07-OCT-24

**History of Java**

Name of the Language : Java

First Name of Java : OAK (Tree Name) 1991

Release Date : 23rd JAN 1996

Developed By : James Gosling and his friends

Project Name : Green Project

Java : Island, Indonesia

Official Symbol : Coffee CUP.

**What Is A Function?**

A function is a self defined block which is used to perform some calculation, printing the data and so on.

***Example***

public void sum(int x, int y)

{

}

**A function is divided into two types:**

***1) Predefined OR Built-in function***

***2) User-defined OR Custom function***

**Predefined OR Built-in function**

A function which is developed by language creator itself called

predefined function.

**User-defined OR Custom Function**

A function which is written by user for its own requirement and

specification is called User-defined function.

**Advantages of Function:**

**1) Modularity :** Dividing the bigger task into number of smaller task.

**2) Easy to understand:** Once the task is divided into number of independent modules then it is easy to understand the entire module.

**3) Reusability :** We can reuse a particular module for 'n' number of times.

Note: In java, we always reuse our java classes.

**4) Easy debugging :** Each module is isolated from another module so the debugging is easy because we can debug only one module where we have syntax or semantics error.

**Why we pass parameter to a function?**

We should pass parameter to a function for getting more information regarding the function.

If We don't pass parameter then the Information are not complete, It is partial information.

***Example***

public void deposit(double amount)

{

}

public void doSum(int x, int y)

{

}

public void sleep(int hours)

{

}

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

**Why functions are called Method in java?**

In C++ language, there is a facility to write a function inside the class as well outside of the class by using scope resolution operator (::) but in java we can write a function inside the class only, we can't define a function outside of the class, that is reason functions are called Method in java. [08-OCT-24]

09-10-2024

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

**What is platform independency in java?**

C and C++ programs are platform dependent programs that means the .exe file created on one machine will not be executed on the another machine if the system configuration is different.

That is the reason C and C++ programs are not suitable for website development.

***The role of java compiler***

1) Syntax verification.

2) Verify the compatibility issues (L.H.S = R.H.S)

3) Will Convert Source code into byte code.

Java is a platform independent language. Whenever we write a java program, the extension of java program must be **.java**

Now this **.java** file we submit to java compiler (javac) for compilation process. After successful compilation the compiler will generate a very special machine code file i.e. **.class** file (also known as bytecode). Now this **.class** file we submit to JVM for execution purpose.

The role of JVM is to load and execute the .class file. Here JVM plays a major role because It converts the .class file into appropriate machine code instruction (Operating System format) so java becomes platform independent language and it is highly suitable for website development. [09-OCT-24]

Note:- We have different JVM for different Operating System that means JVM is platform dependent technology where as Java is platform Independent technology.

JVM internally contains an interpreter so it executes the code line by line. It is written in 'C' language hance platform dependent.

Note: All the browsers internally contain JVM are known as JEB (Java Enabled Browsers) browser.

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

\*\***What is difference between bit code and byte code:**

Bit code is directly understood by Operating System but on the other hand byte code is understood by JVM, JVM is going to convert this byte code into appropriate machine understandable format.

[09-OCT-24]

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

14-10-2024

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

\*\***What is the difference between JDK, JRE, JVM and JIT compiler?**

Paint Diagram [14-OCT]

**JDK**

It stands for *Java Development Kit*. It contains JRE and JDK tools. It is a developer version that means by using JDK we can develop and execute java programs.

In order to develop and execute it supports various JDK tools which are as follows:

a) javac : Java compiler, responsible for compilation.

b) java : Java launcher, responsible for executing java program.

c) jdb : Java debugger, for debugging purpose

d) jconsole : Java Console, to display the output in the console.

e) javap : Java Profiler, to get the details of a class

f) javadoc : Java documention, for Generating java documentation.

**JRE**

It stands for *Java Runtime Environment.* It contains JVM and class libraries. It is a client version that means by using JRE we can only execute our java programs (We can't develop). From java 11 version JRE folder is removed so now, from java 11v we can execute a java program without compilation by using following commnd:

**java FileName.java** [This is the command to directly execute our java code]

**JVM**

JVM stands for *Java Virtual Machine*. The main purpose of JVM to load and execute the .class file. JVM takes .class file and convert the .class file into Operating System understandable format.

JVM is platform dependent, it provides various features like Class loading, class verification, allocating the memory for static data member, Garbage collection, Security Manager and so on.

**What is the difference Compiler and Interpreter**

Paint Diagram [14-OCT]

What is JIT Compiler: (Just-in-Time)

As we know, our interpreter is slow in nature so to boost up the java execution, we have JIT (Just In time) compiler support.

It holds the repeated code instruction and native code instruction, It will directly provide these two instruction at time of line by line execution so our interpreter executes the code in more efficient way hance the overall execution becomes very fast.

**How many data types in java?**

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

Paint Diagram [14-OCT]

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

15-10-2024

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

**What is the difference between statically typed and dynamically typed language?**

**Statically typed language**

The languages where data type is compulsory before initialization of a variable are called Statically typed language.

In these languages, once we define the type of the variable then it will hold same kind of value till the end of the program.

*Example of statically typed languages:*

C, C++, JSE, C# and so on

**Dynamically typed language**

The languages where type is not compulsory, It is optional to initialize the variable are called Dynamically Typed Language.

In these languages we can provide all different kinds of values to the variable during the execution of the program.

*Example of dynamically typed languages:*

Visual Basic, Javascript, Python and so on

**What is comment in java?**

Comments are used to enhance the readability of the code. It is ignored by the Compiler.

In java, we have 3 types of comments

***1) Single line Comment***

//

***2) Multiline Commment***

/\*

\*/

***3) Documentation comment***

/\*\*

Name of the Project : Online Shopping

Number of Modules : 70 Modules

Project Date : 12th March 2024

Last Modification : 9th Oct 2024

Author : James Goling and his friends

\*/

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

// WAP in java to display welcome message:

public class Welcome

{

public static void main(String[] args)

{

System.out.println("Hello Batch 40!!!!");

}

}

**Description of main() method:**

**Public :** It is an access modifier in java which defined the accessibility level of main method. Our main method must be declared as public otherwise JVM can't access our main method so the execution of the program will not be started.

If we don't declare our main method with public access modifier then code will compile but It will not be executed by JVM.

**static :** As of now, we have 2 types of methods in java :

1) static method (OBJECT IS NOT REQUIRED)

2) non static method (OBJECT IS REQUIRED)

static method: If any method declared with static keyword then it is called static method. In order to call and execute static methods, Object is not required.

**Case 1:** If a static method declared in the same class where main method is available then we can directly call the static method from main method as shown in the Program.

public class StaticTest

{

public static void main(String[] args)

{

greet();

}

public static void greet()

{

System.out.println("Hello Everyone!!!");

}

}

**Case 2:** If a static method is available in another class (Where main method is not available) then to call the static method CLASSname is required as shown in the program.

class Welcome

{

public static void greet()

{

System.out.println("Hello Batch 40");

}

}

public class StaticDemo

{

public static void main(String[] args)

{

Welcome.greet();

}

}

Our main method must be declared as static so, JVM can invoke main method with the help of class name.

If we don't declare our main method as static then code will compile (It is a non static method) but it will not be executed by JVM.

Note : There is no any syntax rule given by java compiler that method must be public and static.

**Void :** It is a keyword in java. We should write void before the name of the method so that particular method will not return any kind of value.

If we put the return type of the method as void then there is no communication between one module to another module. [It is one way communication]

If we don't write void or any other kind of return type before the main method then code WILL NOT compile because java compiler has provided syntax rule that every method must contain return type.

Note: If main method is not declared with void then program will NOT BE executed.

Note: Without return type we can't define a method in java [Syntax Rule]

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

16-10-2024

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

**main() method**

It is a user-defined method because a user is responsible to define some logic inside the main method.

main() method is very important method because every program execution will start from main() method only, as well as the execution of the program ends with main() method only.

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

*Q. Can we write multiple method with same name?*

Yes, We can write multiple methods with same name but parameter must be different otherwise code will not compile.

Note :- We can also write multiple main methods with different parameter but JVM will always execute the main method which takes String [] args (String array) as a parameter as shown in the program below.

public class Welcome

{

public static void main(String[] args)

{

System.out.println("Hello Batch 40!!!!");

main("NIT");

}

public static void main(String args)

{

System.out.println(args);

}

}

Output : Hello Batch 40

NIT

**String [] args**

String is a predefined class in java available in java.lang package (just like header file) and args is an array variable of type String so, it can hold multiple values.

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

*Q. Why the main method of java accepts String array as a parameter?*

String is a collection of alpha-numeric character so it can accept all different kind of values. Java software people has provided String array as a parameter so it can ACCEPT MULTIPLE VALUES OF DIFFERENT TYPE, that means providing more wider scope to accept hetrogeneous types of values.

**System.out.println()**

It is an output statement in java, By using this statement we can print different types of values on the console.

In this statement System is a predefined class available in java.lang pacakage, out is a reference variable of type PrintStream class available in java.io package and println() is a predefined method available in PrintStream class.

Note: Actually It is HAS-A relation concept, System class has PrintStream class as shown below.

class System

{

PrintStream out; //HAS-A Relation

}

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

*//WAP in java to add two numbers*

public class Addition

{

public static void main(String[] args)

{

int x = 100;

int y = 200;

int z = x + y;

System.out.println("The Sum is :"+z);

}

}

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

*//WAP to add two numbers without 3rd variable:*

public class AdditionWithout3rdVariable

{

public static void main(String[] args)

{

int x = 100;

int y = 200;

System.out.println("Sum is :"+x+y); //100200

System.out.println(+x+y); //300

System.out.println(""+x+y); //100200

System.out.println("Sum is :"+(x+y));

}

}

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

public class *IQ*

{

public static void main(String[] args)

{

String str = 25 + 25 + "NIT" + 90 + 89;

System.out.println(str); //50NIT9089

}

}

**Command Line Argument**

In java, Whenever we pass any argument to the main method then it is called Command Line Argument.

public static void main(String [] args) *//Command Line Argument*

{

}

By using command line argument, we can pass some value at runtime.

Advantage of command line argument is, single time compilation and number of time execution with different value.

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

*//Programs on Command Line Argument :*

public class Command

{

public static void main(String[] args)

{

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

}

}

cmd :

javac Command.java

java Command Virat Rohit Rahul

Output : Virat

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

public class CommandWithDifferentValues

{

public static void main(String[] x)

{

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

}

}

javac CommandWithDifferentValues.java

java CommandWithDifferentValues 78 9.0 Ravi false

Output: 78

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

*//WAP to print first and last name using command line argument :*

public class FullNameByCommand

{

public static void main(String[] args)

{

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

}

}

javac FullNameByCommand.java

java FullNameByCommand Virat Kohli [Output is : Virat]

java FullNameByCommand "Virat Kohli" [Output is : Virat Kohli]

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

public class Command

{

public static void main(String[] args)

{

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

}

}

javac Command.java

java Command

Output Exception: java.lang.ArrayIndexOutOfBoundsException

Note : This program is expecting at-least one value from the command line argument at runtime, if we don't provide single value then we will get an execption i.e. java.lang.ArrayIndexOutOfBoundsException

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

17-10-2024

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

//How to find out the length of an array variable?

Array is an object in java, array reference variable has provided a predefined property or variable called "length" through which we can get the length of the array.

ArrayLength.java

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

public class ArrayLength

{

public static void main(String[] args)

{

int []arr = {10,20,30,40,50};

System.out.println("Length of the array is :"+arr.length);

}

}

ArrayLengthByCommand.java

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

public class ArrayLengthByCommand

{

public static void main(String[] args)

{

System.out.println("Length of array is :"+args.length);

}

}

javac ArrayLengthByCommand.java

java ArrayLengthByCommand

Length of array is : 0

java ArrayLengthByCommand Lalit

Length of array is : 1

java ArrayLengthByCommand 56 90 89

Length of array is : 3

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

*//WAP in java to add two numbers using command line Argument :*

public class CommandAddition

{

public static void main(String[] args)

{

System.out.println(args[0] + args[1]);

}

}

javac CommandAddition.java

java CommandAddition 100 200

Output: 100200

Here '+' operator will work as String concatenation operator because args is String type variable.

***How to convert String into integer***

In order to convert a String into integer value, java software people has provided a predefined class called Integer, this class contains a static method called parseInt(String x) which accepts String as a parameter and convert this String into int type, that is the reason the return type of parseInt(String x) method is int type.

public class Integer

{

public static int parseInt(String x)

{

return int value after conversion;

}

}

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

public class AdditionByCommand

{

public static void main(String[] args)

{

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

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

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

}

}

javac AdditionByCommand.java

java AdditionByCommand 100 200

Output:

Sum is : 300

**What is Eclipse IDE**

IDE stands for "***Integrated Development Environment***". By using eclipse IDE, we can develop, compile and execute our java program in a single window.

The main purpose of Eclipse IDE to reduce the development time, once the development time will be reduced then automatically the cost of the project will be reduced.

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

**How to create a Project in Eclipse IDE**

File -> new -> Project OR Java Project -> Provide the name for

the project (Batch 40) -> Finish

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

**What is a Package in java**

A package is nothing but folder in windows. It is used to arrange the classes and interfaces into a particular group.

If we arrange our java classes into a particular group by using pacakges (folders) then we will get the following two advantages:

1) Fast searching is possible.

2) Name can be reusable.

Program that describes a package is folder in windows

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

***A package is a keyword in java and it must be the first statement of any java program.***

package arithmetic;

public class Addition

{

}

*Command for compilation of the classes which contains Package statement.*

[javac -d . FileName.java ]

javac -d . Addition.java

It will compile Addition.java, Addition.java contains arithmetic package, one package i.e folder called arithmetic will be created and automatically Addition.class file will be placed inside the package or folder called arithmetic.

**Types of Packages**

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

**1) Predefined OR Built-in package :** The packages which are created by java software people for arranging the programs are called predefined package.

**Example :** java.lang, java.util, java.io, java.sql, java.net and so on

**2) Userdefined Package OR Custom package :** The packages which are created by user for arranging the user-defined programs are called user-defined package.

**Example :**

basic;

com.ravi.basic;

com.tcs.online\_shopping;

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

18-10-2024

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

**Command Line Argument program by using Eclise IDE**

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

WAP to find out the square of the number by using Command Line Argument.

package com.nit.command\_line\_argument;

public class FindSquare

{

public static void main(String[] args)

{

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

System.out.println("Square of "+num+" is :"+(num\*num));

}

}

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

Steps to execute the command Line Argument Program using Eclipse IDE

Right click on the program -> Run As -> Run configuration -> Check your main class name -> select argument tab -> pass the appropriate value -> Run

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

*//WAP to find out the area of rectangle*

package com.nit.command\_line\_argument;

public class AreaOfRectangle {

public static void main(String[] args)

{

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

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

int areaOfRect = length \* breadth;

System.out.println("Area of Rectangle is :"+areaOfRect);

}

}

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

*How to convert String value into float and double*

1) String to float : float x = Float.parseFloat(String str);

2) String to double : double y = Double.parseDouble(String str);

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

*//WAP to find out the area of Circle by using command Line Arg*

package com.nit.command\_line\_argument;

public class AreaOfCircle {

public static void main(String[] args)

{

final double PI = 3.14;

double radius = Double.parseDouble(args[0]);

double areaOfCircle = PI \* radius \* radius;

System.out.println("Area of Circle is :"+areaOfCircle);

}

}

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

*//WAP in java to pass some value from command line argument based on the following criteria.*

If the array length is 0 : It should print length is 0

If the array length is 1 : It should find the cube of the number

if the array length is 2 : It should print sum of the number

package com.nit.command\_line\_argument;

public class ArrayLengthCalculation {

public static void main(String[] args)

{

if(args.length == 0)

{

System.err.println("Array Length is 0");

}

else if(args.length == 1)

{

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

System.out.println("Cube of "+num+" is :"+(num\*num\*num));

}

else if(args.length == 2)

{

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

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

int sum = x + y;

System.out.println("Sum is :"+sum);

}

}

}

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

*//Program that describes how Integer.parseInt(String x) works internally ?*

package com.nit.command\_line\_argument;

class Calculate

{

public static int doSum(int x, int y)

{

return (x+y);

}

}

public class CalculateDemo

{

public static void main(String[] args)

{

int result = Calculate.doSum(12, 24);

System.out.println("Sum is :"+result);

}

}

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

String class has provided a predefined non static method called charAt(int indexPosition) through which we can retrieve a character from the given string based on the index position.

*//Program to retrieve the character from the String*

package com.nit.command\_line\_argument;

public class CharAtDemo {

public static void main(String[] args)

{

String str = "india";

char firstChar = str.charAt(0);

System.out.println(firstChar);

}

}

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

*//WAP to read a character [M/F] from command line argument*

package com.nit.command\_line\_argument;

public class ReadCharacter {

public static void main(String[] args)

{

char gender = args[0].charAt(0);

System.out.println("Your Gender is :"+gender);

}

}

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

**Naming convention in java**

Naming convention provides two important characteristics :

*a) Standard Code (Industry accepted code)*

*b) Readability of the code will enhance.*

***1) How to write a class in java***

While writing a class in java, we should follow pascal naming convention, According to this each word first character must be capital and it should not contain any space. In java a class represents noun.

*Example:* ThisIsExampleOfClass

System

String

Integer

CommandAddition

ArrayIndexOutOfBoundsException

DataInputStream

***2) How to write a method in java***

While writing a method in java we should follow camel case naming convention, According to this naming convention first world will be in small and 2nd word onwards, each word first character must be capital. In java a method represents verb.

*Example:* thisIsExampleOfMethod()

read()

readLine()

parseInt()

charAt()

toUpperCase()

***3) How to write a field/variable in java***

While writing a variable we should follow camel case naming convention but unlike method variable does not have() symbol.

*Example:* rollNumber

customerName

customerBill

studentName

playerName

***4) How to write a final and static variable***

While writing the final and static variable we should follow snake\_case naming convention.

*Example:*

Integer.MIN\_VALUE [MIN\_VALUE is final and static variable]

Integer.MAX\_VALUE [MAX\_VALUE is final and static variable]

***5) How to write a package***

A package must be written in lower case only. Generally it is reverse of company name.

*Example:*

com.nit.basic

com.tcs.introduction

com.wipro.shopping

**Tokens in java**

Token is the smallest unit of the program which is identified by the compiler.

Without token we can't complete statement or expression in java.

**Token is divided into 5 types in java**

1) Keyword

2) Identifier

3) Literal

4) Punctuators (Seperators)

5) Operator

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

19-10-2024

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

**keyword**

A keyword is a predefined word whose meaning is already defined by the compiler.

In java all the keywords must be in lowercase only.

A keyword we can't use as a name of the variable, name of the class or name of the method.

true, false and null look like keywords but actually they are literals.

As of now, we have 67 keywords in java.

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

**Identifiers**

A name in java program by default considered as identifiers.

Assigned to variable, method, classes to uniquely identify them.

We can't use keyword as an identifier.

*Example:*

class Fan

{

int coil;

void switchOn()

{

}

}

*Here Fan(Name of the class), coil (Name of the variable) and switchOn(Name of the Method) are identifiers.*

Rules for defining an identifier

1) Can consist of uppercase(A-Z), lowercase(a-z), digits(0-9), $ sign, and underscore (\_)

2) Begins with letter, $, and \_

3) It is case sensitive

4) Cannot be a keyword

5) No limitation of length

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

**Literals**

Any constant which we are assiging to variable is called Literal.

**In java we have 5 types of Literals :**

1) Integral Literal

2) Floating Point Literal

3) Boolean Literal

4) Character Literal

5) String Literal

Note : null is also a literal

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

***Integral Literal***

If any numeric literal does not contain any decimal OR fraction then it is called Integral Literal.

Example : 12, 90, 56, 34

***In integral literal we have 4 data types:***

a) byte (8 bits)

b) short (16 bits)

c) int (32 bits)

d) long (64 bits)

***An integral literal we can represent in four different forms :***

a) Decimal Literal (Base is 10)

b) Octal Literal (Base is 8)

c) Hexadecimal Literal (Base is 16 [0-9 and a - f])

d) Binary Literal (Base 2) [Available from JDK 1.7v]

*Decimal Literal*

By default our numeric literals are decimal literal. Here base is 10 so, It accepts 10 digits i.e. from 0-9.

*Example:*

int x = 20;

int y = 123;

int z = 234;

*Octal Literal*

If any Integer literal starts with 0 (Zero) then it will become octal literal. Here base is 8 so it will accept 8 digits i.e 0 to 7.

*Example:*

int a = 018; //Invalid becuase it contains digit 8 which is out of range

int b = 0777; //Valid

int c = 0123; //Valid

*Hexadecimal Literal*

If any integric literal starts with 0X or 0x (Zero capital X Or 0 small x) then it is hexadecimal literal. Here base is 16 so it will accept 16 digits i.e 0 to 9 and A to F OR [a to f].

*Example:*

*int x = 0X12; //Valid*

*int y = 0xadd; //Valid*

*int z = 0Xface; //valid*

*int a = 0Xage; //Inavlid because character 'g' out of range*

*Binary Literal*

If a numeric literal starts with 0B (Zero capital B) or 0b (Zero small b) then it will become Binary literal. Binary literal is available from JDK 1.7v.

Here base is 2 so it will accept 2 digits i.e 0 and 1.

*Example:*

int x = 0B101; //valid

int y = 0b111; //Valid

int z = 0B112; //Invalis [2 is out of range]

The deafult type is decimal literal so to generate the output for any different literal JVM converts into decimal literal.

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

*//Programs*

*//Octal Representation*

public class Test

{

public static void main(String[] args)

{

int x = 015;

System.out.println(x);

}

}

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

*//Hexadecimal Representation*

public class Test

{

public static void main(String[] args)

{

int x = 0xadd;

System.out.println(x);

}

}

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

*//Binary Literal*

public class Test

{

public static void main(String[] args)

{

int x = 0b101;

System.out.println(x);

}

}

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

By default every integral literal is of type int only. byte and short are below than int so we can assign integral literal (Which is by default int type) to byte and short but the values must be within the range. [for Byte -128 to 127 and for short -32768 to 32767]

Actually whenever we are assigning integral literal to byte and short data type then compiler internally converts into corresponding type.

byte b = (byte) 12; [Compiler is converting int to byte]

short s = (short) 12; [Compiler is converting int to short]

In order to represent long value we should use either L OR l (Capital L OR Small l) as a suffix to integral literal.

According to IT industry, we should use L because l looks like digit 1.

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

*//By default every integral literal is of type int only*

public class Test4

{

public static void main(String[] args)

{

byte b = 128; *//error*

System.out.println(b);

short s = 32768; *//error*

System.out.println(s);

}

}

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

//Assigning smaller data type value to bigger data type

public class Test5

{

public static void main(String[] args)

{

byte b = 125;

short s = b; *//Automatic type casting OR Widening*

System.out.println(s);

}

}

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

*//Converting bigger type to smaller type*

public class Test6

{

public static void main(String[] args)

{

short s = 127;

byte b = (byte) s; *//explicit type Casting (Narrowing)*

System.out.println(b);

}

}

-----------

21-10-2024

-----------

public class Test7

{

public static void main(String[] args)

{

byte x = (byte) 1L;

System.out.println("x value = "+x);

long l = 29L;

System.out.println("l value = "+l);

int y = (int)18L;

System.out.println("y value = "+y);

}

}

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

**Is java pure Object Oriented language or not?**

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

No, Java is not a pure object oriented langauge because it is accepting primary data type, Actually any language which accepts primary data type is not a pure object oriented language.

Only Objects are moving in the network but not the primary data type so java has introduced Wrapper class concept to convert the primary data types into corresponding wrapper object.

**Primary Data type** **Wrapper Object**

byte : Byte

short : Short

int : Integer

long : Long

float : Float

double : Double

char : Character

boolean : Boolean

Note : Apart from these 8 data types, Everything is an object in java so, if we remove all these 8 data types then java will become pure OOP language.

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

*//Wrapper claases*

public class Test8

{

public static void main(String[] args)

{

Integer x = 24;

Integer y = 24;

Integer z = x + y;

System.out.println("The sum is :"+z);

Boolean b = true;

System.out.println(b);

Double d = 90.90;

System.out.println(d);

Character c = 'A';

System.out.println(c);

}

}

Note: From JAVA 1.5 version we have two conecpt:

AutoBoxing : Converting Primitive to Wrapper Object (int to Integer)

UnBoxing : Converting Wrapper object back to primitive (Integer to int )

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

How to find out the minimum, maximum value as well as size of different data types:

The Wrapper classes like Byte, Short, Integer and Long has provided predefined static and final variables to represent minimum value, maximum value as well as size of the respective data type.

*Example:*

If we want to get the minimum value, maximum value as well as size of byte data type then Byte class (Wrapper class) has provided the following final and static variables

Byte.MIN\_VALUE : 128

Byte.MAX\_VALUE : 127

Byte.SIZE : 8 (bits format)

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

*//Program to find out the range and size of Integeral Data type*

public class Test9

{

public static void main(String[] args)

{

System.out.println("\n Byte range:");

System.out.println(" min:" + Byte.MIN\_VALUE);

System.out.println(" max: " + Byte.MAX\_VALUE);

System.out.println(" size:"+Byte.SIZE);

System.out.println("\n Short range:");

System.out.println(" min: " + Short.MIN\_VALUE);

System.out.println(" max: " + Short.MAX\_VALUE);

System.out.println(" size: "+Short.SIZE);

System.out.println("\n Integer range:");

System.out.println(" min: " + Integer.MIN\_VALUE);

System.out.println(" max: " + Integer.MAX\_VALUE);

System.out.println(" size: "+Integer.SIZE);

System.out.println("\n Long range:");

System.out.println(" min: " + Long.MIN\_VALUE);

System.out.println(" max: " + Long.MAX\_VALUE);

System.out.println(" size: "+Long.SIZE);

}

}

Providing \_ (underscore) in integeral Literal:

In Order to enhance the readability of large numeric literals, Java software people has provided \_ from JDK 1.7V. While writing the big numbers to separate the numbers we can use \_

We can't start or end an integral literal with \_ we will get compilation error.

*//We can provide\_in integral literal*

public class Test10

{

public static void main(String[] args)

{

long mobile = 98\_1234\_5678L;

System.out.println("Mobile Number is:"+mobile);

}

}

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

public class Test11

{

public static void main(String[] args)

{

final int x = 127;

byte b = x;

System.out.println(b);

}

}

Note : The above program will compile and execute without explicit type casting because variable is declared with final modifier.

**How to convert decimal number to Octal, Hexadecimal and Binary:**

Integer class has provided the following static methods to convert decimal to octal, hexadecimal and binary.

1) public static String toBinaryString(int x) : Will convert the decimal into binary in String format.

2) public static String toOctalString(int x) : Will convert the decimal into octal in String format.

3) public static String toHexString(int x) : Will convert the decimal into hexadecimal in String format.

Note: All the above static methods are available in Integer class.

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

*// Converting from decimal to another number system*

public class Test12

{

public static void main(String[] argv)

{

*//decimal to Binary*

System.out.println(Integer.toBinaryString(7)); //111

*//decimal to Octal*

System.out.println(Integer.toOctalString(15)); //17

*//decimal to Hexadecimal*

System.out.println(Integer.toHexString(2781)); //add

}

}

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

**var keyword in java:**

From java 10v, java software people has provided var keyword.

This var keyword can hold any kind of value but initialization is compulsory in the same line [compiler will come to know about the variable type based on value]

var x = 12; //x is int type

After initialization it will hold same kind of value till the end of the program.

var a = 12;

a = true; //Invalid

It can be used for only for local variable.

*//var keyword [Introduced from java 10]*

public class Test13

{

public static void main(String[] args)

{

var x = 12;

System.out.println(x);

x = 90;

System.out.println(x);

// x = "NIT"; //Invalid

}

}

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

22-10-2024

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

**Floating Point Literal**

If any numeric literal contains decimal or fraction then it is called Floating point literal.

*Example :* 3.4, 90.67, 12.67, 0.1

***In floating point literal we have 2 data types***

a) float (32 bits)

b) double (64 bits)

By default every floating point literal is of type double only so, the following statement will generate compilation error

float x = 1.2; //Invalid

float f1 = (float) 1.2; //Valid

float f2 = 12.90F; //Valid

float f3 = 19.15f; //Valid

Even though, every floating point literal is of type double only but still compiler has provided two flavors to represent double value explicitly to enhance the readability of the code.

double d1 = 12d;

double d2 = 90D;

Floating point literal we can represent in exponent form with positive and negative value.

double d1 = 15e2; [15 X 10 to the power of 2]

\* An integral literal we can represent in four different forms i.e. decimal, octal, hexadecimal and binary but a floating point literal we can represent in only one form i.e. decimal.

\* An integral literal i.e. byte, short, int and long we can assign to floating point literal i.e float and double but floating point literal we can't assign to integral literal.

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

//Programs

public class Test

{

public static void main(String[] args)

{

float f = 0.0; //error

System.out.println(f);

}

}

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

public class Test1

{

public static void main(String[] args)

{

float b = 15.29F;

float c = 15.25f;

float d = (float) 15.30;

System.out.println(b + " : "+c+ " : "+d);

}

}

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

public class Test2

{

public static void main(String[] args)

{

double d = 15.15;

double e = 15d;

double f = 90D;

System.out.println(d+" , "+e+" , "+f);

}

}

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

public class Test3

{

public static void main(String[] args)

{

double x = 0129.89;

double y = 0167;

double z = 0178; //error

System.out.println(x+","+y+","+z);

}

}

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

class Test4

{

public static void main(String[] args)

{

double x = 0X29;

double y = 0X91.5; //error

System.out.println(x+","+y);

}

}

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

public class Test5

{

public static void main(String[] args)

{

double d1 = 15e-3;

System.out.println("d1 value is :"+d1);

double d2 = 15e3;

System.out.println("d2 value is :"+d2);

}

}

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

public class Test6

{

public static void main(String[] args)

{

double a = 0791; //error

double b = 0791.0;

double c = 0777;

double d = 0Xdead;

double e = 0Xdead.0; //error

}

}

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

public class Test7

{

public static void main(String[] args)

{

double a = 1.5e3;

float b = 1.5e3; //error

float c = 1.5e3F;

double d = 10;

int e = 10.0; //error

long f = 10D; //error

int g = 10F; //error

long l = 12.78F; //error

}

}

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

*//Range and size of floating point literal*

public class Test8

{

public static void main(String[] args)

{

System.out.println("\n Float range:");

System.out.println(" min: " + Float.MIN\_VALUE);

System.out.println(" max: " + Float.MAX\_VALUE);

System.out.println(" size :"+Float.SIZE);

System.out.println("\n Double range:");

System.out.println(" min: " + Double.MIN\_VALUE);

System.out.println(" max: " + Double.MAX\_VALUE);

System.out.println(" size :"+Double.SIZE);

}

}

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

**char Literal**

It is also known as Character Literal.

In char literal we have only data type i.e. char data type which accepts 16 bits of memory.

***char literal we can represent in the following ways :***

a) Single character enclosed with single quotes.

char ch = 'A';

b) In older languages like C and C++ which support ASCII format and the range is 0 - 255, On the other hand java supports UNICODE format (ASCII + NON ASCII) where the range is 0 - 65535.

char ch = 65535; //Valid

c) We can assign character literal to integral literal to know the UNICODE value of that particular character.

int x = 'A'; //Will return Unicode value of A (65)

d) A character literal we can also represent in 4 digit hexadecimal number which is UNICODE representation, the format is: '\udddd'

Here \u represents UNICODE and d represents digit

Minimum Range is: '\u0000' Maximum range is : '\uffff'

e) All escape sequences can also be represented by char literal.

char ch = '\n';

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

//Programs :

public class Test1

{

public static void main(String[] args)

{

char ch1 = 'a';

System.out.println("ch1 value is:"+ch1);

char ch2 = 97;

System.out.println("ch2 value is:"+ch2);

}

}

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

public class Test2

{

public static void main(String[] args)

{

int ch = 'A';

System.out.println("ch value is :"+ch);

}

}

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

//The UNICODE value for ? character is 63

public class Test3

{

public static void main(String[] args)

{

char ch1 = 63;

System.out.println("ch1 value is :"+ch1);

char ch2 = 64;

System.out.println("ch2 value is :"+ch2);

char ch3 = 65;

System.out.println("ch3 value is :"+ch3);

}

}

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

public class Test4

{

public static void main(String[] args)

{

char ch1 = 45000;

System.out.println("ch1 value is :"+ch1);

char ch2 = 0Xadd;

System.out.println("ch2 value is :"+ch2);

}

}

Note: We will get the output as? because the equivalent language translator is not available in the System.

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

*//Addition of two character in the form of Integer*

public class Test5

{

public static void main(String txt[])

{

int x = 'A';

int y = 'B';

System.out.println(x + y); //131

System.out.println('A'+'B'); //131

}

}

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

//*Range of UNICODE Value (65535) OR '\uffff'*

class Test6

{

public static void main(String[] args)

{

char ch1 = 65535;

System.out.println("ch value is :"+ch1);

char ch2 = 65536; //error [Out of UNICODE range]

System.out.println("ch value is :"+ch2);

}

}

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

*//WAP in java to describe unicode representation of char in hexadecimal format*

public class Test7

{

public static void main(String[] args)

{

int ch1 = '\u0000';

System.out.println(ch1); //0

int ch2 = '\uffff';

System.out.println(ch2); //65535

char ch3 = '\u0041';

System.out.println(ch3); //A

char ch4 = '\u0061';

System.out.println(ch4); //a

}

}

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

class Test8

{

public static void main(String[] args)

{

char c1 = 'A';

char c2 = 65;

char c3 = '\u0041';

System.out.println("c1 = "+c1+", c2 ="+c2+", c3 ="+c3);

}

}

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

class Test9

{

public static void main(String[] args)

{

int x = 'A';

int y = '\u0041';

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

}

}

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

*//Every escape sequence is char literal*

class Test10

{

public static void main(String [] args)

{

char ch ='\n';

System.out.println("Hello");

System.out.println(ch);

}

}

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

public class Test11

{

public static void main(String[] args)

{

System.out.println(Character.MIN\_VALUE); //white space

System.out.println(Character.MAX\_VALUE); //?

System.out.println(Character.SIZE); //16 bits

}

}

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

**boolean literal**

It is used to represent two states i.e. true or false.

Here we have only one data type i.e boolean data type which accepts 1 bit of memory or depends upon JVM implementation.

boolean isValid = true;

boolean isEmpty = false;

Unlike C and C++ we can't assign 0 and 1 to boolean data type becuase in java 0 and 1 are treated as int type.

boolean x = 0; [Valid in C but invalid in java]

boolean y = 1; [Valid in C but invalid in java]

We can't assign String value to boolean type as shown below:

boolean b = "true"; //Invalid

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

//Programs :

public class Test1

{

public static void main(String[] args)

{

boolean isValid = true;

boolean isEmpty = false;

System.out.println(isValid);

System.out.println(isEmpty);

}

}

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

public class Test2

{

public static void main(String[] args)

{

boolean c = 0; //error

boolean d = 1; //error

System.out.println(c);

System.out.println(d);

}

}

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

public class Test3

{

public static void main(String[] args)

{

boolean x = "true";

boolean y = "false";

System.out.println(x);

System.out.println(y);

}

}

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

**String Literal**

String is a predefined class available in java.lang Package.

String is a collection of alpha-nemeric character which is enclosed by double quotes. These characters can be alphabets, numbers, symbol or any special character.

***In java we can create String object by using following 3 ways***

1) By using String Literal String str1 = "india";

2) By using new keyword String str2 = new String("Hyderabad");

3) By using Character array [Old Technique] char ch[] = {'R', 'A', 'J'};

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

*// Programs :*

*//Three Ways to create the String Object*

public class StringTest1

{

public static void main(String[] args)

{

String s1 = "Hello World"; //Literal

System.out.println(s1);

String s2 = new String("Ravi"); //Using new Keyword

System.out.println(s2);

char s3[] = {'H','E','L','L','O'}; //Character Array

System.out.println(s3);

}

}

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

*//String is collection of alpha-numeric character*

public class StringTest2

{

public static void main(String[] args)

{

String x="B-61 Hyderabad";

System.out.println(x);

String y = "123";

System.out.println(y);

String z = "67.90";

System.out.println(z);

String p = "A";

System.out.println(p);

}

}

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

*//IQ*

public class StringTest3

{

public static void main(String []args)

{

String s = 15+29+"Ravi"+40+40;

System.out.println(s);

}

}

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

**Punctuators**

It is also called separators.

It is used to inform the compiler how things are grouped in the code.

() {} [] ; , . @ (var args)

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

*What is a local variable?*

If a variable is declared inside the body of method OR block OR Constructor then it is called Local /Automatic/ Stack/ temporary variable.

*Example:*

public void accept()

{

int x = 100; //x is a local variable

}

* A local variable must be initialized by the developer before use because local variable does not have any default value.
* We can't apply any kind of access modifier on local variable except final.

public void accept()

{

final int x = 100; //final is a valid modifier

}

* As far as it's scope is concerned, It must be used within the same method body only that menas we can't use local variable outside of the method body.
* All local variables are the part of the method body, all the methods are executed in a special memory in java called Stack Memory so local variables are the part of Stack memory.
* A local variable must be pre-declared and initialized before use.

public void m1()

{

System.out.println(x); //error

int x = 100;

}

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

24-10-2024

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

*Why local variables are not accessible outside of the method?*

In java, Every methods are executed in a special memory called Stack Memory.

Stack Memory works on LIFO (Last In First Out) basis.

In java, Whenever we call a method then a separate Stack Frame will be created for each and every method.

[15-OCT]

Once the method execution is over then the corresponding method Stack frame will also be deleted from Stack Area, that is the reason we can't use local variable outside of the method.

***Each stack frame contains 3 parts***

1) Local Variable Array

2) Frame Data

3) Operand Stack

*//Program*

package com.ravi.method\_execution;

public class MethodExecution

{

public static void main(String[] args)

{

System.out.println("Main Method Started!!!");

m1();

System.out.println("Main Method Ended!!!");

}

public static void m1()

{

System.out.println("m1 Method Started!!!");

m2();

System.out.println("m1 Method Ended!!!");

}

public static void m2()

{

System.out.println("Hii I am m2 method");

}

}

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

*Limitation of Commnd line Argument ?*

As we know by using Command Line Argument, we can pass some value at runtime, These values are stroed in String array variable and then only the exceution of the program will be started.

In Command line Argumenet we can't ask to enter the value from our end user as shown in the Program.

//Program

package com.ravi.command;

*//Read Gender [M/F] from Command Line Argument*

public class ReadCharacter

{

public static void main(String[] args)

{

System.out.println("Enter your Gender [M/F]");

char gender = args[0].charAt(4);

System.out.println("Your Gender is:"+gender);

}

}

Note: In the above program, after providing the gender value, It is asking for Gender which is not a recommended way.

Note: charAt (int indexPosition) is a predefined non static method of String class, which is used to retrieve a character from the given String.

public char charAt(int indexPosition)

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

How to read the data from the End user with user friendly mesasage :

There are so many ways we can read the data from client which are as follows :

1) DataInputStream class (java.io)

2) BufferedReader class (java.io)

3) System.in.read() (java.lang)

4) Console (java.io)

5) Scanner class (java.util)

**Scanner class**

It is a predefined class available in java.util pacakge from JDK 1.5v.

It is used to read the data from the client with user-friendly message.

static variables of System class

***System class has provided the following final and static variables which are as follows:***

1) System.out : Used to print normal message.

2) System.err : Used to print error message. (Red Color)

3) System.in : Used to read the data from the Source.

***How to create an Object for Scanner class ?***

Scanner sc = new Scanner(System.in);

***Non static methods of Scanner class***

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

1) public String next() : Used to read a single word.

2) public String nextLine() : Used to read multiple words or complete line.

3) public byte nextByte() : Used to read byte value.

4) public short nextShort() : Used to read short value.

5) public int nextInt() : Used to read int value.

6) public long nextLong() : Used to read long value.

7) public float nextFloat() : Used to read float value.

8) public double nextDouble() : Used to read double value.

9) public boolean nextBoolean() : Used to read boolean value.

10) public char next().charAt(0) : Will read a single characetr

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

*// WAP to read your name from end user.*

import java.util.Scanner;

public class ReadName

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

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

String name = sc.nextLine();

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

}

}

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

*// WAP to read you age from Scanner class :*

package com.ravi.scanner\_demo;

import java.util.Scanner;

public class ReadAge

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

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

int age = sc.nextInt();

System.out.print("Your Age is :"+age);

sc.close();

}

}

-----------

25-10-2024

**Operators**

It is a symbol which describes that how a calculation will be performed on operands.

***Types Of Operators***

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

1) Arithmetic Operator (Binary Operator)

2) Unary Operators

3) Assignment Operator

4) Relational Operator

5) Logical Operators

6) Boolean Operators

7) Bitwise Operators

8) Ternary Operator

9) Member Operator

10) new Operator

11) instanceof Operator

*// Basic Concepts of Operators*

public class Test

{

public static void main(String[] args)

{

int x = 15;

int y = x++;

System.out.println(x + " : "+ y);

}

}

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

class Test

{

public static void main(String[] args)

{

int x = 15;

int y = --x;

System.out.println(x + " : "+ y);

}

}

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

class Test

{

public static void main(String[] args)

{

int x = 20;

int y = ++20; //erro

System.out.println(x + " : "+ y);

}

}

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

class Test

{

public static void main(String[] args)

{

int x = 20;

int y = ++(++x);

System.out.println(x + " : "+ y);

}

}

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

class Test

{

public static void main(String[] args)

{

char ch = 'A';

ch++;

System.out.println(ch);

}

}

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

class Test

{

public static void main(String[] args)

{

double d1 = 12.89;

d1++;

System.out.println(d1);

}

}

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

class Test

{

public static void main(String[] args)

{

while(false)

{

System.out.println("Hello");

}

System.out.println("World");

}

}

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

class Test

{

public static void main(String[] args)

{

boolean b = false;

while(b)

{

System.out.println("Hello");

}

System.out.println("World");

}

}

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

class Test

{

public static void main(String[] args)

{

final int x = 10;

final int y = 20;

while(x > y)

{

System.out.println("x is Greater than y");

}

System.out.println("Hello World");

}

}

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

class Test

{

public static void main(String[] args)

{

do

{

int x = 1;

System.out.println("Hello");

x++;

}

while (x<=10); //error

}

}

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

class Test

{

public static void main(String[] args)

{

int x = 1;

do

{

System.out.println("Hello");

x++;

boolean b = false;

if(b = true)

break;

}

while (x<=10);

}

}

**Expression Conversion**

Whenever we are working with Arithmetic Operator (+,-,\*,/,%) or unary minus operator, after expression exeution the result will be converted (Promoted) to int type, Actually to store the result minimum 32 bits data format is required.

Unary Minus Operator

class Test

{

public static void main(String[] args)

{

int x = 15;

System.out.println(-x);

}

}

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

class Test

{

public static void main(String[] args)

{

byte b = 1;

byte c = 1;

byte d = b + c;

System.out.println(d);

}

}

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

class Test

{

public static void main(String[] args)

{

short s = 12;

short t = 14;

short u = s + t;

System.out.println(u);

}

}

After Arithmetic operator expression the result will be promoted to int type so, to hold the result minimum 32 bit data is required.

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

class Test

{

public static void main(String[] args)

{

byte s = 1;

byte t = 1;

byte u = s + t; //error

System.out.println(u);

}

}

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

class Test

{

public static void main(String[] args)

{

byte b = 2;

byte c = -b; //error

System.out.println(c);

}

}

In Arithmetic operator OR Unary minus operator, the result will be promated to int type (32 bits) so to hold the result int data type is reqd.

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

class Test

{

public static void main(String[] args)

{

/\*

byte b = 1;

b = b + 2;

System.out.println(b); \*/

byte b = 2;

b += 2; //It is valid because short hand operator

System.out.println(b);

}

}

In the above program we are using short hand operator so we will get the result in byte format also.

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

Program On Boolean Operator

class Test

{

public static void main(String[] args)

{

int z = 5;

if(++z > 5 || ++z > 6) //Logical OR

{

z++;

}

System.out.println(z); //7

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

z = 5;

if(++z > 5 | ++z > 6) //Boolean OR

{

z++;

}

System.out.println(z); //8

}

}

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

class Test

{

public static void main(String[] args)

{

int z = 5;

if(++z > 6 & ++z> 6)

{

System.out.println("Inside If");

z++;

}

System.out.println(z);

}

}

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

**Working with Bitwise AND(&), Bitwise OR(|) and Bitwise X-OR (^) :**

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

class Test

{

public static void main(String[] args)

{

System.out.println(true ^ false);

}

}

Here we will get the output based on the input i.e Same input output will be zero.

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

class Test

{

public static void main(String[] args)

{

System.out.println(5 & 6); //4

System.out.println(5 | 6); //7

System.out.println(5 ^ 6); //3

}

}

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

**Bitwise Complement Operator (~)**

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

It will not work with boolean.

public class Test

{

public static void main(String [] args)

{

System.out.println(~ true); //error

}

}

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

public class Test

{

public static void main(String [] args)

{

System.out.println(~ 5); //-6

System.out.println(~ -4); //3

}

}

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

**Member Operator (.)**

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

It is also known as Dot operator OR Period.

It is used to access the member of the class.

If we want to access the variables OR methods which are available in the class then we should use member access Operator.

package com.ravi.m1;

class Welcome

{

static int x = 100; //static variable

public static void accept() //static method

{

System.out.println("Accept static method");

}

}

public class MemberOperator {

public static void main(String[] args)

{

System.out.println(Welcome.x);

Welcome.accept();

}

}

Note : Welcome class contains static variable and static method so, we can directly call static veraible and static method with the help of class name using dot operator.

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

26-10-2024

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

**new keyword**

It is a keyword as well as Operator.

It is used to create the Object so by using Object reference we can access the non static member of the class.

It is used to provide default values to non static variable.

package com.ravi.new\_keyword;

class Welcome

{

int x = 500; //non static variable

public void show() //non static method

{

System.out.println("Hello batch 40");

}

}

public class NewKeywordDemo

{

public static void main(String[] args)

{

Welcome w = new Welcome();

System.out.println(w.x);

w.show();

}

}

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

**Limitation of if else**

The major drawback with if condition is, it checks the condition again and again so It increases the burdon over CPU so we introduced switch-case statement to reduce the overhead of the CPU.

**Switch case statement in java**

It is a selective statement so, we can select one statement among the available statements.

break is optional but if we use break then the control will move from out of the switch body.

We can write default so if any statement is not matching then default will be executed.

In switch case we can't pass long, float and double and boolean value.

[long we can pass in switch case from java 14v]

We can pass String from JDK 1.7v and we can also pass enum from JDK 1.5v.

import java.util.\*;

public class SwitchDemo

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("Please Enter a Character :");

char colour = sc.next().toLowerCase().charAt(0);

switch(colour)

{

case 'r' : System.out.println("Red") ; break;

case 'g' : System.out.println("Green");break;

case 'b' : System.out.println("Blue"); break;

case 'w' : System.out.println("White"); break;

default : System.out.println("No colour");

}

System.out.println("Completed") ;

}

}

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

import java.util.Scanner;

public class SwitchDemo1

{

public static void main(String args[])

{

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

System.out.println("\t\t\*\*100 Police\*\*\n");

System.out.println("\t\t\*\*101 Fire\*\*\n");

System.out.println("\t\t\*\*102 Ambulance\*\*\n");

System.out.println("\t\t\*\*139 Railway\*\*\n");

System.out.println("\t\t\*\*181 Women's Helpline\*\*\n");

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

Scanner sc = new Scanner(System.in);

int choice = sc.nextInt();

switch(choice)

{

case 100:

System.out.println("Police Services");

break;

case 101:

System.out.println("Fire Services");

break;

case 102:

System.out.println("Ambulance Services");

break;

case 139:

System.out.println("Railway Enquiry");

break;

case 181:

System.out.println("Women's Helpline ");

break;

default:

System.out.println("Your choice is wrong");

}

}

}

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

// Passing String value in switch case

import java.util.\*;

public class SwitchDemo2

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter the name of the season :");

String season = sc.next().toUpperCase();

switch(season) //String allowed from 1.7

{

case "SUMMER" : System.out.println("It is summer Season!!");

break;

case "RAINY" : System.out.println("It is Rainy Season!!");

break;

}

}

}

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

public class Test

{

public static void main(String[] args)

{

long l = 12L;

switch(l) //error

{

case 12L :

System.out.println("It is case 12");

break;

}

}

}

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

public class Test

{

public static void main(String[] args)

{

float l = 12;

switch(l)

{

case 12F :

System.out.println("It is case 12");

break;

}

}

}

Note: We can't pass long, float and double value.

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

public class Test

{

public static void main(String[] args)

{

int x = 12;

int y = 12;

switch(x)

{

case y : //error

System.out.println("It is case 12");

break;

}

}

}

Note: In the label of switch we should take constant value.

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

public class Test

{

public static void main(String[] args)

{

int x = 12;

final int y = 12;

switch(x)

{

case y :

System.out.println("It is case 12");

break;

}

}

}

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

public class Test

{

public static void main(String[] args)

{

byte b = 90;

switch(b)

{

case 128 : //error

System.out.println("It is case 127");

break;

}

}

}

Note : Value 128 is out of the range of byte and same applicable for short data type

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

**Loops in java**

**A loop is nothing but repeatation of statements based on the specified condition.**

***In java we have 4 types of loops :***

1) do-while loop

2) while loop

3) for loop

4) for each loop

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

*//program on do-while loop*

public class DoWhile

{

public static void main(String[] args)

{

do

{

int x = 1; //Local Variable (block Level)

System.out.println("x value is :"+x);

x++;

}

while (x<=10); //error

}

}

Note: x is a block level variable because It is declared inside do block so the scope of this x variable will be within the do block only.

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

package com.ravi.loop;

public class DoWhileDemo

{

public static void main(String[] args)

{

int x = 1;

do

{

System.out.println("x value is :"+x);

x++;

}

while(x<=10);

}

}

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

package com.ravi.loop;

public class WhileLoop

{

public static void main(String[] args)

{

int x = 1;

while(x>=-10)

{

System.out.println(x);

x--;

}

}

}

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

*//Program on for loop*

package com.ravi.loop;

public class ForLoop

{

public static void main(String[] args)

{

for(int i=1; i<=10; i++)

{

System.out.println(i);

}

}

}

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

28-10-2024

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

**For Each loop**

It is an enhanced for loop.

It is introduced from JDK 1.5v.

It is used to retrieve OR Iterate the values one by one from the Collection like array.

package com.ravi.for\_each\_loop;

public class ForEachDemo1 {

public static void main(String[] args)

{

int []arr = {20,40,60,50,89,56,45};

for(int x : arr)

{

System.out.println(x);

}

}

}

Note: Internally the compiler will convert this for-each loop into Ordinary for loop.

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

How to sort Array data

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

In java.util pacakge, there is a predefined class called Arrays which has various static methods to sort the array in ascending or alphabetical order.

*Example*

Arrays.sort(int []arr); //For sorting int array

Arrays.sort(Object []arr) //For sorting String array

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

package com.ravi.for\_each\_loop;

public class ForEachDemo2 {

public static void main(String[] args)

{

int [] values = {30,10,20,50,40};

java.util.Arrays.sort(values);

for(int value : values)

{

System.out.println(value);

}

}

}

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

package com.ravi.for\_each\_loop;

import java.util.Arrays;

public class ForEachDemo3 {

public static void main(String[] args)

{

String []fruits = {"Mango","Apple","Guava","Grapes"};

Arrays.sort(fruits);

for(String fruit : fruits)

{

System.out.println(fruit);

}

}

}

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

In java, Can we hold hetrogeneous types of data using array?

Yes, by using Object array we can hold hetrogeneous type of data but we can't peform sorting operation using Arrays.sort(), It will generate

java.lang.ClassCastException

package com.ravi.for\_each\_loop;

public class ForEachDemo4

{

public static void main(String[] args)

{

Object []arr = {'A',12,89.67,true, new String("NIT")};

//Arrays.sort(arr); //java.lang.ClassCastException

for(Object x : arr)

{

System.out.println(x);

}

}

}

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

**What is BLC and ELC class in java?**

**BLC**

BLC stands for **Business Logic class**, In this class we are responsible to write the logic. This class will not contain main method.

The main purpose of this BLC class to reuse this class in various packages.

***Example:***

*//BLC*

public class Calculate

{

*//Here We are responsible to write the logic*

}

**ELC**

It stands for Executable Logic class, It will not contain any logic but the execution of the program will start from this ELC class because it contains main method.

***Example***

*//ELC*

public class Main

{

public static void main(String [] args)

{

}

}

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

**How to reuse a class in java?**

The slogan of java is "WORA" write once run anywhere.

A public class created in one package can be reuse from different packages also by using import statement.

In a single .java file, we can declare only one public class that must be our .java file and that class can be reusable to all the packages.

\*In a single java file, we can write only one public class and multiple non-public classes but it is not a recommended approach because the non public class we can use within the same package only.

So the conclusion is, we should declare every java class in a separate file to enhance the reusability of the BLC classes.

[Note we have 10 classes -> 10 java files]

How many .class file will be created in the above approach?

For a public class in a single file, Only 1 .class file will be created.

For a public class in a single file which contains n number of non public classes then compiler will generate n number of .class file.

***Example***

---- Test.java ----

public class Test

{

}

class A

{

}

class B

{

}

class C

{

}

Note: Here total 4 .class file will be generated.

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

*Here we have 2 packages:*

com.ravi.application

com.ravi.execution

Calculate.java(BLC) *[Available in com.ravi.application]*

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

package com.ravi.application;

//BLC

public class Calculate

{

public static void doSum(int x, int y)

{

System.out.println("Sum is :"+(x+y));

}

}

Main.java(ELC) *[Available in com.ravi.application]*

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

package com.ravi.application;

//ELC

public class Main

{

public static void main(String[] args)

{

Calculate.doSum(10, 20);

}

}

ELC.java (ELC) *[Available in com.ravi.execution]*

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

This ELC class will import Calculate.java from com.ravi.application package as shown below.

package com.ravi.execution;

import com.ravi.application.Calculate;

public class ELC

{

public static void main(String[] args)

{

*//Importing and Using Calculate class*

Calculate.doSum(10, 10);

}

}

-----------

29-10-2024

-----------

**Working with Static Method with different return type**

If a static method is defined inside the ELC class then we can directly call the static method from the main method, class name is not required.

*//A static method can be directly call within the same class*

package com.ravi.pack1;

public class Test1

{

public static void main (String[] args)

{

square(5);

}

public static void square(int x)

{

System.out.println("Square is :"+(x\*x));

}

}

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

**2 files**

---- GetSquare.java ----

package com.ravi.pack2;

//BLC

public class GetSquare

{

public static void getSquareOfNumber(int num)

{

System.out.println("Square of "+num+" is :"+(num\*num));

}

}

---- Test2.java ----

package com.ravi.pack2;

import java.util.Scanner;

//ELC

public class Test2

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter the side :");

int side = sc.nextInt();

GetSquare.getSquareOfNumber(side);

sc.close();

}

}

Here getSquareOfNumber() method return type is void so there is no communication between BLC and ELC class.

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

**2 files:**

---- FindSquare.java ----

//A static method returning integer value

package com.ravi.pack3;

//BLC

public class FindSquare

{

public static int getSquare(int x)

{

return (x\*x);

}

}

---- Test3.java ----

package com.ravi.pack3;

import java.util.Scanner;

//ELC

public class Test3

{

public static void main (String[] arg)

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter the value of side :");

int side = sc.nextInt();

System.out.println("Square of "+side+" is:"+FindSquare.getSquare(side));

sc.close();

}

}

Note: By using System.out.println(), we can't call a method whose return type is void. We will get compilation error as shown below

**2 files:**

---- BLC.java ----

public class BLC

{

public static void greet() //Method return type is void

{

}

}

---- ELC.java ----

public class ELC

{

public static void main(String[] args)

{

System.out.println(BLC.greet()); //error

}

}

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

**2 files:**

---- Calculate.java ----

/\* Program to find out the square and cube of the number by following criteria

a) If number is 0 or Negative it should return -1

b) If number is even It should return square of the number

c) If number is odd It should return cube of the number \*/

package com.ravi.pack4;

//BLC

public class Calculate

{

public static int getSquareAndCube(int num)

{

if(num <=0)

{

return -1;

}

else if(num%2 == 0)

{

return num\*num;

}

else

{

return num\*num\*num;

}

}

}

---- Test4.java ----

package com.ravi.pack4;

import java.util.Scanner;

public class Test4

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter a number :");

int num = sc.nextInt();

int result = Calculate.getSquareAndCube(num);

System.out.println("Result is :"+result);

sc.close();

}

}

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

**2 files**

---- Rectangle.java ----

package com.ravi.pack5;

//BLC

public class Rectangle

{

public static double getAreaOfRectangle(double length, double breadth)

{

return (length \* breadth);

}

}

---- Test5.java ----

package com.ravi.pack5;

import java.util.Scanner;

public class Test5

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter the length of the Rect :");

double length = sc.nextDouble();

System.out.print("Enter the breadth of the Rect :");

double breadth = sc.nextDouble();

double areaOfRectangle = Rectangle.getAreaOfRectangle(length, breadth);

System.out.printf("Area of Rectangle is :%.2f",areaOfRectangle);

sc.close();

}

}

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

**2 files:**

---- EvenOrOdd.java ----

package com.ravi.pack6;

//BLC

public class EvenOrOdd

{

public static boolean isEven(int num)

{

return (num % 2 == 0);

}

}

---- Test6.java ----

package com.ravi.pack6;

import java.util.Scanner;

//ELC

public class Test6

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter a Number :");

int num = sc.nextInt();

boolean isEven = EvenOrOdd.isEven(num);

System.out.println(num+" is Even ?:"+isEven);

System.out.print("Enter another Number :");

num = sc.nextInt();

isEven = EvenOrOdd.isEven(num);

System.out.println(num+" is Even ?:"+isEven);

sc.close();

}

}

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

**2 files**

---- Circle.java ----

*//Area of Circle*

*//If the radius is 0 or Negative then return -1.*

package com.ravi.pack7;

public class Circle

{

public static String getAreaOfCircle(double radius)

{

if(radius <=0)

{

return ""+(-1);

}

else

{

final double PI = 3.14;

double area = PI \* radius \* radius;

return ""+area;

}

}

}

---- Test7.java ----

package com.ravi.pack7;

import java.util.Scanner;

public class Test7

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter the radius :");

double radius = sc.nextDouble();

String areaOfCircle = Circle.getAreaOfCircle(radius);

double area = Double.parseDouble(areaOfCircle);

System.out.printf("Area of Circle is : %.2f",area);

sc.close();

}

}

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

**2 files**

---- Student.java ----

package com.ravi.pack8;

//BLC

public class Student

{

public static String getStudentDetails(int roll, String name, double fees)

{

*//[Student name is : Ravi, roll is : 101, fees is :1200.90]*

return "[Student name is :"+name+", roll is :"+roll+", fees is :"+fees+"]";

}

}

---- Test8.java ----

package com.ravi.pack8;

public class Test8

{

public static void main(String[] args)

{

System.out.println(Student.getStudentDetails(101, "Scott", 12000));

}

}

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

**2 files**

--- Table.java ----

package com.ravi.pack9;

//BLC

public class Table

{

public static void printTable(int num) //5 X 1 = 5

{

for(int i=1; i<=10; i++)

{

System.out.println(num+" X "+i+" = "+(num\*i));

}

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

}

}

---- Test9.java ----

package com.ravi.pack9;

//ELC

public class Test9

{

public static void main(String[] args)

{

for(int i=1; i<=10; i++)

{

Table.printTable(i);

}

}

}

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

**Types of Variable in java**

Based on the data type we have only 2 types of variable in java :

1) Primitive Variables

2) Reference Variables

**1) Primitive Variables**

If a variable is declared with primitive data type like byt, short, int, long and so on then it is called Primitive Variables.

***Example***

int x = 100;

boolean y = true;

**2) Reference Variable**

If a variable is declared with class name, interface name, enum, record and so on then it is called reference variable.

***Example***

Student s; //s is reference variable

Scanner sc = new Scanner(System.in); //sc is reference variable

Integer y = null; //y is reference variable

Note: With primitive variable we can't call a method as well as we can't assign null literal.

int x = null; //Invalid

int y = 45;

y.m1(); //Invalid

***Based on declaration position Variables are divided into 4 types:***

a) Class Variables OR Static Field

b) Instance Variables OR Non Static Field

c) Local /Stack/temporary/Automatic Variable

d) Parameter Variables

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

**Program on Primitive variables**

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

package com.ravi.variables;

class Test

{

static int a = 100; //Class Variable OR Static Field

int b = 200; //Instance Variable OR Non Static Field

public void accept(int c)

{

int d = 400;

System.out.println("Class Variable:"+a);

System.out.println("Instance Variable "+b);

System.out.println("Parameter Variable "+c);

System.out.println("Local Variable "+d);

}

}

public class PrimitiveVariableDemo

{

public static void main(String[] args)

{

Test t1 = new Test();

t1.accept(300);

}

}

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

//Program on Reference variables

package com.ravi.variables;

import java.util.Scanner;

class Student

{

Student s = new Student(); //Non static Field

static Scanner sc = new Scanner(System.in);//static field

}

public class ReferenceVariableDemo

{

public static void main(String[] args)

{

Student st1 = new Student(); //st1 is local variable

}

public static void accept(Student s) //s is a parameter variable

{

}

}

30-10-2024

**Object Oriented Programming (OOPs)**

**-----------------------------------------------------**

**What is an Object?**

An object is a physical entity which exist in the real world.

**Example :-** Pen, Car, Laptop, Mouse, Fan and so on

***An Object is having 3 characteristics***:

a) Identification of the Object (Name of the Object)

b) State of the Object (Data OR Properties OR Variable of Object)

c) Behavior of the Object (Functionality of the Object)

* OOP is a technique through which we can design or develop the programs using class and object.
* Writing programs on real life objects is known as Object Oriented Programming.
* Here in OOP we concentrate on objects rather than function/method.

**Advantages of OOP**

1) Modularity (Dividing the bigger task into smaller task)

2) Reusability (We can reuse the component so many times)

3) Flexibility (Easy to maintain [By using interface])

**Features of OOP**

1) Class

2) Object

3) Abstraction

4) Encapsulation

5) Inheritance

6) Polymorphism

**What is a class?**

A class is model/blueprint/template/prototype for creating the object.

A class is a logical entity which does not take any memory.

A class is a user-defined data type which contains data member and member function.

public class Employee

{

Employee Data (Properties)

**+**

Employee behavior (Function/Method)

}

**A CLASS IS A COMPONENT WHICH IS USED TO DEFINE OBJECT PROPERTIES AND OBJECT BEHAVIOR.**

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

**How to initialize the object properties using reference variable through OOP.**

**2 files**

---Student.java----

package com.ravi.oop\_demo;

//BLC

public class Student

{

String name; //Non static variable

double height; //Non static variable

int age; //Non static variable

public void talk()

{

System.out.println("Hello Everyone, My name is: " +name);

System.out.println("My age is: " +age);

System.out.println("and my height is: " +height);

}

public void writeExam()

{

System.out.println("Every Saturday, I am writing the exam ");

}

}

---StudentDemo.java---

package com.ravi.oop\_demo;

//ELC

public class StudentDemo

{

public static void main(String[] args)

{

Student raj = new Student();

//Initializing the object properties

raj.name = "Raj Gourav";

raj.age = 21;

raj.height = 5.8;

//Calling the behavior

raj.talk(); raj.writeExam();

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

Student priya = new Student();

//Initializing the object properties

priya.name = "Priya";

priya.age = 20;

priya.height = 5.6;

//Calling the behavior

priya.talk(); priya.writeExam();

}

}

**Steps for creating Object Orineted Programming**

Step 1 :- Create the Object based on the BLC class inside ELC class.

[main method]

Step 2 :- Define all the object properties and behavior inside the BLC class

based on your imagination.

Step 3 :- Initialize all the object properties with user friendly value by

using reference variable.

step 4 :- call the behavior (calling the methods)

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

01-11-2024

----------

Instance Variable OR Non Static Field :

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

It is a class Level Variable.

If a non static variable is declared inside a class and outside of the method then it is called Instance Variable OR non static Field.

An instance variable life starts at the time of creating the object that means without an object we can't think abount instance variable.

public class Test

{

int x = 100;

public static void main(String[] args)

{

System.out.println("x value is :"+x); //error

}

}

The scope of an instance variable is within the same class as well as outside of the class but It depends upon the access modifier we have applied on the instance variable.

Instance variables are represented as an Object properties so whenever we want to represent Object level data/variable/properties then we should instance/non static variable.

public class Employee

{

int employeeId; //Instance Variable OR Non Static field

}

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

Initializing the Object properties using Method without parameter (Using Scanner class) :

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

2 files :

----------

Customer.java

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

package com.ravi.oop;

import java.util.Scanner;

//BLC

public class Customer

{

int customerId;

String customerName;

double customerBill;

//Initialize the Object Properties

public void setCustomerData()

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter Customer Id :");

customerId = sc.nextInt();

System.out.print("Enter Customer Name :");

customerName = sc.nextLine();

customerName = sc.nextLine();

System.out.print("Enter Customer Bill :");

customerBill = sc.nextDouble();

sc.close();

}

//Printing the Object Properties

public void getCustomerData()

{

System.out.println("Customer Id is :"+customerId);

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

System.out.println("Customer Bill is :"+customerBill);

}

}

CustomerDemo.java

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

package com.ravi.oop;

public class CustomerDemo {

public static void main(String[] args)

{

Customer scott = new Customer();

scott.setCustomerData();

scott.getCustomerData();

}

}

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

Initializing the object properties using parameter variable :

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

2 files :

---------

Employee.java

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

package com.ravi.oop;

public class Employee

{

int employeeId;

String employeeName;

public void setEmployeeData(int id, String name)

{

employeeId = id;

employeeName = name;

}

public void getEmployeeData()

{

System.out.println("Employee id is :"+employeeId);

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

}

}

EmployeeDemo.java

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

package com.ravi.oop;

public class EmployeeDemo {

public static void main(String[] args)

{

Employee john = new Employee();

john.setEmployeeData(111, "John");;

john.getEmployeeData();

Employee martin = new Employee();

martin.setEmployeeData(222, "Martin");

martin.getEmployeeData();

}

}

Note : Upto here, We know total 3 ways to initialize the object

properties which are as follows :

1) Using Object reference(raj.rollNumber = 122)

2) Using Method without parameter (Scanner class)

3) Using Method with Parameter (Park Story)

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

Constructor [Introduction]

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

If the name of the class and name of the method both are exactly same and It does not contain any return type then it is called Constructor.

Example :

-----------

public class Test

{

public Test() //Constructor

{

}

}

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

In java, Whenever we write a class and If we don't write any kind of constructor then automatically compiler will add one

default no argument constructor in the class at the time of compilation.

Demo.java

-----------

public class Demo

{

//Programmer has not provided any constructor

}

javac Demo.java (Compilation)

[After compilation Demo.class file will be generated]

Demo.class

----------

public class Demo

{

public Demo() //Default No Argument Constructor [Added by

{ java compiler]

}

}

\*Every java class must have at-least one constructor either

explicitly written by user OR implicitly added by java compiler.

The access modifier of default no argument constructor [added by java compiler] will depend upon class access modifier that means, If the class is public then default no argument constructor will also be public otherwise not.

Example :

---------

public class Test

{

}

javac Test.java [java compiler will compile and add default

constructor]

javap Test.class [To see the Constructor added by java compiler]

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

02-11-2024

-----------

Why compiler is adding default constructor to our class :

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

We have 2 reasons that why compiler is adding default constructor :

1) Without default constructor, Object creation is not possible in java by using new keyword, if the class does not contain user-defined constructor.

2) As we know only class level variables are having default values so, default constructor will initialize all the instance variables with default values with the help of new keyword.

Data type - Default value

byte - 0

short - 0

int - 0

long - 0

float - 0.0

double - 0.0

char - (space) '\u0000'

boolean - false

String - null

Object - null (For any class i.e reference variable the default value is null)

Program that describes new keyword is repsonsible to allocate

the default values for non static variables :

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

2 files :

---------

Student.java

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

package com.ravi.oop;

public class Student

{

int rollNumber;

String studentName;

double studentFees;

public void getStudentData()

{

System.out.println("Roll Number is :"+rollNumber);

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

System.out.println("Student Fees is:"+studentFees);

}

}

StudentDemo.java

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

package com.ravi.oop;

public class StudentDemo

{

public static void main(String[] args)

{

Student raj = new Student();

raj.getStudentData();

}

}

Here we will get output as default values i.e 0 , null and 0.0

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

How to initialize the object properties with parameter variable as per our requirement :

2 files :

-----------

Employee.java

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

package com.ravi.oop;

public class Employee

{

int employeeNumber;

String employeeName;

double employeeSalary;

char employeeGrade;

public void setEmployeeData(int eno, String ename, double salary)

{

employeeNumber = eno;

employeeName = ename;

employeeSalary = salary;

}

public void getEmployeeData()

{

System.out.println("Employee Id is :"+employeeNumber);

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

System.out.println("Employee Salary is :"+employeeSalary);

System.out.println("Employee Grade is :"+employeeGrade);

}

public void calculateEmployeeGrade()

{

if(employeeSalary >=75000)

{

employeeGrade = 'A';

}

else if(employeeSalary >=50000)

{

employeeGrade = 'B';

}

else if(employeeSalary >=30000)

{

employeeGrade = 'C';

}

else

{

employeeGrade = 'D';

}

}

}

EmployeeDemo.java

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

package com.ravi.oop;

public class EmployeeDemo

{

public static void main(String[] args)

{

Employee scott = new Employee();

scott.setEmployeeData(101, "Scott", 29000);

scott.calculateEmployeeGrade();

scott.getEmployeeData();

}

}

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

12-11-2024

-----------

Role of instance variable while creating the Object :

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

Whenever we create an objet in java, a separate copy of all the instance variables will be created with each and every object.

package com.ravi.oop;

public class Test

{

int x = 10;

public static void main(String[] args)

{

Test t1 = new Test();

Test t2 = new Test();

++t1.x;

--t2.x;

System.out.println(t1.x); //11

System.out.println(t2.x); //9

}

}

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

What is a static field ?

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

It is a class level variable.

If a variable is declared with static modifier inside a class then it is called class variable OR static field.

A static field variable will be automatically initialized with default values and memory will be allocated (even the variable is final) AT THE TIME OF LOADING THE CLASS INTO JVM MEMORY.

In order to access the static member, we need not to create an object, here class name is required.

Role of static field in Object creation :

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

Whenever we create an object then a single copy of static filed is created and the same single copy is sharable by all the objects so, if we make any changes through one object, it will reflect to all the objects as shown in the program.

package com.ravi.oop;

public class Demo

{

static int x = 100; //102

public static void main(String[] args)

{

Demo d1 = new Demo();

Demo d2 = new Demo();

++d1.x; ++d2.x;

System.out.println(d1.x);

System.out.println(d2.x);

}

}

So the Conclusion is :

Instance Variable = Multiple Copies with each ane every Object

Static Variable = Single Copy for all the objects.

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

\*\*What is Data Hiding ?

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

Data hiding is nothing but declaring our data members with private access modifier so our data will not be accessible from outer world that means no one can access our data directly from outside of the class.

\*We should provide the accessibility of our data through methods so we can perform VALIDATION ON DATA which are coming from outer world.

2 files :

----------

Customer.java

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

package com.ravi.data\_hiding;

public class Customer

{

private double balance = 10000; //Data hiding

public void deposit(double amount)

{

//Data Validation

if(amount <=0)

{

System.err.println("Amount can't be deposited");

}

else

{

this.balance = this.balance + amount;

System.out.println("Balance after deposit is :"+this.balance);

}

}

public void withdraw(double amount)

{

if(this.balance < amount)

{

System.err.println("Insufficient balance");

}

else

{

this.balance = this.balance - amount;

System.out.println("Balance after Withdraw is :"+this.balance);

}

}

}

BankApplication.java

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

package com.ravi.data\_hiding;

public class BankApplication

{

public static void main(String[] args)

{

Customer raj = new Customer();

raj.deposit(5000);

raj.withdraw(2000);

}

}

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

13-11-2024

----------

How to print Object properties (Instance variable) using toString() method :

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

If we want to print our object properties (Instance Variables) then we should generate(override) toString() method in our class from Object class.

Now with the help of toString() method we need not to write any display kind of method to print the object properties i.e instance variable.

In order to generate the toString() method we need to follow the steps

Right click on the program -> source -> generate toString()

In order to call this toString() method, we need to print the corresponding object reference by using System.out.println() statement.

Manager m = new Manager();

System.out.println(m); //Calling toString() method of Manager class

Employee e = new Employee();

System.out.println(e); //Calling toString() method of Employee class.

2 files :

----------

Manager.java

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

package com.ravi.to\_string;

public class Manager

{

private int managerId;

private String managerName;

public void setManagerData(int managerId, String managerName) {

this.managerId = managerId;

this.managerName = managerName;

}

@Override

public String toString()

{

return "Manager [managerId=" + managerId + ", managerName=" + managerName + "]";

}

}

ManagerDemo.java

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

package com.ravi.to\_string;

public class ManagerDemo

{

public static void main(String[] args)

{

Manager scott = new Manager();

scott.setManagerData(111, "Mr. Scott");

System.out.println(scott);

}

}

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

Constructor :

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

What is the advantage of writing constructor in our class ?

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

If we don't write a constructor in our program then variable initialization and variable re-initialization both are done in two different lines.

If we write constructor in our program then variable initialization and variable re-initialization both are done in the same line i.e at the time of Object creation. [13-NOV]

With Constructor approach, we need not to depend on method to re-initialize our instance variable with user value.

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

Defination of Constructor :

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

It is used to construct the object so it is called Constructor.

If the name of the class and name of the method both are exactly same and It should not contain any return type then it is called constructor.

The main purpose of constructor to initialize the object properties (Instance Variables) with user-defined value.

Every class must contain at-least one constructor either implicitly added by compiler or explicitly written by user.

Every time we create an object in java by using new keyword, at-least one constructor must be invoked.

A constructor never contain any return type including void also.

Example :

package com.ravi.command;

class Student

{

public void Student() //Method

{

System.out.println("I am Method");

}

public static void main(String [] args)

{

Student s1 = new Student();

s1.Student();

}

}

A constructor may contain return keyword but not return keyword with value.

package com.ravi.command;

class Student

{

public Student()

{

System.out.println("I am Constructor");

return;

}

public static void main(String [] args)

{

Student s1 = new Student();

}

}

A constructor is automatically called and executed at the time of creating the object.

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

Types of Constructor :

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

We have 3 types of Constructor :

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

1) default no argument constructor (Added by compiler)

2) Non parameterized OR No Argument OR Zero Argument Constructor

(Added by user for default value initialization)

3) Parameterized constructor

default no argumenet constructor :

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

Whenever we write a class and if we don't write any type of constructor then automatically one default constructor is added by the compiler.

The access modifier of default constaructor would be same as class access modifier.

It does not accept any parameter.

Example.java

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

public class Example

{

}

javac Example.java

Example.class

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

public class Example

{

public Example() //default constructor

{

}

}

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

2) No Argument OR Parameter-less OR Non parameterized OR Zero

argument Constructor.

If a user defines a constructor inside a class without argument then it is called no argument constructor.

No argument constructor and default constructor, both look like same the only difference is, default constructor means added by compiler and no argument constructor means written by user.

public class Student

{

private int rollNumber;

private String studentName;

public Student() //No Argument Constructor

{

rollNumber = 111;

studentName = "Raj";

}

}

No argument constructor is not recommended to initialize our object properties because due to no argument constructor all the object properties will be initialized with SAME VALUE as shown in the program. It is mainly used to initialized our object properties with default value.

2 files :

---------

Person.java

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

package com.ravi.no\_args;

public class Person {

private int personId;

private String personName;

public Person()

{

personId = 111;

personName = "Scott";

}

@Override

public String toString() {

return "Person [personId=" + personId + ", personName=" + personName + "]";

}

}

NoArgumentConstructor.java

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

package com.ravi.no\_args;

public class NoArgumentConstructor {

public static void main(String[] args)

{

Person scott = new Person();

System.out.println(scott);

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

Person smith = new Person();

System.out.println(smith);

}

}

In the above program smith and scott both objects are initialized with same data so not a recommened way.

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

Parameterized Constructor :

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

If we pass one or more argument to the constructor then it is called parameterized constructor.

By using parameterized constructor all the objects will be initialized with different values.

Example :

----------

public class Employee

{

int id;

String name;

public Employee(int id, String name)

{

this.id = id;

this.name = name;

}

}

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

Dog.java

---------

package com.ravi.first\_complete\_oop;

public class Dog

{

private String dogName;

private int dogAge;

private double dogHeight;

public Dog(String dogName, int dogAge, double dogHeight)

{

super();

this.dogName = dogName;

this.dogAge = dogAge;

this.dogHeight = dogHeight;

}

@Override

public String toString() {

return "Dog [dogName=" + dogName + ", dogAge=" + dogAge + ", dogHeight=" + dogHeight + "]";

}

}

ParameterizedConstructor.java

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

package com.ravi.first\_complete\_oop;

public class ParameterizedConstructor

{

public static void main(String[] args)

{

Dog tommy = new Dog("Tommy", 5, 3.5);

System.out.println(tommy);

Dog tiger = new Dog("Tiger",6,4.4);

System.out.println(tiger);

}

}

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

Upto here, by using following ways we can initialize the Object Properties :

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

1) By using Object reference

2) By using Method without parameter

3) At the time of variable declaration

4) Using No Argument Constructor

5) Using parameterized Constructor

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

How to write setter and getter methods :

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

public class Customer

{

private double customerBill;

//Initialize the customerBill using parameterized constructor

public Customer(double customerBill)

{

this.customerBill = customerBill;

}

//Writing setter to modify the existing customerBill

public void setCustomerBill(double customerBill)

{

this.customerBill = customerBill;

}

//Writing getter to retrieve the private data value outside of BLC class

public double getCustomerBill()

{

return this.customerBill;

}

}

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

FINAL CONCLUSION :

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

Parameterized Constructor : To initialize the Object properties with user values.

Setter : To modify the existing object data.[Only one data at a time] OR Writing Operation

Getter : To read/retrieve private data value outside of BLC class. [Reading Operation]

toString() : To print Object properties (Instance Variable)

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

\*\*\* What is Encapsulation

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

[Accessing our private data with public methods like setter and getter]

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

Binding the private data with its associated method in a single unit is called Encapsulation.

Encapsulation ensures that our private data (Object Properties) must be accessible via public methods like setter and getter.

It provides security because our data is private (Data Hiding) and it is only accessible via public methods WITH PROPER DATA VALIDATION.

In java, class is the example of encapsulation.

How to achieve encapsulation in a class :

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

In order to achieve encapsulation we should follow the following two

techniques :

1) Declare all the data members with private access modifiers (Data Hiding OR Data Security)

2) Write public methods to perform read(getter) and write(setter) operation on these private data like setter and getter.

Note : If we decalre all our data with private access modifier then it is called TIGHTLY ENCAPSULATED CLASS. On the other hand if we declare our data other then private access modifier then it is called Loosely Encapsulated class.

//Program on Encapsulation :

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

2 files :

---------

Employee.java

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

package com.ravi.setter\_getter;

public class Employee

{

private String employeeName;

private double employeeSalary;

public Employee(String employeeName, double employeeSalary) {

super();

this.employeeName = employeeName;

this.employeeSalary = employeeSalary;

}

@Override

public String toString() {

return "Employee [employeeName=" + employeeName + ", employeeSalary=" + employeeSalary + "]";

}

public String getEmployeeName()

{

return this.employeeName;

}

public void setEmployeeName(String employeeName)

{

this.employeeName = employeeName;

}

public double getEmployeeSalary()

{

return this.employeeSalary;

}

public void setEmployeeSalary(double employeeSalary)

{

this.employeeSalary = employeeSalary;

}

}

EmployeeDemo.java

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

package com.ravi.setter\_getter;

import java.util.Scanner;

public class EmployeeDemo {

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter Employee Name :");

String name = sc.nextLine();

System.out.print("Enter Employee Salary :");

double salary = sc.nextDouble();

Employee emp = new Employee(name,salary);

System.out.println(emp);

emp.setEmployeeSalary(emp.getEmployeeSalary()+10000);

System.out.println(emp);

//Based on the salary, I want to known whether employee

// is Developer, Designer OR Tester

double sal = emp.getEmployeeSalary();

if(sal>=75000)

{

System.out.println(emp.getEmployeeName()+" is a Developer");

}

else if(sal>=50000)

{

System.out.println(emp.getEmployeeName()+" is a Designer");

}

else

{

System.out.println(emp.getEmployeeName()+" is a Tester");

}

}

}

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

Method return type as a class :

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

While declaring a method in java, return type is compulsory.

As a method return type we have following options

1) void as a retutn type of the Method

2) Any primitive data type as a return type of the method.

3) Any class name/interface / enum / record we can take as a return type of the method.

Example 1 :

-----------

package com.nit;

public class Demo

{

public Demo m1() //Factory Method

{

return this; //Valid

OR

return null; //Valid

OR

return new Demo(); //Valid

}

}

Example 2 :

-----------

package com.nit;

public class Demo

{

int x;

public Demo(int x)

{

this.x = x;

}

public Demo m1()

{

return new Demo(8);

}

}

Note : Here the return value depends upon the available constructor in the class.

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

What is a Factory Method :

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

If a method return type is class name menas it is returning the Object of the class then it is called Factory Method.

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

//Program on Method return type as a class :

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

2 files :

---------

Book.java

----------

package com.ravi.factory\_method;

public class Book

{

private String authorName;

private String bookTitle;

public Book(String authorName, String bookTitle)

{

super();

this.authorName = authorName;

this.bookTitle = bookTitle;

}

@Override

public String toString() {

return "Book [authorName=" + authorName + ", bookTitle=" + bookTitle + "]";

}

public static Book getBookObject()

{

return new Book("James Gosling", "Java");

}

}

BookDemo.java

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

package com.ravi.factory\_method;

public class BookDemo {

public static void main(String[] args)

{

Book object = Book.getBookObject();

System.out.println(object);

}

}

In the avove program getBookObject() is providing only one Book object so it is not recommended because the main purpose of any method to provide re-usability as shown in the program below.

Employee.java

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

2 files :

----------

package com.ravi.factory\_method;

import java.util.Scanner;

public class Employee

{

private int employeeId;

private String employeeName;

private double employeeSalary;

public Employee(int employeeId, String employeeName, double employeeSalary) {

super();

this.employeeId = employeeId;

this.employeeName = employeeName;

this.employeeSalary = employeeSalary;

}

@Override

public String toString() {

return "Employee [employeeId=" + employeeId + ", employeeName=" + employeeName + ", employeeSalary="

+ employeeSalary + "]";

}

public static Employee getEmployeeObject()

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter Employee Id :");

int id = sc.nextInt();

System.out.print("Enter Employee Name :");

String name = sc.nextLine();

name = sc.nextLine();

System.out.print("Enter Employee Salary :");

double sal = sc.nextDouble();

return new Employee(id, name, sal);

}

}

EmployeeDemo.java

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

package com.ravi.factory\_method;

import java.util.Scanner;

public class EmployeeDemo {

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("How many Objects you want ?");

int noOfObj = sc.nextInt();

for(int i=1; i<=noOfObj; i++)

{

Employee object = Employee.getEmployeeObject();

System.out.println(object);

}

sc.close();

}

}

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

15-11-2024

-----------

2 files :

---------

Product.java

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

package com.ravi.factory\_method;

import java.util.Scanner;

public class Product

{

private int productId;

private String productName;

private double productPrice;

public Product(int productId, String productName, double productPrice)

{

super();

this.productId = productId;

this.productName = productName;

this.productPrice = productPrice;

}

@Override

public String toString()

{

return "Product [productId=" + productId + ", productName=" + productName + ", productPrice=" + productPrice

+ "]";

}

public static Product getProductObject()

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter Product Id :");

int id = sc.nextInt();

System.out.print("Enter Product Name :");

String pname = sc.nextLine();

pname = sc.nextLine();

System.out.print("Enter Product Price :");

double price = sc.nextDouble();

return new Product(id, pname, price);

}

}

ProductDemo.java

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

package com.ravi.factory\_method;

import java.util.Scanner;

public class ProductDemo {

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("How many product object :");

int noOfObj = sc.nextInt();

for(int i=1; i<=noOfObj; i++)

{

Product object = Product.getProductObject();

System.out.println(object);

}

sc.close();

}

}

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

What is Shallow and Deep copy in java :

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

Shallow Copy :

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

In Shallow copy, Only one Object will be created but the same object will be refered by multiple reference variables.

If we modify the object properties by any of the reference variable then original object will be modified as shown in the program.-

2 files :

---------

Laptop.java

-----------

package com.ravi.shallow\_copy;

public class Laptop

{

private String laptopBrand;

private double laptopPrice;

public Laptop(String laptopBrand, double laptopPrice) {

super();

this.laptopBrand = laptopBrand;

this.laptopPrice = laptopPrice;

}

@Override

public String toString() {

return "Laptop [laptopBrand=" + laptopBrand + ", laptopPrice=" + laptopPrice + "]";

}

public String getLaptopBrand() {

return laptopBrand;

}

public void setLaptopBrand(String laptopBrand) {

this.laptopBrand = laptopBrand;

}

public double getLaptopPrice() {

return laptopPrice;

}

public void setLaptopPrice(double laptopPrice) {

this.laptopPrice = laptopPrice;

}

}

ShallowCopy.java

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

package com.ravi.shallow\_copy;

public class ShallowCopy {

public static void main(String[] args)

{

Laptop lap1 = new Laptop("HP", 86000);

Laptop lap2 = lap1;

System.out.println("Before Change...");

System.out.println(lap1);

System.out.println(lap2);

System.out.println("After Change...");

lap2.setLaptopBrand("Dell");

lap2.setLaptopPrice(90000);

System.out.println(lap1);

System.out.println(lap2);

}

}

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

Deep Copy :

-----------

In deep copy two different objects will be created, the 2nd object will copy the content of first object.

If we modify the object by using reference variable then only one object will be modified because both the objects are created in two different memory location as shown below.

Customer.java

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

package com.ravi.deep\_copy;

public class Customer

{

private double customerBill;

public Customer()

{

customerBill = 0.0;

}

public Customer(double customerBill)

{

super();

this.customerBill = customerBill;

}

@Override

public String toString() {

return "Customer [customerBill=" + customerBill + "]";

}

public double getCustomerBill() {

return customerBill;

}

public void setCustomerBill(double customerBill) {

this.customerBill = customerBill;

}

}

DeepCopy.java

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

package com.ravi.deep\_copy;

public class DeepCopy {

public static void main(String[] args)

{

Customer c1 = new Customer(12000);

Customer c2 = new Customer();

c2.setCustomerBill(c1.getCustomerBill());

System.out.println("Before Change...");

System.out.println(c1);

System.out.println(c2);

c1.setCustomerBill(15000);

System.out.println("After Change...");

System.out.println(c1);

System.out.println(c2);

}

}

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

\*\*\*Pass by Value :

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

Java does not support pointers so java only works with pass by value.

Pass by value means we are sending the copy of orginal data to the method.

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

package com.ravi.pass\_by\_value;

public class PassByValueDemo1

{

public static void main(String[] args)

{

int x = 10;

accept(x);

System.out.println(x);

}

public static void accept(int y)

{

y = 100;

}

}

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

package com.ravi.pass\_by\_value;

public class PassByValueDemo2 {

public static void main(String[] args)

{

int x = 10;

x = accept(x);

System.out.println(x);

}

public static int accept(int y)

{

y = 100;

return y;

}

}

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

package com.ravi.pass\_by\_value;

class Product

{

private double productPrice = 12000;

public void setProductPrice(double productPrice)

{

this.productPrice = productPrice;

}

@Override

public String toString() {

return "Product [productPrice=" + productPrice + "]";

}

}

public class PassByValueDemo3

{

public static void main(String[] args)

{

Product p1 = new Product(); //12000

accept(p1);

System.out.println(p1); //18000

}

public static void accept(Product prod)

{

prod.setProductPrice(18000);

}

}

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

package com.ravi.pass\_by\_value;

class Product

{

private double productPrice = 12000;

public void setProductPrice(double productPrice)

{

this.productPrice = productPrice;

}

@Override

public String toString() {

return "Product [productPrice=" + productPrice + "]";

}

}

public class PassByValueDemo4

{

public static void main(String[] args)

{

Product p1 = new Product(); //12000

accept(p1);

System.out.println(p1); //12000

}

public static void accept(Product prod)

{

prod = new Product(); //new another object

prod.setProductPrice(18000);

}

}

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

package com.ravi.pass\_by\_value;

public class PassByValueDemo5

{

public static void main(String[] args)

{

String str = new String("Hyd");

accept(str);

System.out.println(str); //Hyd

}

public static void accept(String s1)

{

s1 = "india";

}

}

Note : Here String object will not be modified hence we will get the output Hyd because Strings are immutable.

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

16-11-2024

-----------

What is Garbage Collector in java -----------------------------------

It is an automatic memory management technique in java.

In C++ language, A programmer is responsible to allocate as well as de-allocate the memory otherwise we will get OutOfMemoryError.

In java language, Programmer is only responsible to allocate the memory, Memory de-allocation is automatically done by garbage collector.

Garbage Collector is a daemon thread which is responsible to delete the objects from the HEAP Memory. Actually It scans the heap memory and identifying which objects are eligible for Garbage Collector.[THE OBJECTS WHICH DOES NOT CONTAIN ANY REFERENCES ONLY THOSE OBJECTS ARE ELIGIBLE FOR GC]

It internally uses an algorithm called Mark and Sweep algorithm to delete the un-used objects.

As a developer we can also explicitly call garbage collector by writing the following code

System.gc();

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

How many ways we can make an object eligible for Garbage Collector :

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

There are 3 ways we can make an object eligible for GC.

1) Assigning null literal to existing reference variable :

Employee e1 = new Employee(111,"Ravi");

e1 = null;

2) Creating an Object inside a method :

public void createObject()

{

Employee e2 = new Employee();

}

Here we are creating Employee object inside the method so, once the method execution is over then e2 will be deleted from the Stack Frame and the employee object will become eligible for GC.

3) Assigning new Object to the old existing reference variable:

Employee e3 = new Employee();

e3 = new Employee();

Earlier e3 variable was poting to Employee object after that a new Employee Object is created which is pointing to another memory location so the first object is eligible for GC.

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

Memory in java :

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

In java, whenever we create an object then Object and its content (properties and behavior) are stroed in a special memory called HEAP Memory. Garbage collector visits heap memory only.

All the local variables and parameters variables are executed in Stack Frame and available in Stack Memory.

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

18-11-2024

----------

HEAP AND STACK DIAGRAM :

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

HEAP and STACK Digram for CustomerDemo.java

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

class Customer

{

private String name;

private int id;

public Customer(String name , int id)

{

super();

this.name=name;

this.id=id;

}

public void setId(int id)

{

this.id=id;

}

public int getId()

{

return this.id;

}

}

public class CustomerDemo

{

public static void main(String[] args)

{

int val = 100;

Customer c = new Customer("Ravi",2);

m1(c);

//GC [Only one object 3000x eligible for GC]

System.out.println(c.getId());

}

public static void m1(Customer cust)

{

cust.setId(5);

cust = new Customer("Rahul",7);

cust.setId(9);

System.out.println(cust.getId());

}

}

//9 5

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

HEAP and STACK Diagram for Sample.java

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

public class Sample

{

private Integer i1 = 900;

public static void main(String[] args)

{

Sample s1 = new Sample();

Sample s2 = new Sample();

Sample s3 = modify(s2);

s1 = null;

//GC [4 objects 1000x, 2000x, 5000x and 6000x]

System.out.println(s2.i1);

}

public static Sample modify(Sample s)

{

s.i1=9;

s = new Sample();

s.i1= 20;

System.out.println(s.i1);

s=null;

return s;

}

}

//20 9

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

HEAP and STACK Diagram for Test.java

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

public class Test

{

Test t;

int val;

public Test(int val)

{

this.val = val;

}

public Test(int val, Test t)

{

this.val = val;

this.t = t;

}

public static void main(String[] args)

{

Test t1 = new Test(100);

Test t2 = new Test(200,t1);

Test t3 = new Test(300,t1);

Test t4 = new Test(400,t2);

t2.t = t3;

t3.t = t4;

t1.t = t2.t;

t2.t = t4.t;

System.out.println(t1.t.val);

System.out.println(t2.t.val);

System.out.println(t3.t.val);

System.out.println(t4.t.val);

}

}

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