from an external source (input) or writing data to an external destination (output). The Java IO system provides a comprehensive set of classes and interfaces for performing various IO operations.

Java IO is divided into two main streams: Byte Streams and Character Streams.

Byte Streams: These are used for reading and writing binary data, such as images, audio files, etc. Byte Streams are based on the InputStream and OutputStream classes. Some commonly used byte stream classes include FileInputStream, FileOutputStream, BufferedInputStream, BufferedOutputStream, etc.

Character Streams: These are used for reading and writing text data, typically encoded using character encodings such as UTF-8 or ISO-8859-1. Character Streams are based on the Reader and Writer classes. Some commonly used character stream classes include FileReader, FileWriter, BufferedReader, BufferedWriter, etc.

example of reading from a text file using character streams

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class ReadFileExample {

public static void main(String[] args) {

try (BufferedReader br = new BufferedReader(new FileReader("example.txt"))) {

String line;

while ((line = br.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

example of reading from a text file using byte streams in Java

import java.io.FileInputStream;

import java.io.IOException;

public class ReadFileByteStreamExample {

public static void main(String[] args) {

try (FileInputStream fis = new FileInputStream("example.txt")) {

int data;

while ((data = fis.read()) != -1) {

System.out.print((char) data);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

//------------------------------------------------------------------------------------------------------------------------------------

//------------------------------------------------------------------------------------------------------------------------------------

// Java Database Connectivity

What is data?

Data refers to raw facts, figures, symbols, or observations that represent information. It can be in various forms, such as numbers, text, images, audio, or video. Data on its own lacks context and meaning but can be processed and analyzed to extract insights and make informed decisions. In today's digital age, data is generated constantly through various sources like sensors, devices, social media, and transactions. It serves as the foundation for analysis, research, and decision-making across numerous fields, including business, science, healthcare, and technology.

In Java programs, you can store data in various ways depending on your requirements. Here are some common options:

- Variables

- Arrays

- Objects

- Files

Why the files are used to store the data?

Files provide a means of storing data persistently beyond the lifetime of a program. Once data is written to a file, it remains stored until explicitly modified or deleted, even after the program terminates.

Files allow for the storage and manipulation of large datasets that may not fit entirely into memory. By storing data in files, programs can efficiently handle datasets that exceed the available memory capacity, avoiding memory-related issues such as crashes or performance degradation.

Files facilitate data sharing between different programs or instances of the same program. Data stored in files can be accessed by multiple programs, enabling interoperability and data exchange.

Once data is written to a file, it can be transferred and accessed on different systems

package mainpack;

import java.io.\*;

import java.util.Scanner;

class UserData implements Serializable {

private String name;

private int age;

public UserData(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public int getAge() {

return age;

}

}

public class ObjectFileMenu {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nMenu:");

System.out.println("1. Write object to file");

System.out.println("2. Read object from file");

System.out.println("3. Exit");

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

int choice = scanner.nextInt();

switch (choice) {

case 1:

writeObjectToFile();

break;

case 2:

readObjectFromFile();

break;

case 3:

System.out.println("Exiting program...");

System.exit(0);

default:

System.out.println("Invalid choice! Please enter a valid option.");

}

}

}

private static void writeObjectToFile() {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter user details:");

System.out.print("Name: ");

String name = scanner.nextLine();

System.out.print("Age: ");

int age = scanner.nextInt();

UserData user = new UserData(name, age);

String filePath = "e:\\userdata.txt";

try {

FileOutputStream fileOut = new FileOutputStream(filePath);

ObjectOutputStream objectOut = new ObjectOutputStream(fileOut);

objectOut.writeObject(user);

objectOut.close();

fileOut.close();

System.out.println("UserData object has been serialized and stored in " + filePath);

} catch (IOException e) {

e.printStackTrace();

}

}

private static void readObjectFromFile() {

String filePath = "e:\\userdata.txt";

try {

FileInputStream fileIn = new FileInputStream(filePath);

ObjectInputStream objectIn = new ObjectInputStream(fileIn);

UserData storedUser = (UserData) objectIn.readObject();

objectIn.close();

fileIn.close();

System.out.println("Deserialized UserData object:");

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

System.out.println("Age: " + storedUser.getAge());

} catch (IOException | ClassNotFoundException e) {

System.out.println("Error: File not found or unable to deserialize UserData object.");

}

}

}

A File Management system is a DBMS that allows access to single files or tables at a time. In a File System, data is directly stored in a set of files. It contains flat files that have no relation to other files.So for that databases are introduced.

what is database?

A database is an organized collection of structured data that is stored electronically in a computer system. It is designed to efficiently manage, retrieve, and manipulate large volumes of data according to predefined schemas or models. Databases are essential components of modern information systems and are used in various applications and industries.

Key characteristics of a database include:

Structured Format: Data in a database is organized in a structured format, typically using tables, rows, and columns, which allows for easy storage and retrieval.

Querying Capabilities: Databases provide query languages (e.g., SQL - Structured Query Language) that allow users to retrieve and manipulate data based on specific criteria.

Data Integrity: Databases enforce data integrity rules to ensure that data remains accurate, consistent, and reliable over time.

Concurrency Control: Databases support multiple concurrent users or applications accessing and modifying data simultaneously, while ensuring that transactions are executed reliably and without interference.

Scalability: Databases are designed to scale as data volumes and user loads increase. They can handle large amounts of data and support growth without sacrificing performance.

Security: Databases implement security measures to control access to data, protect sensitive information, and prevent unauthorized access or modifications.

Backup and Recovery: Databases provide mechanisms for backing up data regularly and recovering it in case of failures or disasters.

//+-----------------------------------------------------------------------------------------------------------------------------

Different Types of database

There are several types of databases, each designed to handle specific data models, use cases, and requirements. Some of the most common types include:

Relational Databases: Relational databases organize data into tables, with each table consisting of rows and columns. They use structured query language (SQL) for querying and managing data. Examples include MySQL, PostgreSQL, Oracle Database, SQL Server, and SQLite.

NoSQL Databases: NoSQL (Not Only SQL) databases are designed to handle large volumes of unstructured or semi-structured data. They offer flexible schemas and are often used for real-time analytics, content management, and distributed systems. Examples include MongoDB, Cassandra, Couchbase, Redis, and Amazon DynamoDB.

Document Databases: Document databases store data in flexible, JSON-like documents, allowing for nested structures and dynamic schemas. They are well-suited for content management, e-commerce, and applications with evolving data models. Examples include MongoDB, Couchbase, and RavenDB.

Key-Value Stores: Key-value stores store data as key-value pairs, where each key is unique and maps to a corresponding value. They are optimized for high-speed retrieval and are commonly used for caching, session management, and distributed systems. Examples include Redis, Amazon DynamoDB, and Riak.

Column-Family Stores: Column-family stores organize data into columns instead of rows, allowing for efficient storage and retrieval of large datasets. They are suitable for analytics, time-series data, and data warehousing. Examples include Apache Cassandra, HBase, and Google Bigtable.

Graph Databases: Graph databases store data in nodes, edges, and properties, representing relationships between entities in a graph-like structure. They excel at querying and analyzing complex relationships in social networks, recommendation engines, and network management systems. Examples include Neo4j, Amazon Neptune, and JanusGraph.

Time-Series Databases: Time-series databases are optimized for storing and analyzing time-stamped data, such as sensor readings, log files, and IoT data streams. They support efficient data aggregation, visualization, and trend analysis over time. Examples include InfluxDB, Prometheus, and TimescaleDB.

Spatial Databases: Spatial databases specialize in storing and querying spatial data, such as maps, geographic information systems (GIS), and location-based services. They provide features for geometric operations, spatial indexing, and spatial analysis. Examples include PostGIS, Oracle Spatial and Graph, and MongoDB with GeoJSON support.

//-----------------------------------------------------------------------------------------------------------------------------

What is DBMS?

Database Management Systems (DBMS) are software systems used to store, retrieve, and run queries on data. A DBMS serves as an interface between an end-user and a database, allowing users to create, read, update, and delete data in the database.

DBMS manage the data, the database engine, and the database schema, allowing for data to be manipulated or extracted by users and other programs. This helps provide data security, data integrity, concurrency, and uniform data administration procedures.

DBMS optimizes the organization of data by following a database schema design technique called normalization, which splits a large table into smaller tables when any of its attributes have redundancy in values. DBMS offer many benefits over traditional file systems, including flexibility and a more complex backup system.

What is a relational database management system?

A relational database management system (RDBMS) is a database that stores information in the form of tables called relations. These tables include numerous rows and columns, often called records and fields. Each record typically contains its own unique ID called a key. An RDBMS may include millions of rows. The columns often consist of one type of data, such as names and numbers, that contain attributes of the records. Most modern businesses, IT systems and other programs often use RDBMSs because they can use the data in relation to other stored datasets.

These databases can also manage complex data and queries, which typically makes them more capable than flat-file and hierarchical databases. A relational database management system can link information in multiple ways. One table record's characteristics may link to another record in a separate table. One record may also link to multiple records in another table. Multiple records may also link to other records in more than one additional table.

How RDBM systems work

An RDBMS uses the relational database model, which stores information in tables and often makes it easy to add new records and relate records to other records and tables. Each system usually includes varying numbers of tables, and each of these tables contains its own primary key. The primary key identifies each table. Each row within the table includes information for that individual entry, while the columns include information about the specific field. When a system creates a table, it reviews these limitations:

Primary keys: This requires unique information with no null values.

Foreign keys: This information links at least two tables. A foreign key resides in one table and references the primary key of another table.

Not null: This verifies that each record in a table contains no null values, such as empty cells.

Check: This verifies the uniqueness and precise conditions of each entry and column.

A system must also confirm the integrity of all data before creating the table. This typically includes numerous tests related to entity, domain, user-defined and referential integrity. These tests verify the uniqueness of all information and identify any duplicate data. Each test may also investigate the specific conditions of data, such as file format, and how the table adheres to user-defined conditions.

Types of relational database management systems

Here's a list of some common relational database management systems that many companies use to track their operations:

- Oracle

- MySQL

- MariaDB

- SQLite

- PostgreSQL

//-------------------------------------------------------------------------------------------------------------------

// What is JDBC?

Java database connectivity (JDBC) is the Java Software specification of a standard application programming interface (API) that allows Java programs to access database management systems. The JDBC API consists of a set of interfaces and classes written in the Java programming language.

Using these standard interfaces and classes, programmers can write applications that connect to databases, send queries written in structured query language (SQL), and process the results

Java Application JDBC Database

[ ] <--------> [ ] <----------> [ ]

Since JDBC is a standard specification, one Java program that uses the JDBC API can connect to any database management system (DBMS), as long as a driver exists for that particular DBMS.

What is a JDBC driver?

The JDBC API defines the Java™ interfaces and classes that programmers use to connect to databases and send queries. A JDBC driver implements these interfaces and classes for a particular DBMS vendor.

A Java program that uses the JDBC API loads the specified driver for a particular DBMS before it actually connects to a database. The JDBC DriverManager class then sends all JDBC API calls to the loaded driver.

The architecture of JDBC consists of:

Java Applications: It can be either a Java swing or a servlet that has to perform certain data-related tasks. To do so, it must be able to communicate with the data present in the database.

JDBC API: It contains various classes and interfaces that help us to execute SQL queries. We can use JDBC API to handle the database using the Java program and can perform the following activities:

•Connect to the database

•Execute queries and update statements to the database

•Retrieve the result received from the database.

Some of the highly popular JDBC API classes are:

•DriverManager class

•Clob class

•Blob class

•Types class

Similarly, some popular JDBC API interfaces are:

•Driver interface

•Connection interface

•Statement interface

•PreparedStatement

•CallableStatement interface

•ResultSet interface

•ResultSetMetaData interface

•DatabaseMetaData interface

•RowSet interface

JDBC DriverManager: It is a very crucial component of the JDBC. The main work of the DriverManager is to load a specific driver for a database as various databases may have a different driver. The driver manager ensures that the correct driver is chosen for the database we are interacting with and is returned to the application.

JDBC Drivers: It is one of the most crucial parts of the entire JDBC. A JDBC Driver is a software component that enables a Java application to interact with a database.

The important functionalities of the JDBC driver are:

•It handles the communications with the database server.

•Instead of directly creating a Driver object, we use the DriverManager object to deal with the drivers.

•It also abstracts the details associated with working with the Driver objects.

There are four types of JDBC drivers:

•JDBC-ODBC bridge plus ODBC driver, also called Type 1 driver: Translates JDBC API calls into Microsoft ODBC calls that are then passed to the ODBC driver. The ODBC binary code must be loaded on every client computer that uses this type of driver. ODBC is an acronym for Open Database Connectivity.

•Native-API, partly Java driver, also called Type 2 driver: Converts JDBC API calls into DBMS-specific client API calls. Like the bridge driver, this type of driver requires that some binary code is loaded on each client computer.

•JDBC-Net, pure-Java driver, also called Type 3 driver: Sends JDBC API calls to a middle-tier server that translates the calls into the DBMS-specific network protocol. The translated calls are then sent to a particular DBMS.

•Native-protocol, pure-Java driver, also called Type 4 driver: Converts JDBC API calls directly into the DBMS-specific network protocol without a middle tier. This driver allows the client applications to connect directly to the database server.

How to create a Java JDBC connection?

We have learned what a JDBC is, what it is used for, its components, driver types, and its relationship with relational databases. Now, we will be going through detailed steps through which we use JDBC to create a connection to a database in Java.

Here is the 7-step process to create a Java JDBC connection:

1. Import the packages: This includes uploading all the packages containing the JDBC classes, interfaces, and subclasses used during the database programming, using import java.sql.\* However, other classes can be imported if needed in the program.

2. Register the drivers: Before connecting to the database, we’ll need to load or register the drivers once per database. This is done to create a communication channel with the database. Loading a driver can be done in two ways:

•Class.forName()

•DriverManager.registerDriver()

3. Establish a connection: For the next step, the getConnection() method is used to create a connection object that will correspond to a physical connection with the database. To get the getConnection() to access the database, the three parameters are a username, string data type URL, and a password.

Two methods can be used to achieve this:

•getConnection(URL, username, password): This uses three parameters URL, a password, and a username

•getConnection(URL): This has only one parameter - URL. The URL has both a username and password.

There are several JDBC connection strings for different relational databases and some are listed below:

oIBM DB2 database: jdbc:db2://HOSTNAME:PORT/DATABASE\_NAME

oOracle database: jdbc:oracle:thin:@HOST\_NAME:PORT:SERVICE\_NAME

oMy SQL database: jdbc:mysql://HOST\_NAME:PORT/DATABASE\_NAME

4. Create a statement: The statement can now be created to perform the SQL query when the connection has been established. There are three statements from the createStatement method of the connection class to establish the query. These statements are

•Statement: This is used to create simple SQL statements with no parameter.

An example is: Statement statemnt1 = conn.createStatement();.

This statement returns the ResultSet object.

•PreparedStatement: This extends the Statement interface. It improves the application's performance because it has more features and compiles the query only once. It is used for precompiled SQL statements that have parameters.

•CallableStatement: CallableStatements also extends the PreparedStatement interface. It is used for SQL statements with parameters that invoke procedures or functions in the database. It is simply created by calling the prepare all method of the connection object.

5. Execute the query: This uses a type statement object to build and submit SQL statements to a database. It has four distinct methods:

•ResultSet executeQuery(String sql)

•executeUpdate(String sql)

•execute(String sql)

•executeBatch()

6. Retrieve results: When queries are executed using the executeQuery() method, it produces results stored in the ResultSet object. The ResultSet object is then used to access the retrieved data from the database.

7. Close the connections: The JDBC connection can now be closed after all is done. The resource has to be closed to avoid running out of connections. It can be done automatically using ‘conn.close();’. But for versions of Java 7 and above, it can be closed using a try-catch block.

//----------------------------------------------------------------------------------------------------------------

Let’s check it with the example Using MySQL:

- Lets create the database

- Install MySql and open it using cmd as (>mysql -u roor -p) use password to login

- use(>show databases;)to see list of databases.

- Create database using command (> create database demodb;) and again see the database list.

- now write a java program for the database connection

//----------------------------------------------------------------------------------------------------------------

package mainpack;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class Main {

public static void main(String[] args) {

// JDBC URL for MySQL database named demodb

String jdbcUrl = "jdbc:mysql://localhost:3306/demodb";

String username = "root"; // Replace with your MySQL username

String password = "archer@12345678"; // Replace with your MySQL password

// Attempt to establish a connection to the database

try {

// Load the MySQL JDBC driver

Class.forName("com.mysql.cj.jdbc.Driver");

// Create a connection to the database

Connection connection = DriverManager.getConnection(jdbcUrl, username, password);

// Check if the connection is successful

if (connection != null) {

System.out.println("Connection to demodb database successful!");

// You can perform database operations here

// For example: execute queries, update data, etc.

} else {

System.out.println("Failed to connect to demodb database!");

}

// Close the connection

connection.close();

} catch (ClassNotFoundException e) {

System.out.println("MySQL JDBC driver not found!");

e.printStackTrace();

} catch (SQLException e) {

System.out.println("Connection to demodb database failed!");

e.printStackTrace();

}

}

}

if you face this exception, then you need to add mysql connector jar

java.lang.ClassNotFoundException: com.mysql.cj.jdbc.Driver

(download connector jar from https://downloads.mysql.com/archives/c-j/

then rigth click on project inr eclipse -> properties -> java built path -> Libraries -> Add External JAR -> Apply -> Apply and close)

and again run the code.

//----------------------------------------------------------------------------------------------------------

- Now create the new database (mysql> create database traveldb;)

- check it is created or not (mysql> show databases;)

- select that database (mysql> use traveldb;)

- Execute following query to create the table Travel

CREATE TABLE Travel(

id INT AUTO\_INCREMENT PRIMARY KEY,

destination VARCHAR(255),

departure\_date DATE,

return\_date DATE,

traveler\_name VARCHAR(100),

traveler\_age INT,

booking\_reference VARCHAR(50)

);

- Then write a java program to insert and display the data

package mainpack;

import java.sql.\*;

import java.util.Scanner;

public class TravelManager {

static final String JDBC\_DRIVER = "com.mysql.cj.jdbc.Driver";

static final String DB\_URL = "jdbc:mysql://localhost:3306/traveldb";

static final String USERNAME = "root";

static final String PASSWORD = "archer@12345678";

public static void main(String[] args) {

Connection conn = null;

Statement stmt = null;

Scanner sc=new Scanner(System.in);

try {

Class.forName(JDBC\_DRIVER);

conn = DriverManager.getConnection(DB\_URL, USERNAME, PASSWORD);

stmt = conn.createStatement();

int choice;

do {

System.out.println("1. Insert Travel Record");

System.out.println("2. Display Travel Records");

System.out.println("3. Exit");

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

choice = sc.nextInt();

switch (choice) {

case 1:

insertRecord(stmt);

break;

case 2:

displayRecords(stmt);

break;

case 3:

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

break;

default:

System.out.println("Invalid choice! Please try again.");

}

} while (choice != 3);

stmt.close();

conn.close();

} catch (Exception e) {

e.printStackTrace();

}

}

static void insertRecord(Statement stmt) throws SQLException {

Scanner sc=new Scanner(System.in);

System.out.print("Enter booking reference: ");

String bookingReference = sc.nextLine();

System.out.print("Enter destination: ");

String destination = sc.nextLine();

System.out.print("Enter departure date (YYYY-MM-DD): ");

String departureDate = sc.nextLine();

System.out.print("Enter return date (YYYY-MM-DD): ");

String returnDate = sc.nextLine();

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

String travelerName = sc.nextLine();

System.out.print("Enter traveler age: ");

int travelerAge = sc.nextInt();

String sql = "INSERT INTO Travel (destination, departure\_date, return\_date, traveler\_name, traveler\_age, booking\_reference) " +

"VALUES ('" + destination + "', '" + departureDate + "', '" + returnDate + "', '" + travelerName + "', " + travelerAge + ", '" + bookingReference + "')";

int rowsAffected = stmt.executeUpdate(sql);

if (rowsAffected > 0) {

System.out.println("Travel record inserted successfully!");

} else {

System.out.println("Failed to insert travel record.");

}

}

static void displayRecords(Statement stmt) throws SQLException {

String sql = "SELECT \* FROM Travel";

ResultSet rs = stmt.executeQuery(sql);

System.out.println("Travel Records:");

while (rs.next()) {

int id = rs.getInt("id");

String destination = rs.getString("destination");

Date departureDate = rs.getDate("departure\_date");

Date returnDate = rs.getDate("return\_date");

String travelerName = rs.getString("traveler\_name");

int travelerAge = rs.getInt("traveler\_age");

String bookingReference = rs.getString("booking\_reference");

System.out.println("ID: " + id);

System.out.println("Destination: " + destination);

System.out.println("Departure Date: " + departureDate);

System.out.println("Return Date: " + returnDate);

System.out.println("Traveler Name: " + travelerName);

System.out.println("Traveler Age: " + travelerAge);

System.out.println("Booking Reference: " + bookingReference);

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

}

rs.close();

}

}

//---------------------------------------------------------------------------------------------------------------------------------------------------------------------

/// Update and delete records

package mainpack;

import java.sql.\*;

import java.util.Scanner;

public class Main {

static final String JDBC\_DRIVER = "com.mysql.cj.jdbc.Driver";

static final String DB\_URL = "jdbc:mysql://localhost:3306/traveldb";

static final String USERNAME = "root";

static final String PASSWORD = "archer@12345678";

public static void main(String[] args) {

Connection conn = null;

Statement stmt = null;

Scanner sc=new Scanner(System.in);

try {

Class.forName(JDBC\_DRIVER);

conn = DriverManager.getConnection(DB\_URL, USERNAME, PASSWORD);

stmt = conn.createStatement();

int choice;

do {

System.out.println("1. Insert Travel Record");

System.out.println("2. Display Travel Records");

System.out.println("3. Modify Travel Record");

System.out.println("4. Delete Travel Record");

System.out.println("5. Exit");

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

choice = sc.nextInt();

switch (choice) {

case 1:

insertRecord(stmt);

break;

case 2:

displayRecords(stmt);

break;

case 3:

modifyRecord(stmt);

break;

case 4:

deleteRecord(stmt);

break;

case 5:

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

break;

default:

System.out.println("Invalid choice! Please try again.");

}

} while (choice != 5);

stmt.close();

conn.close();

} catch (Exception e) {

e.printStackTrace();

}

}

static void insertRecord(Statement stmt) throws SQLException {

Scanner sc=new Scanner(System.in);

System.out.print("Enter booking reference: ");

String bookingReference = sc.nextLine();

System.out.print("Enter destination: ");

String destination = sc.nextLine();

System.out.print("Enter departure date (YYYY-MM-DD): ");

String departureDate = sc.nextLine();

System.out.print("Enter return date (YYYY-MM-DD): ");

String returnDate = sc.nextLine();

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

String travelerName = sc.nextLine();

System.out.print("Enter traveler age: ");

int travelerAge = sc.nextInt();

String sql = "INSERT INTO Travel (destination, departure\_date, return\_date, traveler\_name, traveler\_age, booking\_reference) " +

"VALUES ('" + destination + "', '" + departureDate + "', '" + returnDate + "', '" + travelerName + "', " + travelerAge + ", '" + bookingReference + "')";

int rowsAffected = stmt.executeUpdate(sql);

if (rowsAffected > 0) {

System.out.println("Travel record inserted successfully!");

} else {

System.out.println("Failed to insert travel record.");

}

}

static void displayRecords(Statement stmt) throws SQLException {

String sql = "SELECT \* FROM Travel";

ResultSet rs = stmt.executeQuery(sql);

System.out.println("Travel Records:");

while (rs.next()) {

int id = rs.getInt("id");

String destination = rs.getString("destination");

Date departureDate = rs.getDate("departure\_date");

Date returnDate = rs.getDate("return\_date");

String travelerName = rs.getString("traveler\_name");

int travelerAge = rs.getInt("traveler\_age");

String bookingReference = rs.getString("booking\_reference");

System.out.println("ID: " + id);

System.out.println("Destination: " + destination);

System.out.println("Departure Date: " + departureDate);

System.out.println("Return Date: " + returnDate);

System.out.println("Traveler Name: " + travelerName);

System.out.println("Traveler Age: " + travelerAge);

System.out.println("Booking Reference: " + bookingReference);

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

}

rs.close();

}

static void modifyRecord(Statement stmt) throws SQLException {

Scanner sc=new Scanner(System.in);

System.out.print("Enter ID of the record to modify: ");

int id = sc.nextInt();

System.out.print("Enter new booking reference: ");

String bookingReference = sc.nextLine();

bookingReference = sc.nextLine();

System.out.print("Enter new destination: ");

String destination = sc.nextLine();

System.out.print("Enter new departure date (YYYY-MM-DD): ");

String departureDate = sc.nextLine();

System.out.print("Enter new return date (YYYY-MM-DD): ");

String returnDate = sc.nextLine();

System.out.print("Enter new traveler name: ");

String travelerName = sc.nextLine();

System.out.print("Enter new traveler age: ");

int travelerAge = sc.nextInt();

String sql = "UPDATE Travel SET destination = '" + destination + "', departure\_date = '" + departureDate + "', return\_date = '" + returnDate +

"', traveler\_name = '" + travelerName + "', traveler\_age = " + travelerAge + ", booking\_reference = '" + bookingReference +

"' WHERE id = " + id;

int rowsAffected = stmt.executeUpdate(sql);

if (rowsAffected > 0) {

System.out.println("Travel record with ID " + id + " modified successfully!");

} else {

System.out.println("Failed to modify travel record with ID " + id + ".");

}

}

static void deleteRecord(Statement stmt) throws SQLException {

Scanner sc=new Scanner(System.in);

System.out.print("Enter ID of the record to delete: ");

int id = sc.nextInt();

String sql = "DELETE FROM Travel WHERE id = " + id;

int rowsAffected = stmt.executeUpdate(sql);

if (rowsAffected > 0) {

System.out.println("Travel record with ID " + id + " deleted successfully!");

} else {

System.out.println("Failed to delete travel record with ID " + id + ". Record not found.");

}

}

}