javaravishanker@gmail.com

10-OCT-23

---

A language is a communication media.

Any language contains two important things

1) Syntax (Rules)

2) Semantics (Structure OR Meaning)

English langugae translation :

---

Subject + verb + Object (Syntax)

He is a boy. (Valid)

He is a box. (Invalid)

Example of Progamming Language :

---

```
int a = 10;
int b = 0;
int c = a/b;   //Exception
```

Note :-

Syntax of the programming language is taken care by compiler.

compiler generates errors if a user does not follow the syntax of the programming language.

Semantics is taken care by our runtime Environment. It generates Exception if a user does not follow the semantics.

Statically(Strongly) typed language :-

---

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

In these languages we can hold same kind of value during the execution of the program.

Ex:- C,C++,Core Java, C#

Dynamically(Loosly) typed language :-

---

The languages where data type is not compulsory and it is optional before initialization of a variable then it is called dynamically typed language.

In these languages we can hold different kind of value during the execution of the program.

Ex:- Visual Basic, Javascript, Python

---

12-Oct-23

---

Flavors Of JAVA language :

---

1) JSE (Java Standard Edition) ->J2SE -> Core Java

2) JEE (Java Enterprise Edition) -> J2EE -> Advanced Java

3) JME (Java Micro Edition) -> J2ME -> Android Application

4) Java FX [out dated by using we can create GUI applications]

---

What is the difference between stand-alone programs and web-related programs?

Standalone Application

---

If the creation(development), compilation and execution of the program, everthing is done in a single system then it is called stand-alone program.

Eg:- C, C++, Java, C# and so on.

Stand alone programs are also known as Software OR Desktop application.

Web - related Application :-

---

If creation of the program, compilation of the program and execution of the program, Everything is done on different places then it is called web related program.

Eg:- Advanced Java, PHP, ASP.NET, Python

Web related programs are also known as websites or web application.

---

What is a function :-

---

A function is a self defined block for any general purpose, calculation or printing some data.

The major benefits with function are :-

---

1) Modularity :- Dividing the bigger modules into number of smaller modules where each module will perform its independent task.

2) Easy understanding :- Once we divide the bigger task into number of smaller tasks then it is easy to understand the entire code.

3) Reusability :- We can reuse a particular module so many number of times so It enhances the reusability nature.

Note :- In java we always reuse our classes.(WORA)

4) Easy Debugging :- Debugging means finding the errors, With function It is easy to find out the errors because each module is independent with another module.

Why we pass parameter to a function :-

---

We pass parameter to a function for providing more information regrading the function.

Eg:-

```
public void deposit(int amount)
{
}

public void sleep(int hours)
{
}
```

---

13-Oct-23

---

Why functions are called method in java?

---

In C++ there is a facility to write a function inside the class as well as outside of the class by using :: (Scope resolution Operator), But in java all the functions must be declared inside the class only.

That is the reason member functions are called method in java.

Variable -->  Field

function ---> Method

---

The role of compiler :

---

a) Compilers are used to check the syntax.

b) It also check the compatibility issues(LHS = RHS)

c) It converts the source code into machine code.

Some imporatnt points to remember :

---

a) Java programs must be saved having extension .java

b) java compiler(javac) is used to compile our code.

c) After successful compilation we are getting .class file (bytecode)

d) This .class file we submit to JVM for execution prupose (for executing my java code)

JVM :- It stands for Java Virtual Machine. It is a software in the form of interpreter written in 'C' language.

Every browser contains JVM, Those browsers are known as JEB (Java Enabled Browsers) browsers.

---

Why java language is so popular in the It industry :-

---

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.

Where as on the other hand 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.

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

---

What is the difference between the 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 machine understandable format (Operating System format).

---

14-Oct-23

---

History of java :

---

First Name of Java : OAK (In the year 1991 which is a tree name)

Project Name :- Green Project

Inventor of Java : - James Gosling and his friends

Official Symbol :- Coffee CUP

Java :- Island (Indonesia)

Birthday :- 23rd January 1996 [JDK 1.0]

---

Comments in java :

---

Comments are used to increase the readability of the program. It is ignored by the compiler.

In java we have 3 types of comments

1) Single line Comment (//)

2) Multiline Comment (/* ------------------------------ */)

3) Documentation Comment (/** ------------------------- */)

```
/**
Name of the Project : Online Shopping
Date created :- 12-12-2021
Last Modified - 16-01-2022
Author :- Ravishankar
Modules : -  10 Modules
*/
```

---

Write a program in java to display welcome message :

---

```
public class Welcome
{
	public static void main(String[] args)
	{
		System.out.println("Hello World!");
	}

}
```

Description of main() method :

---

public :-

---

public is an access modifier in java. The main method must be declared as public otherwise JVM cannot execute our main method or in other words JVM can't enter inside the main method for execution of the program.

If main method is not declared as public then program will compile but it will not be executed by JVM.

Note :- From java compiler point of view there is no rule to declare our methods as public.

static :-

---

In java our main method is declared as static so we need not to create an object to call the main method.

If a static method is declared in the same class then from main method we can directly call that method, here object is not required.

If the static method is declared in another class then we can call that static method with the help of class name, here also object is not required.

If we don't declare the main method as static method then our program will compile but it will not be executed by JVM.

---

Points to remember :

---

1) Non-static method, we cannot call directly from main

method(Object is required)

2) static method declared in the same class, we can directly call from main method

3) static method declared in another class, we can call with the help of class name.

```
class Labclass
{
   public static void main(String [] args)
   {
     Integer.parseInt(String x); //"123" --------> 123
   }
}
```

---

16-Oct-23

---

void :-

---

It is a keyword in Java. It means no return type. Whenever we define any method in java and if we don't want to return any kind of value from that particular method then we should write void before the name of the method.

Eg:

```
public void input()
{
}
```

```
public int accept()
{
   return 15;
}
```

Note :- In the main method if we don't write void or any other kind of return type then it will generate a compilation error.

In java whenever we define a method then compulsory we should define return type of method.(Syntax rule).

---

main() :-

---

It is a userdefined function/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.

---

Command Line Argument :-

---

Whenever we pass an argument/parameter to the main method then it is called Command Line Argument.

The argument inside the main method is String because String is a alpha-numeric collection of character so, It can accept numbers,decimals, characters, combination of number and character.

That is the reason java software people has provided String as a parameter inside the main method.(More Wider scope to accept the values)

---

System.out.println() :-

---

It is an output statement in java, By using System.out.println() statement we can print anything on the console.

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

In System.out, .(dot) is a member access operator. It is called as period. It is used to access the member of the class.

In java we have print() and println() both methos are available, print() method will print the data and keep the cursor in the same line where as println() method will print the data and move the cursor to the next line.

---

WAP in java to add two numbers.

---

```
//WAP in java to add two numbers
public class Addition
{
	public static void main(String[] args)
	{
		int x = 12;
		int y = 36;
		int z = x + y;
		System.out.println(z);
	}
}
```

Note :- In the above program the output is not userfriendly.

---

//WAP in java to add two numbers with user-friendly message

```
public class Sum
{
	public static void main(String[] args)
	{
		int x = 12;
		int y = 12;
		System.out.println("The sum is :"+x+y); //1212
		System.out.println(+x+y);//24
		System.out.println(""+x+y);//1212
		System.out.println("The sum is :"+(x+y));//24
	}
}
```

---

```
//Interview Question
public class IQ
{
	public static void main(String[] args)
	{
		String str = 90 + 12 + "NIT" + 40 + 40 + 40 ;

		System.out.println(str);
	}
}
```

---

Command Line Argument :

---

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

By using Command Line Argument we can pass some value at runtime.

The advantage of command line argument is "Single time compilation and number of times execution".

---

```
//WAP in java to accept the value from command line argument
public class Command
{
	public static void main(String a[])
	{
		System.out.println(a[0]);
	}
}
```

javac Command.java (Compilation)

java Command Virat Rohit (Execution)

At the time of execution we can pass some value, here we are passing Virat and Rohit, Virat is in 0th position where as Rohit is in 1st position so it will print Virat.

---

```
//WAP in java to accept the full name from command line argument
public class Command
{
	public static void main(String a[])
	{
```

```
                System.out.println(a[0]);
        }
}
```

javac Command.java (Compilation)

java Command "Virat Kohli" (Execution)

Here it will print Virat Kohli

---

//WAP in java to add two numebers by using command line argument

```
public class CommandAdd
{
        public static void main(String[] args)
        {
                System.out.println(args[0] + args[1]);
        }
}
```

javac Command.java (Compilation)

java Command 12 34 (Execution)

Here it will 1234

---

How to convert a String value to integer :

---

If we want to convert any String value into integer then java software people has provided a predefined class called Integer available in java.lang package, this class contains a predefined static method parseInt(String x) through which we can convert any String value into integer.

---

//WAP in java to add two number using Command Line Argument

```
public class CommandAddition
{
  public static void main(String [] x)
  {
      //converting String to integer
      int a =  Integer.parseInt(x[0]);
      int b =  Integer.parseInt(x[1]);
      System.out.println("The Sum is :"+(a+b));
  }
}
```

---

//WAP in java to re-use the classes and perform addition and subtraction.

This package contains 3 files :

---

Addition.java

---

```
package com.ravi.parse_int;

public class Addition
{
  public static int doSum(int x, int y)
  {
            int z = x + y;
            return z;
```

```
 }
}
```

Subtraction.java

---

```
package com.ravi.parse_int;

public class Subtraction
{
  public static int doSub(int x, int y)
  {
            int z = x - y;
            return z;
  }
}
```

Calculate.java

---

```
package com.ravi.parse_int;

public class Calculate
{
        public static void main(String[] args)
        {
                int doSum = Addition.doSum(100, 200);
                System.out.println("Sum is :"+doSum);

                int doSub = Subtraction.doSub(500,400);
```

```
                System.out.println("Sub is :"+doSub);



        }

}
```

---

//Program on Command Line Argument to find out the square of the number.

FindingSquare.java

---

```
package com.ravi;

public class FindingSquare
{
        public static void main(String[] args)
        {
         //Converting String value into integer
         int i = Integer.parseInt(args[0]);
   System.out.println("Square of "+i+" is :"+(i*i));


        }

}
```

Note :- From the command line argument if we pass any String value

like four then parseInt(String s) method is unable to

convert this value four into numeric format hance it will

generate an exception i.e java.lang.NumberFormatException.

How to execute Command line program in Eclipse IDE :

---

Right click on the Program -> Run as -> Run configuration -> Arguments -> Program Argument (Pass some value accroding to your use ) -> Click on Run

---

What is a package :

---

-> A package is nothing but folder in windows.

-> A package must be written in lowercase only (com.nit.basic)

-> Package provides a good arrangement of our java programs so fast searching will become easy.

```
package com.nit.intro;
public class Test
{

}
```

In order to compile the above program we can use following command

```
javac  -d  . Test.java
```

The above command will create 3 folders i.e com, nit and intro

Inside the com folder, nit folder, inside nit folder, intro folder

will be there and inside thhis intro folder Test.class file will

be available.

---

19-10-2023

---

Naming convention in java language :

---

WE SHOULD COMPULSORY FOLLOW NAMING CONVENTION TO INCREASE THE READABILITY OF THE CODE.

1) How to write a class in java

---

While writing a class in java we should follow pascal naming conventation.

Java classes are treated as Noun

ThisIsExampleOfClass (Each word first letter is capital)

Example :

---

String

System

Integer

BufferedReader

DataInputStream

ClassNotFoundException

ArithmeticException

ArrayIndexOutOfBoundsException

2) How to write a method in java :

---

In order to write methods in java we need to follow camel case naming conventation.

thisIsExampleOfMethod()

Example:

---

read()

readLine()

toUpperCase()

charAt()

parseInt()

Methods are treated as Verb.

3) How to write variable(Fields) in java

---

In order to write variables in java we need to follow camel case naming convention.

rollNumber;

employeeName;

customerNumber;

customerBill;

4) How to write final variable(Field)

---

[Each character must be capital]

final double PI = 3.14;

final int A = 90;

5) How to write final and static variable

MAX_VALUE;

MIN_VALUE;

Each character must be capital and in between every word _ symbol should be there.

6) While writing the package name we should take all characters in smallletter.

Example :-

com.nit.basic;

com.nit.literal;

com.nit.operator;

---

Token :

---

A token is the smallest unit of the program that is identified by the compiler.

Every Java statements and expressions are created using tokens.

A token can be divided into 5 types

1) Keywords

2) Identifiers

3) Literals

4) Punctuators

5) Operators

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.

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.

Ex:-

```
class Fan
{
  int coil;
```

```
  void start()
  {
  }
}
```

Here Fan(Name of the class), coil (Name of the variable) and start(Name of the function) 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 :

---

Assigning some constant value to variable is called Literal.

Java supports 5 types of Literals :

1) Integral Literal Ex:- int x = 15;

2) Floating Point Literal Ex:- float x = 3.5f;

3) Character Literal Ex:- char ch = 'A';

4) Boolean Literal Ex:- boolean b = true;

5) String Literal Ex:- String x = "Ravi";

Note :- null is also a literal.

---

Integeral Literal

---

If any numeric literal is declared without decimal or fraction then it is called Integral Literal.

Example :- 90, 78, 123, 678

In integral literal we have 4 data types

byte (8 bits)

short(16 bits)

int (32 bits)

long (64 bits)

An integral literal we can represent in 4 different forms

a) Decimal (base 10)

b) Octal (base 8)

c) Hexadecimal(base 16)

d) Binary(Base 2) [It came from java 1.7v]

Decimal Literal :

---

In java any integral literal is represented by Decimal literal. Decimal Literal base is 10 so, it accepts 10 digits i.e 0 to 9.

Octal Literal :

---

If any integral literal starts with 0 (zero) then it will become

Octal literal. Octal Literal base is 8 so, it accepts 8 digits i.e 0 to 7.

Hexadecimal literal :

---

If any integral literal starts with 0X(zero capital x) OR 0x(zero small x) then it will become Hexadecimal literal. Hexadecimal Literal base is 16 so, it accepts 16 digits i.e 0 to 9 and A-F.

Binary Literal :

---

It came from java 1.7v. If any integral literal starts with 0B(zero capital B) or 0b(Zero small b) then it will become Binary literal.

Binary Literal base is 2 so, it accepts 2 digits i.e 0 and 1 only.

Note :- As a Java developer we can represent integral literal in different forms like decimal, octal, hexadecimal and binary BUT FOR OUTPUT PURPOSE JVM WILL ALWAYS PRODUCE THE RESULT IN DECIMAL FORM

ONLY.

---

20-10-2023

---

//Octal literal

```
public class Test1
{
	public static void main(String[] args)
	{
		int one = 01;
		int six = 06;
		int seven = 07;
		int eight = 010;
		int nine = 011;
    System.out.println("Octal 01 = "+one);
    System.out.println("Octal 06 = "+six);
		System.out.println("Octal 07 = "+seven);
		System.out.println("Octal 010 = "+eight);
		System.out.println("Octal 011   = "+nine);
	}
}
```

---

```
//Hexadecimal
public class Test2
{
public static void main(String[] args)
	{
		int i = 0x10;    //16
		int j = 0Xadd;   //2781
		System.out.println(i);
		System.out.println(j);
	}
}
```

---

```
//Binary Literal
public class Test3
{
public static void main(String[] args)
        {
                int i = 0b101;
                int j = 0B111;
                System.out.println(i); //5
                System.out.println(j); //7



        }
}
```

---

By default every integral literal is of type int only but we can specify explicitly as long type by suffixing with l (small l) OR L (Capital L).

According to industry standard L is more preferable because l (small l) looks like 1(digit 1).

There is no direct way to specify byte and short literals (B or S) explicitly. If we assign any integral literal to byte variable and if the value is within the range (-128 to 127) then it is automatically treated as byte literals.

If we assign integral literals to short and if the value is within the range (-32768 to 32767) then automatically it is treated as short literals.

---

```
/* 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;
                System.out.println(s);
        }
}
```

---

```
//Converting bigger type to smaller type
public class Test6
{
public static void main(String[] args)
```

```
        {
                short s = 128;
                byte b =  (byte) s;
                System.out.println(b);
        }
}
```

---

21-10-2023

---

```
public class Test7
{
 public static void main(String[] args)
{
                byte x = (byte) 127L;
                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);

 }
}
```

---

21-10-2023

---

Is java pure Object-Oriented language ?

---

No, Java is not a pure Object-Oriented language. In fact any language which accepts the primary data type like int, float, char is not a pure object oriented language hence java is also not a pure object oriented language.

If we remove all 8 primitive data types from java then Java will become pure object oriented language.

In java we have a concept called Wrapper classes through which we can convert the primary data types into corresponding Wrapper Object.

| Primary data types | | Corresponding Wrapper Object |
|---|---|---|
| byte | - | Byte |
| short | - | Short |
| int | - | Integer |
| long | - | Long |
| float | - | Float |
| double | - | Double |
| char | - | Character |
| boolean | - | Boolean |

All these wrapper classes are available in java.lang package.

---

```
//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);
    }
}
```

---

How to know the minimum and maximum value as well as size of integral literal data types:

---

Thses classes (Wrapper classe) are providing the static and final variables through which we can find out the minimum, maximum value as well as size of the data types

Example:- I want to find out the range and size of Byte class

Byte.MIN_VALUE = -128

Byte.MAX_VALUE = 127

Byte.SIZE = 8 (in bits format)

Here MIN_VALUE, MAX_VALUE and SIZE these are static and final variables available in these classes(Byte, Short, Integer and Long).

---

```
 //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);

	}
}
```

---

From java 7v onwards we have a facility to provide _ (underscore)

symbol with integral literal to enhance the readability of the code.

```
//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);
	}
}
```

The above program will compile and execute.

---

```
// 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
public class Test13
{
        public static void main(String[] args)
        {
                var x = 12;
                System.out.println(x);
        }
}
```

---

Floating point Literals :

---

Any integeral literal contains decimal or fraction then it is called Floating point literal.

Example :- 2.3, 8.9, 123.78

In floating point literals we have 2 data types

float (4 bytes)

double (8 bytes)

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

```
float f1 = 2.3;
```

Now, we have 3 solutions

```
float f1 = (float)2.3;
float f2 = 12.7f;
float f3 = 19.56F;
```

As we know by default every floating point literal is of type double only but still compilar has provided two flavors to represent double value explicitly to enhance the readability of the code.

```
double d1 = 2.3d;
    double d2 = 56.78D;
```

*While working with integral literal we had four types of representation i.e decimal, octal, hexadecimal and binary but in floating point literal we have only one flavor i.e decimal only.

*Integral literal we can assign to floating point literal but floating point litearl we cannot assign to integral literal

Floating point literal we can also represent in exponent form

double d1 = 15e2; (15 X 10 to the power 2)

---

```
public class Test
{
	public static void main(String[] args)
	{
		float f = 2.0; //error
		System.out.println(f);
	}
}
```

---

```
public class Test1
{
	public static void main(String[] args)
	{
		//float a = 1.0;
		float b = 15.29F;
		float c = 15.25f;
		float d = (float) 15.25;
		System.out.println(b +" : "+c+" : " +d);

	}
}
```

---

public class Test2

```
{
	public static void main(String[] args)
	{
		double d = 15.15;
		double e = 15.15d;
		double f = 15.15D;
		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 = 0X9.15; //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;
          float c = 1.5e3F;
          double d = 10;
          int e = 10.0;
          long f = 10D;
          int g = 10F;
          long l = 12.78F;
        }
}
-----------------------------------------------------------
//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);
        }
}
```

---

25-10-2023

---

Character Literal :

---

1) It is also known as char literal.

2) In char literal we have one data type i.e char data type which accepts 2 bytes (16 bits) of memory.

3) There are multiple ways to represent char literal as shown below

a) Single character enclosed with single quotes.

Ex:- char c = 'a';

b) We can assign integral literal to char data type to represent UNICODE values.

The older languages like C and C++ support ASCII Value whose range is 0-255 only;

The Java language supports UNICODE values where the range is 0- 65535.

c) Char literals we can also assign to integral data type to get the UNICODE value of that particular character.

d) Char literals we can also represent in UNICODE format where it must contain 4 digit hexadecimal number.

The format is '\uXXXX' [\u0000 to \uffff]

Note :- XXXX is hexadecimal number

e) A charcter starts with \ (Back slash) is called as escape sequence. Every Escape sequence is also char literal. Java supports the following escape sequences.

In java we have 8 escape sequences

a) \n -> Inserting a new line

b) \t -> For providing tab space

c) \r -> carriage return(move the cursor to the first line)

d) \b -> Inserting a Backspace

e) \f ->(Form feed) Inserts a form feed (For moving to next page)

f) \' -> single quotes

g) \" -> Double quotes

h) \\

---

```
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);



	}
}
-----------------------------------------------------------------
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 = 1;
                System.out.println("ch3 value is :"+ch3);
        }
}
```

---

```
public class Test4
{
        public static void main(String[] args)
        {
                char ch1 = 65535;
    System.out.println("ch1 value is :"+ch1);

                char ch2 = 0Xadd;
                System.out.println("ch2 value is :"+ch2);
        }
}
```

Note :- Here we will get the output as ? because the equivalent language translator for these particular characters are not available in my 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);
        System.out.println('A' + 'A');
  }
}
```

---

```
//Range of UNICODE Value (65535)
class Test6
{
        public static void main(String[] args)
        {
                char ch1 = 65535;
                System.out.println("ch value is :"+ch1);

                char ch2 = 65536;
                System.out.println("ch value is :"+ch2);
        }
}
```

The above program will generate compilation error because 65536 is out of the range.

---

```
//WAP in java to describe unicode representation of char in hexadecimal format
class Test7
{
        public static void main(String[] args)
        {
```

```
                char ch1 = '\u0001';
                System.out.println(ch1);

                char ch2 = '\uffff';
                System.out.println(ch2);

                char ch3 = '\u0041';
    System.out.println(ch3);

                char ch4 = '\u0061';
                System.out.println(ch4);
        }
}
-----------------------------------------------------------------
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(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
	}
}
-----------------------------------------------------------------
```

```
//Java Unicodes
public class Test12
{
	public static void main(String[] args)
	{
		System.out.println(" Java Unicodes\n");

		for (int i = 31; i < 126; i++)
		{
			char ch = (char)i; // Convert unicode to character
			String str = i + "  "+ ch;
			System.out.print(str + "\t\t");
			if ((i % 5) == 0) // Set 5 numbers per row
			System.out.println( );
		}
	}
}
```

---

26-10-2023

---

boolean Literal :

---

1) boolean literal contains only one data type i.e boolean data type which accepts 1 bit of memory and it has two states i.e true and false.

2) It takes one bit of memory i.e true or false as well as it also depends on JVM implemntation.

Example:-

```
    boolean isValid = true;
    boolean isEmpty = false;
```

3) Unlike C and C++, In java it is not possible to assign integreal literal to boolean data type.

```
     boolean b = 0; (Invalid in java but valid in C and C++)
     boolean c = 1; (Invalid in java but valid in C and C++)
```

4) We can't assign String value to boolean data type.

```
   boolean d = "true"; //here true is String literal not boolean, not possible
   boolean e = "false";//here false is String literal not boolean, not possible
```

---

```
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"; //error
        boolean y = "false";//error
        System.out.println(x);
     System.out.println(y);
        }
}
```

---

String Literal :

---

A string literal in Java is basically a sequence of characters. These characters can be anything like alphabets, numbers or symbols which are enclosed with double quotes. So we can say String is alpha-numeric collection of character.

How we can create String in Java :-

---

In java String can be created by using 3 ways :-

1) By using String Literal

```
String x = "Ravi";
```

2) By using new keyword

```
String y = new String("Hyderabad");
```

3) By using character array

```
char z[] = {'H','E','L','L','O'};
```

---

```
//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);


        }
}
```

---

4) Punctuators :

---

It is  also called separators.

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

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

---

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

---

Arithmetic Operator OR Binary Operator :

---

It is known as Arithmetic Operator OR Binary Operator because it works with minimum two operands.

Ex:- +, - , *, / and % (Modula Or Modulus Operator)

---

```
//Arithmetic Operator
// Addition operator to join two Strings working as String concatenation optr
public class Test1
{
	public static void main(String[] args)
	{
		String s1 = "Welcome to";
```

```
                String s2 = " Java ";
                String s3 = s1 + s2;
                System.out.println("String after concatenation :"+s3);

        }
}
```

---

How to read the value from the user/keyboard (Accepting the data from client)

---

In order to read the data from the client or keyboard, java software people has provided a predefined class called Scanner available in java.util package.

It is available from java 5v.

static variables of System class :

---

System is a predefined class which contains 3 static variables.

System.out :- It is used to print normal message on the screen.

System.err :- It is used to print error message on the screen.

System.in :- It is used to take input from the user.(Attaching the keyboard with System resource)

How to create the Object for Scanner class :

---

```
Scanner sc = new Scanner(System.in);  //Taking the input from the user
```

Scanner class provides various methods :

---

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

String  nextLine() :- Used to read complete line or multiple Words.

byte nextByte() :- Used to read byte value

short nextShort() :- Used to read short value

int nextInt() :- Used to read integer value

float nextFloat() :- Used to read float value

double nextDouble() :- Used to read double value

boolean nextBoolean() :- Used to read boolean value.

char next().charAt(0) :- Used to read a character

---

```
//WAP to read your name from the keyboard
import java.util.*;
public class Test2
{
	public static void main(String [] args)
	{
```

```
    Scanner sc = new Scanner(System.in);
            System.out.print("Enter your Name :");
            String name =  sc.next(); //will read single word
            System.out.println("Your name is :"+name);
          }
}
```

---

27-10-2023

---

//WAP to read your name from the keyboard

```
import java.util.Scanner;
public class ReadCompleteName
{
        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);
        }
}
```

---

//Reading Employee data

```
import java.util.Scanner;
public class EmployeeData
{
        public static void main(String[] args)
```

```
        {
                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(); //Buffer Problem
                name = sc.nextLine();

                System.out.println("Employee Id is :"+id);
                System.out.println("Employee Name is :"+name);
        }
}
-----------------------------------------------------------------
//Arithmetic Operator (+, -, *, / , %)
//Reverse of a 3 digit number
import java.util.*;
class Test3
{
        public static void main(String[] args)
        {
                System.out.print("Enter a three digit number :");
                Scanner sc = new Scanner(System.in);

                int num =  sc.nextInt(); //num = 567

                int rem = num % 10;  //rem = 7
```

```
        System.out.print("The Reverse is :"+rem); //The reverse is :765


        num = num /10;    //num = 56
                        rem = num % 10;  //rem = 6
        System.out.print(rem);


                        num = num/10; //num = 5
                        System.out.println(num);
                }
}
```

---

Unary Operator :

---

The operator which works upon single operand is called Unary Operator.In java we have so many unary operators which are as follows :

1) Unary minus operator (-)

2) Increment Operator (++)

3) Decrement Operator (--)

---

```
//*Unary Operators (Acts on only one operand)
//Unary minus Operator
class Test4
{
        public static void main(String[] args)
```

```
	{
		int x = 15;
		System.out.println(-x);
		System.out.println(-(-x));
	}
}
```

---

```
//Unary Operators
//Unary Pre increment Operator
class Test5
{
	public static void main(String[] args)
	{
		int x = 15;
		int y = ++x;    //First increment then assignment
		System.out.println(x+":"+y);
	}
}
```

---

```
//Unary Operators
//Unary Post increment Operator
class Test6
{
	public static void main(String[] args)
	{
		int x = 15;
		int y = x++; //First assignment then increment
		System.out.println(x+":"+y);
```

```
	}
}
------------------------------------------------------------
//Unary Operators
//Unary Pre increment Operator
class Test7
{
	public static void main(String[] args)
	{
		int x = 15;
		int y = ++15;   //error
		System.out.println(y);
	}
}
------------------------------------------------------------
//Unary Operators
//Unary Pre increment Operator
class Test8
{
	public static void main(String[] args)
	{
		int x = 15;
		int y = ++(++x); //error
		System.out.println(y);
	}
}
------------------------------------------------------------
```

```
//Unary Operators
//Unary post increment Operator
class Test9
{
	public static void main(String[] args)
	{
		int x = 15;
		System.out.println(++x + x++);
		System.out.println(x);
      System.out.println(".....");

		int y = 15;
		System.out.println(++y + ++y);
		System.out.println(y);
	}
}
```

---

Note :- Increment and decrement operator we can apply with any data type except boolean.

---

```
//Unary Operators
//Unary post increment Operator
class Test10
{
	public static void main(String[] args)
	{
		char ch ='A';
```

```
		ch++;
		System.out.println(ch);
	}
}
-------------------------------------------------------------
//Unary Operators
//Unary post increment Operator
class Test11
{
	public static void main(String[] args)
	{
		double d = 15.15;
		d++;
		System.out.println(d);
	}

}

-------------------------------------------------------------
//Unary Operators
//Unary Pre decrement Operator
class Test12
{
	public static void main(String[] args)
	{
		int x = 15;
		int y = --x; //First decrement then assignment
		System.out.println(x+":"+y);
```

```
	}
}
```

---

```
//Unary Operators
//Unary Post decrement Operator
class Test13
{
	public static void main(String[] args)
	{
		int x = 15;
		int y = x--;
		System.out.println(x+":"+y);
	}
}
```

---

Interview Question

---

Whenever we work with Arithmetic Operator or Unary minus operator, the minimum data type required is int, So after calculation of expression it is promoted to int type.

```
//IQ
class Test14
{
	public static void main(String args[])
	{
		byte i = 1;
		byte j = 1;
		byte k = i + j;  //error
```

```
                System.out.println(k);
        }
}
```

---

```
class Test15
{
        public static void main(String args[])
        {
                /*byte b = 6;
    b = b + 7;  //error
                System.out.println(b); */

 byte b = 6;
 b += 7;//short hand operator  b += 7 is equal to (b = b + 7)
 System.out.println(b);

        }
}
```

Note :- In the above program it generates error while working with Arithmetic Operator but when we change the operator from

Arithmetic to short hand operator then the expression result we can assign on byte data type.

---

```
class Test16
{
         public static void main(String args[])
        {
                byte b = 1;
```

```
                byte b1 = -b; //error
        System.out.print(b1);
         }
}
```

---

What is a local variable :

---

If a variable is declared inside a method  body(not as a method parameter) then it is called Local / Stack/ Temporary / Automatic variable.

Ex:-

```
public void input()
{
   int y = 12;
}
```

Here in the above example y is local variable.

Local variable we can't use outside of the function or method.

A local variable must be initialized before use otherwise we wiil get compilation error.

We can't use any access modifier on local variable except final.

Program

---

```
public class Test17
```

```
{
        public static void main(String [] args)
        {
         int x ; //must be initialized before use
         System.out.println(x);

    public int y = 100;//only final is acceptable
    System.out.println(y);
        }
}
```

Note :- In the above program we will get compilation error

---

Why we cannot use local variables outside of the method ?

---

In java all the methods are executed as a part of Stack Memory. Stack Memory works on the basis of LIFO (Last In First Out).

Once the method execution is over, local variables are also deleted from stack frame so we cannot use local variables outside of the method.(Diagram 27-OCT-23)

```
class StackMemory
{
        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()
	{
		int x = 100;
		System.out.println("I am m2 method!!!"+x);
	}
}
```

---

28-10-2023

---

```
//*Program on Assignment Operator
class Test18
{
	public static void main(String args[])
	 {
	int x = 5, y = 3;
	System.out.println("x = " + x);
	System.out.println("y = " + y);

	x %= y;		//short hand operator  x = x % y
	System.out.println("x = " + x);
		}
```

}

---

What is BLC class :

---

BLC stands for Business Logic class. A BLC class never contains main method. It is used to write the logic only.

What is ELC class :

---

ELC stands for Executable Logic class. An ELC class always contains main method. It is called ELC class because the execution of the program starts from ELC class.

Note :- WE SHOULD ALWAYS TAKE OUR JAVA CLASSES IN A SEPARATE FILE OTHERWISE THE RE-USABILITY OF THE CLASS IS NOT POSSIBLE.

---

Here we have 2 files :

---

Circle.java(BLC)

---

```
package com.ravi.blc_elc;
/*
 Find the area of circle. Accept the radius value from the user
 if the radius is zero or negative then return -1.
*/

//BLC
public class Circle
{
  public static double getAreaOfCircle(int radius)
```

```
  {
            if(radius <=0)
            {
                        return -1;
            }
            else
            {
                        final double PI = 3.14;
                        double areaOfCircle = PI * radius * radius;
                        return areaOfCircle;
            }
  }
}
```

AreaOfCircle.java(ELC)

---

```
package com.ravi.blc_elc;

import java.util.Scanner;

//ELC
public class AreaOfCircle
{
            public static void main(String[] args)
            {
                        Scanner sc = new Scanner(System.in);
                        System.out.print("Enter the radius of the Circle :");
                        int radius = sc.nextInt();
```

```
                double areaOfCircle = Circle.getAreaOfCircle(radius);
                System.out.println("Area of Circle is :"+areaOfCircle);
                sc.close();
        }

}
```

---

Relational Operator :-

---

These operators are used to compare the values. The return type is boolean. We have total 6 Ralational Operators.

1) > (Greater than)

2) < (Less than)

3) >= (Greater than or equal to)

4) <= (Less than or equal to)

5) == (double equal to)

6) != (Not equal to )

```
//*Program on relational operator(6 Operators)
class Test19
{
```

```
        public static void main(String args[])
        {
    int a = 10;
    int b = 20;
    System.out.println("a == b : " + (a == b) ); //false
    System.out.println("a != b : " + (a != b) );  //true
    System.out.println("a > b : " + (a > b) );   //false
    System.out.println("a < b : " + (a < b) );   //true
    System.out.println("b >= a : " + (b >= a) ); //true
    System.out.println("b <= a : " + (b <= a) ); //false
   }
}
```

---

If condition :

---

It is decision making statement. It is used to test a boolean expression. The expression must return boolean type.

```
//Program to check a number is 0 or +ve or -ve
import java.util.Scanner;
class Test20
{
        public static void main(String args[])
        {
                Scanner sc = new Scanner(System.in);
                System.out.print("Please enter a Number :");

                int num = sc.nextInt();
```

```
                if(num == 0)
                System.out.println("It is zero");

                else if(num>0)
                System.out.println(num+" is positive");
                else
                System.out.println(num+" is negative");

                sc.close(); //To close Scanner resource
        }
}
```

---

```
/*program to calculate telephone bill
For 100 free call rental = 360
For 101 - 250, 1 Rs per call
For 251 - unlimited , 1.2 Rs per call
*/
import java.util.*;
class Test21
{
public static void main(String args[])
        {
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter current Reading :");
                int curr_read = sc.nextInt();

                System.out.print("Enter Previous Reading :");
                int prev_read = sc.nextInt();
```

```
                int nc = curr_read - prev_read;         [curr_read > prev_read]
                System.out.println("Your Number of call for this month is :"+nc);



                double bill = 0.0;
                if (nc <=100)
                {
                        bill = 360;
                }
                else if(nc<=250)
                {
        bill = 360 + (nc-100)*1.0;
                }
      else if(nc >250)
                {
         bill = 360 + 150 + (nc-250)*1.2;
                }
       System.out.println("The bill is :"+bill);
          }
}
```

---

Nested if:

---

If an 'if condition' is placed inside another if condition then it is called Nested if.

In nested if condition, we have one outer if and one inner if condition, the inner if condition will only execute when outer if condition returns true.

```
if(condition)  //Outer if condition
```

```
{
  if(condition) //inner if condition
  {
  }
  else //inner else
  {
  }
}
else //outer else
{
}
```

---

```
//Nested if
//big among three number
class Test22
{
public static void main(String args[])
        {
                int a =15;
                int b =12;
                int c =18;

                int big=0;

                if(a>b)  //(Outer if condition)
                {
                        if(a>c)    //Nested If Block (inner if)
                                big=a;
```

```
                else
                        big=c;
        }
        else    //already confirmed b is greater than a
        {
                if(b>c)
                        big=b;
                else
                        big=c;
        }
    System.out.println("The big number is :"+big);
    }
}
```

Note :- In the above program to find out the biggest number among three number we need to take the help of nested if condition but the code becomes complex, to reduce the length of the code Logical Operator came into the picture.

---

Logical Operator :-

---

It is used to combine or join the multiple conditions into a single statement.

It is also known as short-Circuit logical operator.

In Java we have 3 logical Operators

1) &&  (AND Logical Operator)

2) ||  (OR Logical Operator)

3) !  (NOT Logical Operator)

&&  :- All the conditions must be true. if the first expression is false it will not check right side expressions.

||  :- Among multiple conditions, at least one condition must be true. if the first expression is true it will not check right side expressions.

!  :- It is an inverter, it makes true as a false and false as a true.

Note :- The && and || operator only works with boolean operand so the following code will not compile.

```
if(5 && 6)
{

}
```

---

```
//*Program on Logical Operator (AND, OR, Not Operator)
//Biggest number among 3 numbers

class Test23
{
public static void main(String args[])
        {
                java.util.Scanner sc = new java.util.Scanner(java.lang.System.in);
                System.out.print("Enter the value of a :");
```

```
		int a = sc.nextInt();
		System.out.print("Enter the value of b :");
		int b = sc.nextInt();
		System.out.print("Enter the value of c :");
		int c = sc.nextInt();

		int big =0;

		if(a>b && a>c)
			big = a;
		else if(b>a && b>c)
			big = b;
		else
			big = c;
		System.out.println("The big number is :"+big);
	}
}
```

---

```
//OR Operator (At least one condition must be true)
class Test24
{
public static void main(String args[])
	{
		int a=10;
		int b=5;
		int c=20;
		System.out.println(a>b || a<c); //true
		System.out.println(b>c || a>c); //false
```

```
        }
}
```

---

```
// !Operator (not Operator works like an Inverter)
class Test25
{
  public static void main(String args[])
        {
                System.out.println(!true);
        }
}
```

---

Boolean Operators :

---

Boolean Operators work with boolean values that is true and false. It is used to perform boolean logic upon two boolean expressions.

It is also known as non short circuit. There are two non short circuit logical operators.

& boolean AND operator (All condions must be true but if first expression is false still it will check all right side expressions)

| boolean OR operator (At least one condition must be true but if the first condition is true still it will check all right side expression )

---

```
//* Boolean Operators
/*
& boolean AND operator
| boolean OR operator
```

```
*/
//Works with boolean values

class Test26
{
	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 Test27
{
```

```
	public static void main(String[] args)
	{
		int z = 5;
		if(++z > 6 & ++z> 6)
		{
			z++;
		}
		System.out.println(z);
	}
}
```

---

30-OCT-23

---

Bitwise Operator :-

---

In order to work with binary bits java software people has provided Bitwise operator. It also contains 3 operators

& (Bitwise AND) :- Returns true if both the inputs are true.

| (Bitwise OR) :- Returns false if both the inputs are false

^ (Bitwise X-OR) :- Returns true if both the arguments are opposite to each other.

```
//Bitwise Operator
class Test28
{
```

```
        public static void main(String[] args)
        {
      System.out.println(true & true); //true
        System.out.println(false | true); //true
        System.out.println(true ^ true); //true



                System.out.println(6 & 7);  //6
                System.out.println(6 | 7);  //7
                System.out.println(6 ^ 7); //1
        }
}
```

---

Bitwise complement operator :

---

-> It will not work with boolean literal.

```
//Bitwise Complement Operator
public class Test29
{
   public static void main(String args[])
        {
                //System.out.println(~ true); Invalid
                System.out.println(~ -8);

   }
}
```

---

Ternary Operator OR Conditional Operator :

---

The ternary operator (? :) consists of three operands. It is used to evaluate boolean expressions. The operator decides which value will be assigned to the variable.It is used to reduced the size of if-else condition.

```
//Ternary Operator OR Conditional Operator
public class Test30
{
   public static void main(String args[])
        {
                int a = 60;
                int b = 59;
                int max = 0;

                max=(a>b)?a:b;  //Type casting
                System.out.println("Max number is :"+max);

   }
}
```

---

```
public class Test
{
   public static void main(String [] args)
        {
          char ch = 'A';
          float i = 12;
          System.out.println(false?i:ch); //65.0  //type casting
```

```
        System.out.println(true?ch:i);  //65.0 //type casting


      }


}
```

---

Member access Operator Or Dot Operator :

---

It is used to access the member of the class so whenever we want to call the member of the class (fields + methods) then we should use dot(.) operator.

We can directly call any static method and static variable from the main method with the help of class name , here object is not required as shown in the program below.

If static variable or static method is present in the same class where main method is availble then we can directly call but if the static variable and static method is available in another class then to call those static members of the class, class name is required.

```
class Welcome
{
  static int x = 100;


  public static void access()
  {
    System.out.println(x);
  }
}
public class Test
{
```

```
   public static void main(String [] args)
          {
            System.out.println(Welcome.x);
            Welcome.access();
          }

}
```

---

//* new Operator

This Operator is used to create Object. If the member of the class (field + method) is static, object is not required. we can directly call with the help of class name.

On the other hand if the member of the class (variables + method) is not declared as static then it is called non-static member Or instance member , to call the non-static member object is required.

Program :

---

```
class Welcome
{
  int x = 100;  //instance variable (Non-static field)

  public void access()  //instance method
  {
    System.out.println(x);
  }
}
public class Test
```

```
{
   public static void main(String [] args)
        {
                Welcome w = new Welcome();
                System.out.println(w.x);
                w.access();

        }

}
```

---

instanceof operator :-

1)This Operator will return true/false

2) It is used to check a reference variable is holding the particular/corresponding type of Object or not.

3) It is also a keyword.

4) In between the object reference and class name , we must have some kind of relation (assignment relation) otherwise we will get compilation error.

```
//* instanceof operator
public class Test
{
        public static void main(String[] args)
        {
```

```
                String str = "India";


                if(str instanceof String)
                {
                        System.out.println("str is pointing to String object");
                }


    Integer i = 45;
                if(i instanceof Integer)
                {
                        System.out.println("i is pointing to Integer object");
                }


    Double d = 90.67;
                if(d instanceof Number)  //IS-A relation between Double and Number
class
                {
                        System.out.println("d is pointing to Double object");
                }


        }
}
```

---

31-10-2023

---

Types of variables in java

---

-> Based on the data type we can define the variable into two

categories

a) Primitive type variables

b) Non-primitive type(Reference type) variable

-> Example of primitive type variables

int x = 15; [byte, short, int, long, float, double, char and boolean]

-> Example of Reference type variable

Customer c = new Customer();

Here 'c' is reference variable

Note :- Any variable if we declare with the help of class then it is called reference variable.

-> Based on the declaration position we can define a varible

into 4 categories

a) Instance variable OR Non-Static field

b) Class variable OR Static field

c) Parameter variable

d) Local variable [1 more flavour = block level variable]

Program on primitive variable :

---

```
package com.ravi.variable;
```

```
class Demo
{
	int x = 120; //Instance variable (non-static field)
	static int y = 200; //class variable (static field)

	public void acceptData(int z) //parameter variable
	{
		int localVar = 78;
		System.out.println("Parameter Variable :"+z);
		System.out.println("Local Variable :"+localVar);
	}

}

public class Primitive
{
	public static void main(String[] args)
	{
    System.out.println("Class Variable :"+Demo.y);
    Demo d1 = new Demo();
    System.out.println("Instance variable "+d1.x);
    d1.acceptData(999);
	}

}
```

---

Program on reference variable :

```
--------------------------------

package com.ravi.variable;


import java.util.Scanner;


class Employee

{

	String name = "Vijay"; //Instance variable

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


	public void getEmployeeName()

	{

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

	}




}


public class Reference

{

	public static void main(String[] args)

	{

	   Employee e1 = new Employee();  //local variable

	   getEmployeeData(e1);

	}


	public static void getEmployeeData(Employee emp) //parameter variable

	{
```

```
                emp.getEmployeeName();
        }


}
```

---

What is drawback of if condition :-

---

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 :-

---

In switch case dpending upon the parameter the appropriate case would be executed otherwise default would be executed.

In this approch we need not to check each and every case, if the appropriate case is available then directly it would be executed.

break keyword is optional here but we can use as per requirement. It will move the control outside of the body of the switch.

---

```
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.*;
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");
                }
   }
}
```

---

```
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().toLowerCase();

		switch(season)
		{
			case "summer" :
				System.out.println("It is summer Season!!");
			 break;

			 case "rainy" :
				System.out.println("It is Rainy Season!!");
			 break;
		}
	}
}
```

---

```
public class Test2
{
	public static void main(String[] args)
	{
		double val = 1;
		switch(val) //Error, can't pass long, float and double
		{
```

```
                        case 1:
                                System.out.println("Hello");
                        break;
                }
        }
}
```

Note :- In the switch statement we can't pass long, float and double value.

Strings are allowed from JDK 1.7 version. enums are allowed from java 5 version.

---

01-11-2023

---

Loops in java :

---

A loop is nothing but repeatation of statement based on some condition.

In java we have 4 types of loop :

---

1) do-while loop (exit control loop)

2) while loop (entry control 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;  //block level variable
                        System.out.println("x value is :"+x);
                        x++;

    }
    while (x<=10); //error x is out of the scope
        }
}
```

The above program will generate compilation error because here 'x' is a block level variable.

---

```
public class DoWhile
{
        public static void main(String[] args)
        {
                int x = 1;  //local variable
    do
    {

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

    }
    while (x<=10); //error x is out of the scope
        }
```

```
}
```

---

Program on while loop :

---

```
public class While
{
	public static void main(String[] args)
	{
		int x = 0;
		while(x>=-10)
		{
	System.out.println(x);
		 x--;
		}

	}
}
```

---

```
//For loop
class For
{
	public static void main(String[] args)
	{
		for(int i=1; i<=10; i++)
		{
	if(i==5)
		break;
		  System.out.println(i);
```

```
                }
        }
}
```

---

Nested Loop :

---

If we place a loop inside another loop then it is called nested loop.

In nested loop with one value of outer loop the entire inner loop will be rotated.

```
for(int i=1; i<=5; i++)
{
  for(int j=1; j<=5; j++)
   {
   }
}
```

i=1 j=1 i=2 j=1 and so on

i=1 j=2

i=1 j=3

i=1 j=4

i=1 j=5

---

```
public class NestedLoop
{
        public static void main(String[] args)
        {
                int week = 4;
                int days = 7;
```

```
                for(int i = 1; i<=week; i++)
                {
                        System.out.println("Week :"+i);
                        for(int j=1; j<=days; j++)
                        {
                                System.out.println("Day :"+j);
                        }
        System.out.println();
                }
         }
}
```

---

for-each loop :

---

It is introduced from JDK 1.5 onwards.

It is also known as enhanced for loop.

It is used to retrieve the values from the collection.

---

```
public class FoEachLoop
{
        public static void main(String[] args)
        {
                int []x = {56, 45, 32, 91, 99, 34};

                java.util.Arrays.sort(x);
```

```
                    for(int  y  :  x)
                    {
                      System.out.println(y);
                    }
        }
}
```

Note :

---

1) In the above program each value of x is assigning to y variable.

2) x is an array variable but y is an ordinary variable

3) in java.util package there is a predefined class called Arrays which contains a static method sort(), by using this static method we can sort an array in ascending order. sort() method takes Object array as a parameter.

---

```
class ForEachLoopWithString
{
        public static void main(String[] args)
        {
                String fruits[] = {"Orange", "Mango","Apple","Kiwi"};

                 java.util.Arrays.sort(fruits);

                for(String fruit : fruits)
                {
                        System.out.println(fruit);
```

```
		}
	}
}
```

---

```
//Working with hetrogeneous types of data
public class FoEachLoop
{
	public static void main(String[] args)
	{
		Object x[] = {12, 89.78, true, "NIT"};

		for(Object y : x)
			System.out.println(y);

	}
}
```

---

02-11-2023

---

Working with static Method parameter and return type :

---

BLC :- The class contains logic only.

ELC :- The class which contains main method.

---

```
//A static method can be directly call within the same class
package com.ravi.pack1;
```

```
public class Test1
{
	public static void main (String[] arg)
	{
		square(5);
	}

	public static void square(int x)
	{
	 System.out.println(x*x);
	}
}
```

Here class name is also not required because static method is available in the same class.

---

Thia package contains two classes :

---

FindSquare.java

---

```
package com.ravi.pack2;

//BLC(Business Logic class)
public class FindSquare
{
  public static void getSquare(int x)
  {
    System.out.println("Square of "+x+" is :"+(x*x));
```

```
 }
}
```

Test2.java

---

```
//A static method available in another class can be
//call with class name

package com.ravi.pack2;

import java.util.Scanner;

//ELC(Executable Logic class)
public class Test2
{
	public static void main (String[] args)
	{
		Scanner sc = new Scanner(System.in);
		System.out.print("Please Enter a number :");
		int num = sc.nextInt();

	   FindSquare.getSquare(num);
	   sc.close();
	}
}
```

---

Thia package contains two classes :

---

FindSquare.java

---

```
//A static method returning integer value
package com.ravi.pack3;

public class FindSquare
{
        public static int getSquare(int x)
        {
          return x*x;
        }
}
```

Test3.java

---

```
package com.ravi.pack3;

public class Test3
{
        public static void main (String[] arg)
        {
                for(int i = 1; i <= 10; i++)
                {
                    int result = FindSquare.getSquare(i);
                    System.out.println("Square of " + i + " is: " + result);
                }
```

```
        }
}
```

In the above program communucation is possible from one module to anoather module as well as same getSquare() method we are using n number of times.

---

Thia package contains two classes :

---

Calculate.java(C)

---

```
/*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 squareAndCube(int num)
```

```
  {
            if(num == 0 || 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;

//ELC
public class Test4
{
          public static void main(String[] args)
          {
      Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Please Enter a Number :");
        int num = sc.nextInt();
                        int val = Calculate.squareAndCube(num);
                        System.out.println("Value is :"+val);
                        sc.close();
                }

}
```

---

Thia package contains two classes :

---

Rectangle.java

---

```
package com.ravi.pack5;

//BLC
public class Rectangle
{
  public static double getArea(double height, double width)
  {
            double area = height*width;
            return area;
  }
}
```

Test5.java

---

```
package com.ravi.pack5;


public class Test5
{
	public static void main(String[] args)
	{
		double areaOfRect = Rectangle.getArea(2.3, 8.9);
		System.out.println("Area of Rectangle is :"+areaOfRect);
	}
}
```

---

Thia package contains two classes :

---

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;


//ELC
public class Test6
{
	public static void main(String[] args)
	{
	 boolean val =   EvenOrOdd.isEven(4);
	 System.out.println("4 is even :"+val);


	 val =   EvenOrOdd.isEven(5);
	 System.out.println("5 is even :"+val);
	}


}
```

---

DecimalFormat :

---

There is predefined class called DecimalFormat availale in java.text package, it will provide Decimal format as per our

requirement.[("00.00")]

It contains a predefined method format() which takes double as a parameter.

```
DecimalFormat df = new DecimalFormat("00.00");
```

00.00 is the format.

This package contains 2 files :

Circle.java

---

```
//Area of Circle
package com.ravi.pack7;
public class Circle
{
 public static String getArea(double rad)
 {
        //if radius is negative or 0 then it should return 0
        if(rad ==0 || rad<0)
        {
            return ""+0;
        }
        else
        {
        final double PI = 3.14;  //final keyword to declare constant variable

        double area = PI*rad*rad;

        return  ""+area;
        }

 }
```

```
}
```

Test7.java

---

```
package com.ravi.pack7;

import java.text.DecimalFormat;
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 of Circle :");
	   double rad = sc.nextDouble();

	   String area = Circle.getArea(rad);
	  double areaOfCircle = Double.parseDouble(area);

    DecimalFormat df = new DecimalFormat("000.000");
    System.out.println("Area of Circle is :"+df.format(areaOfCircle));



	}
}
```

---

This package contains 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)
        {
                String details = Student.getStudentDetails(101, "Ravi", 14000.90);
```

```
                System.out.println("Student Details are :"+details);
        }

}
```

---

Table.java

---

```
package com.ravi.pack9;

//BLC
public class Table
{
  public static void printTable(int num) //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);
		}
	}

}
```

---

03-11-2023

---

Object Oriented Programming(OOPs) :

---

An Object is a physical entity which is existing in the real world.

Writing Java program on those Real World Object is known as Object Oriented Programming.

OOPs is a methdology to develop the programs using class and object.

In object oriented programming we concentrate on Objects.

Every Object contains properties(Data members or Variables) and behavior (Member function or Method).

Advantages of OOPs

---

We have 3 advantages

1) Modularity

2) Reusability

3) Flexibility

Features of OOPs

---

We have 6 features

1) Class

2) Object

3) Abstraction

4) Encapsulation

5) Inheritance

6) Polymorphism

class :-

---

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

A class is userdefined data type which contains data members and member function.

Example :

---

public classs Student

{

Student Data (Student Variables or Student properties)

+

Student behavior (Function / Method of the student)

}

A class is a logical representation of object.

*A class is a component which is used to define object properties and object behavior.

Object :

---

An object is a physical entity.

Anything which is existing in the real world is called as object.

Example :

---

Mouse, Laptop, key, pen and so on.

An object has 3 characteristics :

---

1) Identification (Name of the Object)

2) Properties (Variables OR data members)

3) Behavior (Function Or Method.)

---

Student.java

---

```
package com.ravi.oop;


public class Student
{
	//Properties
	String regNo;
	String name;
	double height;

	//behavior
	public void talk()
	{
		System.out.println("My name is : "+name);
		System.out.println("My Registration Number is : "+regNo);
		System.out.println("My height is : "+height);
	}

	public void writeExam()
	{
	 System.out.println(name +" writes exam on Saturday");
	}


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

```
        {
    Student raj = new Student();

    //Initializing the Object Properties
    raj.regNo = "NIT23001";
    raj.name = "Raj Gourav";
    raj.height = 5.9;

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

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

    Student priya = new Student();
    priya.regNo = "NIT23002";
    priya.name = "Priya Kumari";
    priya.height = 5.2;
    priya.talk();
    priya.writeExam();

        }

}
```

In student class we have created two Objects raj and priya(Diagram 03-NOV-23)

---

Pen.java

---

```
package com.ravi.oop;


public class Pen
{
	//Properties of Pen Object
	String name;
	String type;
	double price;
	String color;


	public String getPenInformation()
	{
		return "Pen name is :"+name+" It is a "+type+" Pen "+" it's price is :"+price+" and Pen color is :"+color;
	}


	public void write()
	{
		System.out.println("I am a "+type+ " type pen and I am used to write");
	}


	public static void main(String[] args)
	{
		Pen jetter = new Pen();
		jetter.name= "Jetter Pen";
		jetter.type ="ball";
		jetter.price = 35.00;
		jetter.color = "black";
```

```
                jetter.write();
                String information = jetter.getPenInformation();
                System.out.println(information);
        }

}
```

---

```
public class Test
{
        int x = 10; //Instance variable (non-static field)

        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
        }

}
```

Here with both the  objects, separate copy of x variable will be created.

t1 -----> x = 11

t2 -----> x = 9

---

04-11-2023

---

Default constructor added by the compiler

---

In java whenever we write a class and if we don't write any type of constructor then automatically compiler will add default constructor to the class.

```
class Test
{
   //Here in this class we don't have constructor
}
```

javac Test.java  (At the time of compilation automatically compiler will add default constructor)

```
class Test
{
   Test() //default constructor added by the compiler
   {
   }
}
```

```
public class Test
{
   public Test() //default constructor added by the compiler
   {
   }
```

}

---

Why compiler adds default constructor to our class :

---

If the compiler does not add default constructor to our class then object creation is not possible in java. At the time of object creation by using new keyword we depend upon the constructor.

The main purpose of defualt constructor(added by the compiler) to initialize the instance variables of the class with some default values.

The default values are:

byte - 0

short - 0

int - 0

long - 0

float - 0.0

double - 0.0

char - (Space)

boolean - false

String - null

Object - null

---

```
package com.ravi.instance_demo;


public class Employee
{
	int eno;
	String ename;
```

```
	public void show()
	{
		System.out.println(eno);
		System.out.println(ename);
	}

	public static void main(String[] args)
	{
		Employee raj = new Employee();
		raj.show();

	}

}
```

---

06-11-2023

---

How to provide our user-defined values for the instance variable :

---

The default values provided by compiler are not useful for the user, hence user will take a separate method (setStudentData()) to re-initialize the instance variable value so the user will get its own user-defined values as shown in the program. [Diagram 06th NOV]

---

```
package com.ravi.initializing_variable;

public class Student
{
   int rollNumber;
   String studentName;
```

```
  public void setStudentData()
  {
        rollNumber = 111;
        studentName = "Raj";
  }

  public void showStudentData()
  {
        System.out.println(rollNumber);
        System.out.println(studentName);
  }

        public static void main(String[] args)
        {
              Student raj = new Student();
              raj.showStudentData();

              raj.setStudentData();
              raj.showStudentData();
        }

}
```

---

//Program to initialize the Object properties using Scanner class :

---

```
package com.ravi.initializing_variable;
```

```
import java.util.Scanner;

public class Product
{
   int productId;
   String productName;

   public void setProductData()
   {
         Scanner sc = new Scanner(System.in);

         System.out.print("Enter Product Id :");
         productId = sc.nextInt();

         System.out.print("Enter Product Name :");
         productName = sc.nextLine();
         productName = sc.nextLine();
         sc.close();
   }

   public void getProductData()
   {
         System.out.println("Product is :"+productId);
         System.out.println("Product Name is :"+productName);
   }

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

```
	{
		Product laptop = new Product();
		laptop.getProductData();

		laptop.setProductData();
		laptop.getProductData();

	}

}
```

---

Writing BLC and ELC class :

---

In java, if we write everything in a single class then it is not an

object oriented approach even we are creating the object.

It is not a recommended approach becuase here we cannot reuse the BLC class.

We should always separate BLC and ELC class in two different files so we can reuse our BLC class as per my requirement.

3 Files :

---

Player.java

---

```
package com.ravi.blc_elc;

//BLC
```

```
public class Player
{
  int playerId;
  String playerName;

  public void setPlayerData(int id, String name)
  {
        playerId = id;
        playerName = name;
  }

  public void getPlayerData()
  {
        System.out.println("Player Id is :"+playerId);
        System.out.println("Player Name is :"+playerName);
  }

}
```

Virat.java

---

```
package com.ravi.blc_elc;

//ELC
public class Virat
{
      public static void main(String[] args)
      {
```

```
		Player virat = new Player();
		virat.setPlayerData(18, "Virat Kohli");
		virat.getPlayerData();
	}

}
```

Rohit.java

---

```
package com.ravi.blc_elc;

//ELC
public class Rohit {

	public static void main(String[] args)
	{
		Player rohit = new Player();
		rohit.setPlayerData(45, "RS");
		rohit.getPlayerData();

	}

}
```

Note :- In the above program we are assigning the parameter variable value to instance variable because parameter variables we cannot use outside of the method but

instance varibel we can use within the class as well as outside of the class depending upon access modifier.

---

07-11-2023

---

instance variable :

---

A non-static variable which is declared inside the class but outside of a method is called instance variable.

The life of an instance variable starts at the time of object creation.

[Instance variable is having strong association with object , we can't think about instance variable without object]

instance variables are always the part of the object.

As far as as its accessibility is concerned, instance variable we can use anywhere with object reference.

Parameter variable :

---

If a variable is declared inside the method parameter (not inside the method body) then the variables are called as parameter variables.

As far as its scope is concerned, parameters variables we can access within the same method body but not outside of the method.

---

this keyword :

---

Whenever our instance variable name and parameter variable name both are same then at the time of variable initialization our runtime environment gets confused that which one is an instance variable which one is parameter variable.

To avoid this problem we should use this keyword. this keyword always refers to the current object and we know that instance variables are the part of object.

this keyword we can't use from a static context.

---

2 Files :

---

Product.java

---

```
package com.ravi.this_keyword;

//BLC
public class Product
{
  int productId;
  String productName;

  public void setProductData(int productId, String productName)
  {
        this.productId = productId;
        this.productName = productName;

  }
```

```
  public void getProductData()
  {
            System.out.println("Product Id is :"+this.productId);
            System.out.println("Product Name is :"+this.productName);
  }
}
```

ProductDemo.java

---

```
package com.ravi.this_keyword;

public class ProductDemo
{
        public static void main(String[] args)
        {
                Product camera = new Product();
                camera.setProductData(111, "Nikon");
                camera.getProductData();
        }

}
```

---

Role of instance variable while creating the Object :

---

Whenever we create an object in java, a separate copy of all the instance variables will be created with each and every object as shown in the program below.(Diagram 07th NOV)

```
package com.ravi.this_keyword;
```

```
public class Test
{
	int x = 100;

	public static void main(String[] args)
	{
		Test t1 = new Test();
		Test t2 = new Test();

		++t1.x;   --t2.x;

		System.out.println(t1.x);
		System.out.println(t2.x);

	}

}
```

---

Role of static variable at the time of creating the object :

---

In static variable only one copy will be created and this single copy will be sharable by all the objects as shown in the program below.(Diagram 7th Nov)

```
package com.ravi.this_keyword;

public class Demo
{
```

```
   static int x = 100;

        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);
        }

}
```

The conclusion is :-

Instance variable -> Multiple copies with each object

static variable  -> Single copy for all the objects

---

08-11-2023

---

When we should declare a variable as an instance variable and when we should declare a variable as a static variable.

If the value of the variable is different for different objects then we should declare that variable as an instance variable,

On the other hand if the value of the variable is common for all the Objects then we should declare that variable as a static variable.

Note :- static variables are used to save the memory.

The following package contains 2 programs :

Student.java

---

```
package com.ravi.instance_static;
//BLC
public class Student
{
  int rollNumber;
  String studentName;
  String studentAddress;
  static String collegeName = "NIT";
  static String courseName = "Java";

  public void setStudentData(int roll, String name, String address)
  {
        rollNumber = roll;
        studentName = name;
        studentAddress = address;
  }

  public void getStudentData()
  {
```

```
        System.out.println("Student Roll Number is :"+this.rollNumber);
        System.out.println("Student Name is :"+this.studentName);
        System.out.println("Student Address is :"+this.studentAddress);
        System.out.println("College Name is :"+Student.collegeName);
        System.out.println("Course Name is :"+Student.courseName);
  }

}
```

StudentDemo.java

---

```
package com.ravi.instance_static;

public class StudentDemo
{
    public static void main(String[] args)
    {
        Student raj = new Student();
        raj.setStudentData(1, "Raj Gourav", "S R Nagar");

        Student priya = new Student();
        priya.setStudentData(2, "Priya", "Ameerpet");

        raj.getStudentData();
        System.out.println(".....");
        priya.getStudentData();
```

```
        }

}
```

---

How to print object properties value (instance variable value) :

---

If we want to print object properties value then we need to override a method called toString() available in java.lang.Object class.

Whenever we override this toString() method in our class then we need not to write any display() method to display our data(instance variable).

In order to call toString() method we need to print the object reference(name of the object) using System.out.println()

Example :-

```
Manager m = new Manager();
System.out.println(m); //Calling the toString() method of Manager class
```

---

2 files :

Manager.java

---

```
package com.ravi.object_properties;

public class Manager
{
  int managerId;
```

```
 double managerSalary;

 public void setManagerData(int id, double salary)
 {
          this.managerId = id;
          this.managerSalary = salary;
 }

        @Override
        public String toString()
        {
                return "Manager {managerId=" + managerId + ", managerSalary=" +
managerSalary + "}";
        }

}
```

ManagerDemo.java

---

```
package com.ravi.object_properties;

public class ManagerDemo {

        public static void main(String[] args)
        {
          Manager m = new Manager();
          m.setManagerData(111,80000);
          System.out.println(m); //toString method of Manager class
```

```
        }
}
```

---

Lab program :

---

2 files :

---

Employee.java

---

```
package com.ravi.lab;

public class Employee
{
 private int employeeNumber;
 private String employeeName;
 private double employeeSalary;
 private char employeeGrade;

 public void setEmployeeData(int eno, String name, double salary)
 {
        employeeNumber = eno;
        employeeName = name;
        employeeSalary = salary;
 }

@Override
public String toString()
```

```
{
	return "Employee [employeeNumber=" + employeeNumber + ",
employeeName=" + employeeName + ", employeeSalary="
				+ employeeSalary + ", employeeGrade=" + employeeGrade + "]";
}

 public void calculateEmployeeGrade()
 {
	 if(employeeSalary >= 100000)
		employeeGrade = 'A';
	 else if(employeeSalary >= 75000)
		employeeGrade = 'B';
	 else if(employeeSalary >= 50000)
		employeeGrade = 'C';
	 else
		employeeGrade = 'D';
 }

}
```

EmployeeDemo.java

---

```
package com.ravi.lab;

public class EmployeeDemo {

	public static void main(String[] args)
	{
```

```
        Employee e1 = new Employee();
        e1.setEmployeeData(111, "Virat", 20000);
        e1.calculateEmployeeGrade();
        System.out.println(e1);
    }

}
```

---

09-11-2023

---

Data hiding :

---

Data hiding means our data (variables) must be hidden from outer world that means no one can access our data directly from outside of the class.

To achieve this concept we should declare our class properties or data members as private.

We should not provide access of data directly but we can provide access to our data through methods. Once we are providing access to our data through method then we can PERFORM VALIDATION ON DATA WHICH ARE COMING FROM OUTER WORLD.

Note :- Data members must ne declared as private where as member functions (Method) must be declared as public.

---

2 files :

---

Customer.java

---

```
package com.ravi.data_hiding;

//BLC
public class Customer
{
 private double balance = 1000; //Data Hiding

 public void deposit(int deposit)
 {
        //Data validation
        if(deposit <= 0)
        {
             System.out.println("Amount cannot be deposited");
        }
        else
        {
             balance = balance + deposit;
            System.out.println("Balance after deposit :"+balance);
        }
 }

 public void withdraw(int withdraw)
 {
        balance = balance - withdraw;
        System.out.println("Balance after withdraw :"+balance);
 }

}
```

BankApplication.java

---

```
package com.ravi.data_hiding;


//ELC
public class BankApplication
{
	public static void main(String[] args)
	{
		Customer hacker = new Customer();
		hacker.deposit(1000);
		hacker.withdraw(500);


	}


}
```

---

Abstraction : [Hiding the Complexcity]

---

Showing the essential details without showing the background details (non-essential) is called Abstraction.

In real world a user always interacts with the functionality of the product but not the data or internal details, so for a user method/function is essential details where as data is non-essential details.

So being a developer we should always hide the data from the user(by declaring them private) where as on the other hand we should always decalre member function/Method as public so a user can easily interact with the product.

Example :

---

```
class Fan
{
  private int coil;
  private int wings;

  public void switchOn()
  {
  }

  public void switchOff()
  {
  }
}
```

Note :- Here User will interact with the functionality of the fan i.e switchOn and switchOff but will not interact with data(coil, wings) directly.

Note :- In java we can achieve abstarction by using abstract class and interface concept.

abstract class provide partial abstraction(0-100%) where as interface provides 100% abstraction. (Diagram 09-NOV-23)

---

10-11-2023

---

Constructor :

---

1) What is the advantage of writing constructor in our program :

---

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.

(As shown in the diagram [10 NOV])

---

Defination of Constructor :

---

It is known as Constructor because it is used to construct the object.

In java whenever the name of the class and name of the method both are same as well as if it does not contain any return type then it is called constructor.

Explicitly constructor does not return any value but implicitly constructor is returning current class object.(this keyword)

If we provide return type to the constructor then there is no compilation error but it will behave as a method.

*THE MAIN PURPOSE OF CONSTRUCTOR TO INITIALIZE THE OBJECT DATA I.E INSTANCE VARIABLE OF THE CLASS.

Every java class contains at least one constructor either explicitly written by user or implicitly added by compiler.

The access modifier of default constructor depends upon class access modifier.

We can write only return keyword inside the body of the constructor but not any kind of return value.

Constructors are automatically called and executed at the time of creating the object.

A constructor is executed only once per object that means every time we will create the object constructor will be executed.

Without constructor, Object creation is not possible in java by using new keyword.

Types of Constructor :

---

In Java we have 3 types of constructor :-

1) Default constructor

2) No Argument OR Zero Argument OR Non Parameterized OR Parameterless Constructor

3) Parameterized constructor

Default Constructor :

---

The constructor added by compiler, if a user does not write any type of constructor in a class called Default constructor.

```
public class Test
{
  //User has not written any kind of constructor
}

javac Test.java

program compiled .class file generated

public class Test
{
  public Test() //Default constrauctor
  {
  }
}
```

---

11-11-2023

---

2) No Argument Constructor :

---

If user writes any constructor without parameter then it is called No Argument constructor.

```
public class Trainer
{
  private int trainerId;
  private String trainerName;
```

```
 public Trainer() //No Argument Constructor
 {
  trainerId = 15;
  trainerName = "Raj";
 }
}
```

With No-Argument Constructor the limitation is, all the objects will

be initialized with same data so, it is not recomended to initialize the objecct data. To avoid this we introduced Parameterized constructor.

This packege contains two files :

---

Trainer.java

---

```
package com.ravi.no_arg_cons;

//BLC
public class Trainer
{
 private int trainerId;
 private String trainerName;

 public Trainer() //No Argument constructor
 {
        trainerId = 15;
        trainerName = "Raj";
```

```
 }
 @Override
 public String toString() {
	return "Trainer [trainerId=" + trainerId + ", trainerName=" + trainerName + "]";
 }

}
```

NoArgumentConstructor.java

---

```
package com.ravi.no_arg_cons;

public class NoArgumentConstructor
{
	public static void main(String[] args)
	{
		Trainer raj = new Trainer();

		Trainer ram = new Trainer();

		System.out.println(raj);
		System.out.println(ram);
	}

}
```

---

Parameterized Constructor :

---

If we pass 1 or more than 1 argumenet inside a constructor as a parameter then it is called parameterized constructor.

With the help of parameterized constructor all the objects will be initialized with different values as shown in the prgram.

First complete OOP uisng parameterized constructor :

---

2 Files :

Dog.java

---

```
package com.ravi.para;

public class Dog
{
  private String dogName;
  private double dogHeight;

        public Dog(String dogName, double dogHeight)
        {
                super();
                this.dogName = dogName;
                this.dogHeight = dogHeight;
        }

        @Override
        public String toString()
```

```
	{
		return "Dog [dogName=" + dogName + ", dogHeight=" + dogHeight + "]";
	}

}
```

ParameterizedConstructor.java

---

```
package com.ravi.para;

public class ParameterizedConstructor
{
  public static void main(String[] args)
 {
	Dog tommy = new Dog("Tommy", 3.3);
	System.out.println(tommy);



	Dog tiger = new Dog("Tiger",4.4);
	System.out.println(tiger);
 }
}
```

---

HAS-A relation program :

---

If we use class name as an object property to another class then it is called HAS-A Realtion Program.

---

Example :1

---

```
class Institute
{
}

class Student
{
  private Institute institute; //HAS-A relation
}
```

---

Example 2 :

---

```
class Order
{
}

class Customer
{
 private Order order; //HAS-A Relation
}
```

---

Example 3:

---

```
class Comapny
{
}
```

```
class Employee
{
  private Comapny comp; //HAS-A relation
}
```

---

Example 4 :

---

```
class Account
{
}

class Customer
{
  private Account acc; //HAS-A relation
}
```

---

3 files :

---

Institute.java

---

```
package com.ravi.has_a_relation;

//BLC
public class Institute
{
  private int instituteRegNo;
  private String instituteName;
```

```
	public Institute(int instituteRegNo, String instituteName)
	{
		super();
		this.instituteRegNo = instituteRegNo;
		this.instituteName = instituteName;
	}

	@Override
	public String toString()
	{
		return "INSTITUTE [instituteRegNo=" + instituteRegNo + ", instituteName="
+ instituteName + "]";
	}



}
```

Student.java

---

```
package com.ravi.has_a_relation;

public class Student
{
  private String regNo;
  private String studentName;
  private double studentFees;
```

```
 private Institute inst;  //HAS-A relation


public Student(String regNo, String studentName, double studentFees, Institute inst)
//inst = ins
{
	super();
	this.regNo = regNo;
	this.studentName = studentName;
	this.studentFees = studentFees;
	this.inst = inst;


}


@Override
public String toString() {
	return "Student [regNo=" + regNo + ", studentName=" + studentName + ",
studentFees=" + studentFees + ", inst="
				+ inst + "]";
}
```

```
}
```

HasA_Relation.java

---

```
package com.ravi.has_a_relation;
```

```
public class HasA_Relation {

	public static void main(String[] args)
	{
		Institute ins = new Institute(1521, "NIT");

		Student s1 = new Student("NIT26001","Raj",30000,ins);
		System.out.println(s1);

		Student s2 = new Student("NIT26002", "Priya", 29000, ins);
		System.out.println(s2);

	}

}
```

---

13-11-2023

---

Program on HAS-A relation :

---

3 files :

---

Comapny.java

---

```
package com.ravi.has_a_relation;
//BLC
public class Company
{
```

```
  private String companyName;
  private String compantLocation;

	public Company(String companyName, String compantLocation)
	{
		super();
		this.companyName = companyName;
		this.compantLocation = compantLocation;
	}

	@Override
	public String toString() {
		return "Company [companyName=" + companyName + ",
compantLocation=" + compantLocation + "]";
	}

}
```

Employee.java

---

```
package com.ravi.has_a_relation;

//BLC
public class Employee
{
  private int employeeId;
  private String employeeName;
  private double employeeSalary;
```

```
 private Company company; //HAS-A Realtion


public Employee(int employeeId, String employeeName, double employeeSalary,
Company company) //company = comp
{
	super();
	this.employeeId = employeeId;
	this.employeeName = employeeName;
	this.employeeSalary = employeeSalary;
	this.company = company;
}


@Override
public String toString() {
	return "Employee [employeeId=" + employeeId + ", employeeName=" +
employeeName + ", employeeSalary="
				+ employeeSalary + ", company=" + company + "]";
}


}
```

HasA_Relation.java

---

```
package com.ravi.has_a_relation;


public class HasA_Relation {


	public static void main(String[] args)
	{
```

```
        Company comp = new Company("TCS", "Hyd");


        Employee e1 = new Employee(1, "ABC", 40000, comp);
        System.out.println(e1);
    }

}
```

---

How to change the existing object data :

---

If we are thinking that by constructor (by creating the new object) we can change/modify the existing object data then it is not possible because it will create the object in a new memory location which does not have any concern with old object.

```
Company comp1 = new Company("TCS", "Hyd");


Company comp2 = new Company("Wipro", "Pune");
```

In order to modify the existing Object data as well as to read the private data value from outside of the class, java software people has provided setter and getter concept.

How to write setter and getter :

---

```
public class Employee
{
  private int eno;
  private String ename;

  //setter and getter for eno variable
```

```
    public void setEno(int eno)  //setter for eno variable
    {
      this.eno = eno;
    }

    public int getEno()  //getter for eno variable
    {
       return this.eno;
    }

    public void setEname(String ename) //setter for ename variable
    {
      this.ename = ename;
    }

    public String getEname() //setter for ename variable
    {
      return this.ename;
    }
}
```

setter :- It is used to modify existing object data and it is providing write operation

getetr :- It is used to read/get the private data value from BLC class so it is performing read operation.

Up to Here the Conclusion is :

---

CONSTRUCTOR :- Use to initialize the instance variable

SETTER :- Used to modify the existing object single data

GETTER :- Used to read single private data member value from BLC class.

Method :- To perform calculation OR any opertion on data

toString() :- Used to print Object properties.

---

Program on Setter and getter (2 files) :

---

Employee.java

---

```
package com.ravi.setter_getter;

public class Employee
{
  private int eno;
  private String ename;
  private double salry;
  private String role;

        public Employee(int eno, String ename, double salry, String role) {
                super();
                this.eno = eno;
                this.ename = ename;
```

```
        this.salry = salry;
        this.role = role;
}

public int getEno() {
        return eno;
}

public void setEno(int eno) {
        this.eno = eno;
}

public String getEname() {
        return ename;
}

public void setEname(String ename) {
        this.ename = ename;
}

public double getSalry() {
        return salry;
}

public void setSalry(double salry) {
        this.salry = salry;
}
```

```
	public String getRole() {
		return role;
	}

	public void setRole(String role) {
		this.role = role;
	}

	@Override
	public String toString() {
		return "Employee [eno=" + eno + ", ename=" + ename + ", salry=" + salry +
", role=" + role + "]";
	}




}
```

SetterAndGetter.java

---

```
package com.ravi.setter_getter;

public class Employee
{
  private int eno;
  private String ename;
  private double salry;
```

```
private String role;

	public Employee(int eno, String ename, double salry, String role) {
		super();
		this.eno = eno;
		this.ename = ename;
		this.salry = salry;
		this.role = role;
	}

	public int getEno() {
		return eno;
	}

	public void setEno(int eno) {
		this.eno = eno;
	}

	public String getEname() {
		return ename;
	}

	public void setEname(String ename) {
		this.ename = ename;
	}

	public double getSalry() {
		return salry;
```

```
	}

	public void setSalry(double salry) {
		this.salry = salry;
	}

	public String getRole() {
		return role;
	}

	public void setRole(String role) {
		this.role = role;
	}

	@Override
	public String toString() {
		return "Employee [eno=" + eno + ", ename=" + ename + ", salry=" + salry +
", role=" + role + "]";
	}

}
```

Main.java

---

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

```
    Employee emp = new Employee(101, "Raj", 40000, "Tester");
    emp.setRole("developer");
    System.out.println(emp);
 }
}
```

---

Encapsulation :

---

Binding the data with its associated method is called Encapsulation.

Encapsulation forces us to access our data through methods only.

Without data hiding encapsulation is not possible.

Encapsulation provides security because data is hidden from outer world.

We can achieve encapsulation in our class by using 2 process.

a) Declaring all the data members with private access modifier(data Hiding).

b) Defining setter and getter for all the data members individually.

Note :-

If we declare all the data memebrs with private access modifier then it is called TIGHTLY ENCAPSULATED CLASS.

On the other hand if some data members are declared with private but other are declared with protected or default or public then it is called LOOSLY ENCAPSULATED CLASS.

---

14-11-2023

---

* Passing Object reference to the Constructor (Copy Constructor)

---

We can pass an object reference to the constructor so we can copy the content of one object to another object.

```
class Employee
{
}

class Manager
{
  public Test(Employee emp)  //Passing an object reference to the constructor
  {

  }
}
```

Program on Copy Constructor :

---

We have 2 files :

Employee.java

---

```
package com.ravi.copy_constructor;

//BLC class
public class Employee
{
  private int employeeId;
  private String employeeName;

	public Employee(int employeeId, String employeeName)
	{
		super();
		this.employeeId = employeeId;
		this.employeeName = employeeName;
	}

	@Override
	public String toString() {
		return "Employee [employeeId=" + employeeId + ", employeeName=" +
employeeName + "]";
	}

	public int getEmployeeId() {
		return employeeId;
	}

	public String getEmployeeName() {
		return employeeName;
	}
```

```
}
```

Manager.java

---

```
package com.ravi.copy_constructor;

//BLC
public class Manager
{
	private int managerId;
	private String managerName;

	public Manager(Employee emp)  //emp = e1
	{
		this.managerId = emp.getEmployeeId();
		this.managerName = emp.getEmployeeName();
	}

	@Override
	public String toString() {
		return "Manager [managerId=" + managerId + ", managerName=" +
managerName + "]";
	}


}
```

CopyConstructor.java

---

```
package com.ravi.copy_constructor;


//ELC
public class CopyConstructor {


	public static void main(String[] args)
	{
		Employee e1 = new Employee(111, "Raj");
		Manager m1 = new Manager(e1);
		System.out.println(m1);


	}

}
```

Note :- In the above program we are taking the content of Employee class to initialize the Manager class data so we have 2 different BLC classes

We can also copy the content of same class object to another object as shown in the program below.

2 files :-

---

Player.java

---

```
package com.ravi.copy_constructor_demo;
```

```
public class Player
{
  private String name1, name2;

  public Player(String name1, String name2)
  {
        this.name1 = name1;
        this.name2 = name2;
  }

  public Player(Player p) //p = p1
  {
        this.name1 = p.name2;
        this.name2 = p.name1 ;
  }

@Override
public String toString() {
      return "Player [name1=" + name1 + ", name2=" + name2 + "]";
}




}
```

CopyConstructorDemo.java

---

```
package com.ravi.copy_constructor_demo;

public class CopyConstructorDemo {

	public static void main(String[] args)
	{
		Player p1 = new Player("Virat", "Rohit");
		System.out.println(p1);
		System.out.println(".............");
		Player p2 = new Player(p1);
		System.out.println(p2);

	}

}
```

---

Lab Program :

---

The payroll system of an organization involves calculating the gross salary of each type of employee and the tax applicable to each.

Create the following entity classes as described below.

Class Employee

Fields: id: int, name : String, basicSalary : double, HRAPer : double, DAPer : double

Public Method: calculateGrossSalary() - returns a double

Calculate the gross salary as : basicSalary +HRAPer +DAPer

Class Manager

Fields: id: int, name : String, basicSalary : double, HRAPer : double,DAPer : double, projectAllowance: double

Public Method: calculateGrossSalary() - returns a double

Calculate the gross salary as : basicSalary +HRAPer +DAPer + projectAllowance

Class Trainer

Fields: id: int, name : String, basicSalary : double, HRAPer : double,DAPer : double, batchCount: int, perkPerBatch: double

Public Method: calculateGrossSalary() - returns a double

Calculate the gross salary as : basicSalary +HRAPer +DAPer +(batchCount * perkPerBatch)

Class Sourcing

Fields: id: int, name : String, basicSalary : double, HRAPer : double,DAPer : double, enrollmentTarget: int, enrollmentReached: int, perkPerEnrollment: double

Public Method: calculateGrossSalary() - returns a double

Calculate the gross salary as : basicSalary +HRAPer +DAPer +((enrollmentReached/enrollmentTarget)*100)*perkPerEnrollment)

Class TaxUtil

Fields: None

Public Methods:

calculateTax(Employee) - returns a double

calculateTax(Manager) - returns a double

calculateTax(Trainer) - returns a double

calculateTax(Sourcing) - returns a double

Tax Calculation Logic: If gross salary is greater than 50000 tax is 20% else, tax is 5%

Note : Attributes/Fields must be non-Private for the above classes.

A ClassObject class is given to you with the main Method. Use this class to test your solution.

---

Employee.java

---

```
package com.batch_26_lab;


public class Employee
{
  int employeeId;
  String employeeName;
  double basicSalary;
  double HRAPer;
  double DAPer;




  public Employee(int employeeId, String employeeName, double basicSalary, double hRAPer, double dAPer) {
        super();
        this.employeeId = employeeId;
```

```
        this.employeeName = employeeName;
        this.basicSalary = basicSalary;
        HRAPer = hRAPer;
        DAPer = dAPer;
}




public double calculateGrossSalary()
  {
        return basicSalary +HRAPer +DAPer;
  }

}
```

Manager.java

---

```
package com.batch_26_lab;

public class Manager
{
  int managerId;
  String managerName;
  double basicSalary;
  double HRAPer;
  double DAPer;
```

```
 double projectAllowance;



 public Manager(int managerId, String managerName, double basicSalary, double
hRAPer, double dAPer,
                    double projectAllowance) {
        super();
        this.managerId = managerId;
        this.managerName = managerName;
        this.basicSalary = basicSalary;
        HRAPer = hRAPer;
        DAPer = dAPer;
        this.projectAllowance = projectAllowance;
}



public double calculateGrossSalary()
 {
         return basicSalary +HRAPer +DAPer + projectAllowance;
 }
}
```

Trainer.java

---

```
package com.batch_26_lab;
```

```
public class Trainer
{
 int trainerId;
 String trainerName;
 double basicSalary;
 double HRAPer;
 double DAPer;
 int batchCount;
 double perkPerBatch;



 public Trainer(int trainerId, String trainerName, double basicSalary, double hRAPer,
double dAPer, int batchCount,
                double perkPerBatch) {
        super();
        this.trainerId = trainerId;
        this.trainerName = trainerName;
        this.basicSalary = basicSalary;
        HRAPer = hRAPer;
        DAPer = dAPer;
        this.batchCount = batchCount;
        this.perkPerBatch = perkPerBatch;
}
```

```
public double calculateGrossSalary()
 {
          return basicSalary +HRAPer +DAPer +(batchCount * perkPerBatch);
 }

}
```

Sourcing.java

---

```
package com.batch_26_lab;

public class Sourcing
{
 int sourceId;
 String sourceName;
 double basicSalary;
 double HRAPer;
 double DAPer;
 int enrollmentTarget;
 int enrollmentReached;
 double perkPerEnrollment;



 public Sourcing(int sourceId, String sourceName, double basicSalary, double hRAPer,
double dAPer, int enrollmentTarget,
                  int enrollmentReached, double perkPerEnrollment) {
          super();
```

```
        this.sourceId = sourceId;
        this.sourceName = sourceName;
        this.basicSalary = basicSalary;
        HRAPer = hRAPer;
        DAPer = dAPer;
        this.enrollmentTarget = enrollmentTarget;
        this.enrollmentReached = enrollmentReached;
        this.perkPerEnrollment = perkPerEnrollment;
}



public double calculateGrossSalary()
 {
          return basicSalary +HRAPer +DAPer
+(((enrollmentReached/enrollmentTarget)*100)*perkPerEnrollment);
 }

}
```

TaxUtil.java

---

```
package com.batch_26_lab;

public class TaxUtil
{
  public double calculateTax(Employee employee)
```

```
{
	if(employee.calculateGrossSalary()> 50000)
	{
		return employee.calculateGrossSalary()*0.20;
	}
	else
	{
		return employee.calculateGrossSalary()*0.05;
	}
}

public double calculateTax(Manager manager)
{
	 if(manager.calculateGrossSalary()> 50000)
		{
			return manager.calculateGrossSalary()*0.20;
		}
		else
		{
			return manager.calculateGrossSalary()*0.05;
		}
}

public double calculateTax(Trainer trainer)
{
	 if(trainer.calculateGrossSalary()> 50000)
		{
			return trainer.calculateGrossSalary()*0.20;
```

```
                }
                else
                {
                        return trainer.calculateGrossSalary()*0.05;
                }
 }

 public double calculateTax(Sourcing sourcing)
 {
        if(sourcing.calculateGrossSalary()> 50000)
                {
                        return sourcing.calculateGrossSalary()*0.20;
                }
                else
                {
                        return sourcing.calculateGrossSalary()*0.05;
                }
 }

}
```

ClassObject.java

---

```
package com.batch_26_lab;

public class ClassObject {
```

```
        public static void main(String[] args)
        {
         Employee emp = new Employee(1, "Virat",40000, 3200, 2500);

         Manager m1 = new Manager(2, "Dravid", 75000, 3500, 4000, 10000);

         Trainer t1 = new Trainer(3, "Bumrah", 60000, 3500, 2800, 10, 250);

         Sourcing source = new Sourcing(4, "Rohit", 80000, 3300, 2800, 100, 20, 70);

         TaxUtil t = new TaxUtil();
        System.out.println("Employee Tax is :"+t.calculateTax(emp));
        System.out.println("Manager Tax is :"+t.calculateTax(m1));
        System.out.println("Trainer Tax is :"+t.calculateTax(t1));
        System.out.println("Sourcing Tax is :"+t.calculateTax(source));

        }

}
```

---

24-11-2023

---

Working with return type of the method :

---

We cannot define a method in java without return type.

A return type of a method can be the followings :

a) void [Will not return any kind of value]

b) All primitive data types [byte, short, int, long as so on]

c) class name as a return type.

Example :

```
public Test m2()
{

 return new Test();
        OR
    return null;
}
```

---

Program that describes how to work with method return type as a class :

---

This package contains 2 files :

---

Student.java

---

```
package com.batch_26_lab;

import java.util.Date;
import java.util.Scanner;

public class Student
{
```

```
  private int studentId;
  private String studentName;
  private double studentFees;
  private Date dateOfJoining;  //HAS-A Relation

public Student(int studentId, String studentName, double studentFees, Date
dateOfJoining)
{
        super();
        this.studentId = studentId;
        this.studentName = studentName;
        this.studentFees = studentFees;
        this.dateOfJoining = dateOfJoining;
}

 public static Student getStudentObject()
 {
         Scanner sc = new Scanner(System.in);
         System.out.print("Enter Student Id :");
         int id = sc.nextInt();

         System.out.print("Enter Student Name :");
         String name = sc.nextLine();
         name = sc.nextLine();

         System.out.print("Enter Student Fees :");
         double fees = sc.nextDouble();
```

```
        Date d = new Date();
        return new Student(id, name, fees, d);
}


@Override
public String toString() {
        return "Student [studentId=" + studentId + ", studentName=" + studentName + ",
studentFees=" + studentFees
                        + ", dateOfJoining=" + dateOfJoining + "]";
}


}
```

ClassAsAReturnType.java

---

```
package com.batch_26_lab;

import java.util.Scanner;

public class ClassAsAReturnType {

        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                System.out.print("How many objects you want :");
```

```
            int noOfObjects = sc.nextInt();


            for(int i=1; i<=noOfObjects; i++)
            {
                  Student object = Student.getStudentObject();
                  System.out.println(object);
            }


      }

}
```

Note :- Date is a predefined class avialble in java.util package which is used to provide current system date and time.

---

A class called Customer is given to you.

The task is to find the Applicable Credit card Type and create CardType object based on the Credit Points of a customer.

Define the following for the class.

Attributes :

customerName : String,private

creditPoints: int, private

Constructor :

parameterizedConstructor: for both cusotmerName & creditPoints in that order.

Methods :

Name of the method : getCreditPoints

Return Type : int

Modifier : public

Task : This method must return creditPoints

Name of the method : toString, Override it,

Return type : String

Task : return only customerName from this.

Create another class called CardType. Define the following for the class

Attributes :

customer : Customer, private

cardType : String, private

Constructor :

parameterizedConstructor: for customer and cardType attributes in that order

Methods :

Name of the method : toString Override this.

Return type : String

Modifier : public

Task : Return the string in the following format.

The Customer 'Rajeev' Is Eligible For 'Gold' Card.

Create One more class by name CardsOnOffer and define the following for the class.

Method :

Name Of the method : getOfferedCard

Return type : CardType

Modifiers: public,static

Arguments: Customer object

Task : Create and return a CardType object after logically finding cardType from creditPoints as per the below rules.

| creditPoints | | cardType |
|---|---|---|
| 100 - 500 | - | Silver |
| 501 - 1000 | - | Gold |
| 1000 > | - | Platinum |
| < 100 | - | EMI |

Creat an ELC class which contains Main method to test the working of the above.

---

This package contains 4 files :

---

Customer.java

---

```
package com.ravi.lab;


public class Customer
{
  private String customerName;
  private int creditPoints;


        public Customer(String customerName, int creditPoints)
```

```
	{
		super();
		this.customerName = customerName;
		this.creditPoints = creditPoints;
	}

	public int getCreditPoints()
	{
		return this.creditPoints;
	}

	@Override
	public String toString()
	{
		return this.customerName;
	}
}
```

CardType.java

---

```
package com.ravi.lab;

public class CardType
{
  private Customer customer; //HAS-A Relation
  private String cardType;
```

```
	public CardType(Customer customer, String cardType)
	{
		super();
		this.customer = customer;
		this.cardType = cardType;
	}

	@Override
	public String toString()
	{
		return "The Customer '"+this.customer+"' Is Eligible For
'"+this.cardType+"' Card.";
	}




}
```

CardsOnOffer.java

---

```
package com.ravi.lab;

public class CardsOnOffer
{
	public static CardType getOfferedCard(Customer obj)
	{
		int creditPoint = obj.getCreditPoints();
```

```
		if(creditPoint >=100 && creditPoint <=500)
		{
			return new CardType(obj, "Silver");
		}
		else if(creditPoint >500 && creditPoint <=1000)
		{
			return new CardType(obj, "Gold");
		}
		else if(creditPoint > 1000)
		{
			return new CardType(obj, "Platinum");
		}
		else
		{
			return new CardType(obj, "EMI");
		}
	}
}
```

CreditCard.java

---

```
package com.ravi.lab;

import java.util.Scanner;

public class CreditCard {

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

```
	{
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter Cutsomer Name :");
		String name = sc.nextLine();

		System.out.print("Enter Customer credit points :");
		int creditPoint = sc.nextInt();

		Customer c1 = new Customer(name, creditPoint);
		CardType cardType = CardsOnOffer.getOfferedCard(c1);
		System.out.println(cardType);
		sc.close();


	}

}
```

---

HEAP and STACK Diagram :

---

25-11-2023

---

Garbage Collector :

---

In C++, It is the responsibility of the programmer to allocate as well as to de-allocate the memory otherwise the corresponding memory will be blocked and we will get OutOfMemoryError.

In Java, Programmer is responsible to allocate the memory, memory de-allocation will be automatically done by garbage collector.

Garbage Collector will scan the heap area, identify which objects are not in use (The objects which does not contain any references) and it will delete those objects which are not in use.

---

How many ways we can make an object eligible for garbage collector :

There are 3 ways to make an object eligible for Garbage Collector.

1) Assigning null literal to the reference variable :

```
Employee e1 = new Employee();
    e1 = null;
```

2) Creating an object inside the method :

```
public void createObject()
{
  Employee e2 = new Employee();
}
```

Here we are creating the employee object inside a method so, once method execution is over the employee object is eligible for GC.

3) Assigning another object to the existing reference variable :

```
  Employee e3 = new Employee();
        e3 = new Employee();
```

---

HEAP And STACK Diagram Programs :

---

HEAP and Stack Diagram for CustomeDemo.java

---

```
class Customer
{
        private String name;
        private int id;

        public Customer(String name , int id) //constructor
        {
                this.name=name;
                this.id=id;
        }

        public void setId(int id)  //setter
        {
                this.id=id;
        }

        public int getId() //getter
        {
                return 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 is eligible for GC i.e 3000]

                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 are eligible for GC i.e 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
```

---

27-11-2023

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; //3000x
                t3.t = t4; //4000x
                t1.t = t2.t; //3000x
                t2.t = t4.t; //2000x

        System.out.println(t1.t.val);
        System.out.println(t2.t.val);
        System.out.println(t3.t.val);
        System.out.println(t4.t.val);
        }

}
// 300 200  400  200
------------------------------------------------------------------------
public class Employee
{
        int id = 100;

        public static void main(String[] args)
        {
```

```
            int val = 200;

            Employee e1 = new Employee();

            e1.id = val;

            update(e1);

            System.out.println(e1.id);

Employee e2 = new Employee();

            e2.id = 500;



            switchEmployees(e2,e1);     //3000x , 1000x

              //GC [2 objects 2000x and 4000x are eligible for GC]

                    System.out.println(e1.id);
              System.out.println(e2.id);
       }

     public static void update(Employee e)  //e = e1
    {
e.id = 500;
            e = new Employee();
            e.id = 400;
```

```
        }

        public static void switchEmployees(Employee e1, Employee e2)
        {
                int temp = e1.id;
                e1.id = e2.id; //500
                e2 = new Employee();
                e2.id = temp;
        }
 }
```

//500 500  500

---

Can we declare a Constructor with private access modifier ?

---

Yes, We can declare a constructor with private access modifier.

Example :

---

```
public class Demo
{
        private Demo()
        {
                System.out.println("Private Constructor !!");
        }
        public static void main(String[] args)
        {
```

```
                new Demo(); //Nameless OR Anonymous object
        }
}
```

---

We should declare a consructor as a private when that class contains only static method as well as when we want to create only one object for that class (Singleton class).

Example : java.lang.Math class contains private constructor.

We cannot declare constructor static and final.

---

28-11-2023

---

Instance block OR Non-static block :

---

Instance Block is a special block which will be executed at the time of creating the object.

Everytime we will create the object first of all instance block will be executed and then the body of the constructor will be executed because the 2nd line of any constructor is reserved for instance block.

The main purpose of instance block to initialize the instance variable before constructor body execution so, it is also known as Instance Initializer.

If we have multiple instance blocks are available in the class then it would be executed according to the order.[Top to bottom]

---

Program that describes instance block will be executed before the constructor body.

```
package com.ravi.instance;


class Test
{
	public Test()
	{
		super();
		System.out.println("No Argument constructor !!!");
	}


	{
		System.out.println("Instance Block");
	}
}


public class InstanceDemo
{
	public static void main(String[] args)
	{
		Test t1 = new Test();
	}


}
```

---

```
package com.ravi.instance;


class Demo
```

```
{
	int x;

	public Demo()
	{
		x = 34;
		System.out.println(x);
	}

	{
		x = 12;
		System.out.println(x);
	}
}

public class InstanceDemo1
{
	public static void main(String[] args)
	{
		new Demo();

	}

}
```

First of all instance initializer will initialize the instance variable and then Constructor will initialize.

---

```
package com.ravi.instance;


class Foo
{
	int x;

	public Foo()
	{
		System.out.println(x);
	}

	{
		x = 100;
		System.out.println(x);
	}


	{
		x = 200;
		System.out.println(x);
	}


	{
		x = 300;
		System.out.println(x);
	}
```

```
}



public class InstanceDemo2
{

	public static void main(String[] args)
	{
		new Foo();


	}
}
```

Instance blocks are executed according to the order.

---

```
package com.ravi.instance;


class Stuff
{
	public Stuff()
	{
		System.out.println("No Argument Constructor!!");


		{
			System.out.println("Instance Block");
		}
	}
```

```
}

public class InstanceDemo3
{
	public static void main(String[] args)
	{
		new Stuff();
	}

}
```

If we write instance block after the body of the constructor then first of all constructor body will be executed and then only instance block will be executed.

---

Relationship between the classes :

---

In Java, in between the classes we have 2 types of relation

1) IS-A Relation (We can achieve by using Inheritance Concept)

2) HAS-A Relation (We can achieve by using Association)

Inheritance (IS-A Relation)

---

Acquiring the properties of parent class in such a way that all the properties and features (except private) is by default available to the

child class is called Inheritance.

It is one of the most important feature of OOPs which provides "Code Reusability".

In java the parent class is known as super class where as child class is known as sub class.

By default all the features and properties of super class is available to sub class so the sub class need not to start the process from begning onwards.

Inheritance provides us hierarchical classification of classes, in this hierarchy if we move towards upward direction more generalized properties

will occur, on the other hand if we move towards downward direction more specialized properties will occur.

Types of Inheritance in Java :

---

We have 5 types of Inheritance

1) Single Level Inheritance

2) Multi Level Inheritance

3) Hierarchical Inheritance

4) Multiple Inheritance (Not supported by class)

5) Hybrid Inheritance

29-11-2023

---

Program on Single Level Inheritance :

---

3 Files :

---

Parent.java

---

```
package com.ravi.single_level;

public class Parent
{
  public void bike()
  {
            System.out.println("Honda Bike");
  }
}
```

Child.java

---

```
package com.ravi.single_level;

public class Child extends Parent
{
  public void car()
  {
            System.out.println("Audi Car");
  }
}
```

SingleLevelDemo1.java

---

```
package com.ravi.single_level;

public class SingleLevelDemo1 {
```

```
	public static void main(String[] args)
	{
		Child c = new Child();
		c.bike();
		c.car();

	}

}
```

---

Another Program on Single Level Inheritance :

---

3 Files :

---

Super.java

---

```
package com.ravi.single_level;

public class Super
{
	private int x;
	private int y;

	 public Super(int x, int y)
		{
			super();
			this.x = x;
```

```
                this.y = y;
        }

        public int getX()
        {
                return x;
        }

        public int getY()
        {
                return y;
        }
}
```

Sub.java

---

```
package com.ravi.single_level;

public class Sub extends Super
{
  public Sub()
  {
        super(100,200);
  }

  public void showData()
  {
         System.out.println("x values is :"+getX());
```

```
            System.out.println("y value is :"+getY());
  }



}
```

SingleLevelDemo2.java

---

```
package com.ravi.single_level;


public class SingleLevelDemo2
{
        public static void main(String[] args)
        {
                Sub s = new Sub();
                s.showData();
        }


}
```

Two important points from the above program :

---

1) If super class contains any constructor the to call the constructor of super class, super class object is not required, we can call the super class constructor with sub class object using super keyword.

2) In the super class we can declare our data member with private access modifier and by using getter() we can use these private data in the child

class (Encapsulation)

---

Another Program on Single Level Inheritance :

---

Organization

---

Emp (TemporaryEmployee)

|- eno

|- ename

|- eaddr

Pemp (PermanentEmployee)

|- department

|- designation

3 Files :

---

Emp.java

---

```
package com.ravi.oop;

public class Emp
{
  protected int employeeId;
  protected String employeeName;
  protected String employeeAddress;

        public Emp(int employeeId, String employeeName, String employeeAddress)
        {
```

```
		super();
		this.employeeId = employeeId;
		this.employeeName = employeeName;
		this.employeeAddress = employeeAddress;
	}

	@Override
	public String toString() {
		return "Emp [employeeId=" + employeeId + ", employeeName=" +
employeeName + ", employeeAddress="
				+ employeeAddress + "]";
	}
}
```

Pemp.java

---

```
package com.ravi.oop;

public class Pemp extends Emp
{
  protected String department;
  protected String designation;

	public Pemp(int employeeId, String employeeName, String employeeAddress,
String department, String designation)
	{
		super(employeeId, employeeName, employeeAddress);
		this.department = department;
		this.designation = designation;
```

```
	}

	@Override
	public String toString()
	{
		return ""+super.toString()+ "Pemp [department=" + department + ", designation=" + designation + "]";
	}
}
```

SingleLevelDemo3.java

---

```
package com.ravi.oop;

public class SingleLevelDemo3 {

	public static void main(String[] args)
	{
	  Pemp p = new Pemp(1, "Rohit", "Mumbai", "Cricket", "Batter");
	  System.out.println(p);

	}
}
```

---

01-12-2023

---

super keyword :

---

It is used to access the member of immediate super class. In java we can super keyword in 3 ways :

1) To call super class variable.

2) To call super class method.

3) To call super class constructor.

1) To call super class variable :

---

If the super class variable name and sub class variable name both are same (Variable shadow) and if we create an object for sub class then it will provide more priority to sub class variable, if we want to access the super class variable then we should use super keyword.

super keyword always refers to the immediate super class.

Just like this keyword we cannot also used super keyword from static

context.

3 Files :

---

Father.java

---

```
package com.ravi.super_var_demo;

public class Father
{
  protected double balance = 50000;
}
```

Daughter.java

---

```
package com.ravi.super_var_demo;

public class Daughter extends Father
{
	protected double balance = 18000;

	public void getBalance()
	{
		System.out.println("Daughter Balance is :"+balance);
		System.out.println("Father Balance is :"+super.balance);
	}
}
```

SuperVar.java

---

```
package com.ravi.super_var_demo;

public class SuperVar
{
	public static void main(String[] args)
	{
		Daughter d = new Daughter();
		d.getBalance();
	}
```

}

---

2) To call super class method :

---

If the super class method name and sub class method name both are same and if we create an object for sub class method then sub class will provide more priority to sub class method, if we want to call super class method then we should use super keyword.

SuperMethod.java [Single File Approach]

---

```
package com.ravi.super_method;


class Super
{
	public void show()
	{
		System.out.println("Super class Show method");
	}
}
class Sub extends Super
{
	public void show()
	{
		super.show();
		System.out.println("Sub class Show method");
	}
}
public class SuperMethod
{
```

```
    public static void main(String[] args)
    {
      new Sub().show();
    }
}
```

---

3) To call the super class Constructor :

---

In java, Whenever we write a class and if we don't write any type of

constrcutor then automatically one default constructor will be added by the compiler.

The First line of any constructor is reserved either for super() [super of] or this() [this of].
We have total 4 cases which are as follows :

Case 1 :

---

super() :- Automatically added by the compiler and it will invoke the

no argument constructor or defualt constructor of super class.

Program :

---

```
class Base
{
    public Base()
    {
        super();
        System.out.println("Base class No Argument constructor");
    }
}
```

```
class Derived extends Base
{
	public Derived()
	{
		super();
		System.out.println("Derived class No Argument constructor");
	}
}

public class CallingSuperClassNoArgument
{
	public static void main(String[] args)
	{
		Derived d1 = new Derived();

	}
}
```

---

Case 2 :

---

super("nit") :- It is used to call parameterized constructor of super class.

ConstructorTest.java[Single File Approach]

---

```
package com.ravi.oop;
class A
```

```
{
	A(String str)
	{
		super();
		System.out.println("My Institute Name is :"+str);
	}
}
class B extends A
{
	public B()
	{
		super("Nit");
		System.out.println("No Argument Constructor of B class");
	}
}

public class ConstructorTest
{
	public static void main(String[] args)
	{
		new B();
	}
}
```

---

Case 3 :

---

this() :- It is used to call no argument constructor of it own class.

```
package com.ravi.oop;


class Parent
{
	public Parent()
	{
		super();
		System.out.println("No Argument of Parent class");
	}


	public Parent(int x)
	{
		this();
		System.out.println("Parameterized constructor of Parent class :"+x);
	}
}
class Child extends Parent
{
	public Child()
	{
		super(15);
		System.out.println("No Argumenet of Child class");
	}
}


public class CallingNoArgumentOfOwn
{
```

```
	public static void main(String[] args)
	{
		new Child();
	}

}
```

---

case 4 :

---

this("Nit") :- It is used to call parameterized constructor of its own class. [Current class]

```
package com.ravi.oop;
class A
{
	A()
	{
		this("Nit"); //Own class parameterized constructor
		System.out.println("No Argument Constructor of A ");
	}
	A(String str)
	{
	  super();
		System.out.println("My Institute Name is :"+str);
	}
}
class B extends A
{
```

```
	public B()
	{
		super();
		System.out.println("No Argument Constructor of B ");
	}
}

public class ConstructorTest
{
	public static void main(String[] args)
	{
		new B();
	}
}
```

---

02-12-2023

---

Program on super keyword using Single level Inheritance :

---

SuperDemo1.java[Single File Approach]

---

```
package com.ravi.super_ex;

class Shape
{
	protected int x; //x = 10

	public Shape(int x) //x = 10
```

```
        {
         this.x= x;
         System.out.println("x value is :"+this.x);
        }
}
class Square extends Shape
{
        public Square(int side)  //side = 10
        {
                super(side);
        }

        public void getSquareArea()
        {
                System.out.println("Area of Square :"+(x*x));
        }
}

public class SuperDemo1
{
        public static void main(String[] args)
        {
    Square ss = new Square(10);
    ss.getSquareArea();
        }

}
```

---

Program on super keyword using Hierarchical Inheritance :

---

```
package com.ravi.super_demo;

import java.util.Scanner;

class Shape
{
	protected int x;

	public Shape(int x)
	{
	 this.x= x;
	 System.out.println("x value is :"+this.x);
	}
}

class Circle extends Shape
{
	protected final double PI = 3.14;
	public Circle(int radius)
	{
		super(radius);
	}

	public void getAreaOfCircle()
	{
		double area = PI*x*x;
```

```
                System.out.println("Area of Circle is :"+area);
        }

}

class Rectangle extends Shape
{
        protected int breadth;
        public Rectangle(int length, int breadth)
        {
                super(length);
                this.breadth = breadth;
        }

        public void getAreaOfRectangle()
        {
                double area = breadth * x;
                System.out.println("Area of Rectangle is :"+area);
        }
}
public class SuperDemo2
{
        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter the radius of the Circle :");
                int radius = sc.nextInt();
```

```
                Circle c1 = new Circle(radius);
                c1.getAreaOfCircle();

                System.out.print("Enter the length of the Rectangle :");
                int length = sc.nextInt();

                System.out.print("Enter the breadth of the Rectangle :");
                int breadth = sc.nextInt();

                Rectangle rr = new Rectangle(length, breadth);
                rr.getAreaOfRectangle();

        }

}
```

---

Program on Multilevel Inheritance :

---

```
package com.ravi.super_demo;

class Student
{
        protected int rollNumber;
        protected String studentName;
        protected String studentAddress;
}
class Science extends Student
{
```

```
	protected int physics, chemistry;
}
class PCM extends Science
{
	int maths;

	public PCM(int sno, String sname, String addr, int phy, int che, int math)
	{
		rollNumber = sno;
		studentName = sname;
		studentAddress = addr;
		physics = phy;
		chemistry = che;
		maths = math;
	}

	@Override
	public String toString()
	{
		return "PCM [maths=" + maths + ", physics=" + physics + ", chemistry=" +
chemistry + ", rollNumber="
				+ rollNumber + ", studentName=" + studentName + ",
studentAddress=" + studentAddress + "]";
	}
}

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

```
	{
		System.out.println(new PCM(1,"A","Ameerpet",78,89,99));
	}
}
```

---

Program on Multilevel Inheritance :

---

```
GrandFather
  |-> land()

Father
  |-> house()

 Son
  |-> car()
```

---

Program on Hierarchical Inheritance :

---

4 files :

---

Employee.java

---

```
package com.ravi.hier;

public class Employee
{
  protected double salary;
}
```

Developer.java

---

```
package com.ravi.hier;

public class Developer extends Employee
{
  public Developer(double salary)
  {
            this.salary = salary;
  }

        @Override
        public String toString()
        {
                return "Developer [salary=" + salary + "]";
        }



}
```

Designer.java

---

```
package com.ravi.hier;
```

```
public class Designer extends Employee
{
	 public Designer(double salary)
	 {
		  this.salary = salary;
	 }

	@Override
	public String toString() {
		return "Designer [salary=" + salary + "]";
	}

}
```

HierarchicalInheritance.java

---

```
package com.ravi.hier;

public class HierarchicalInheritance {

	public static void main(String[] args)
	{
		System.out.println(new Developer(80000));
		System.out.println(new Designer(45000));

	}
}
```

---

IQ :

---

Why Java does not support multiple Inheritance ?

---

Multiple Inheritance is a situation where a sub class wants to inherit the properties of two or more than two super classes.

If a sub class contains two super classes then the super keyword available in the sub class constructor will confuse to call the super class constructor through super keyword so, Java does not support multiple inheritance using classes but the same we can achieve by using interface.

It is also known as Diamond problem in Java.(Diagram 02-DEC-23).

---

04-12-2023

---

Access modifiers in java :

---

In terms of accessibility, Java software people has provided 4 access modifiers which will describe the accessibility level of the class as well as the member of the class.

Test.java

---

```
class Demo  [We can use within the same package only because class is
{         not public]
}

public class Test [We can use this class within the same package from
{                 another package also by using import statement]
```

}

The 4 access modifiers are :

a) private (Accessible within the same class only)

b) default (Accessible within the same package only)

c) protected (Accessible from another package as well but using Inheritance only)

d) public (Accessible from everywhere, No restriction)

private :- It is the most restrictive access modifier because the member declared as private can be accessible within the same class only.

In java we cannot declare an outer class with private access modifier.

[An outer class we can declare with public, abstract and final only]

default :- It is an access modifier which is less restrictive than private because the member which are declared as default will be accessible within the same package only.

If a user does not write any kind of access modifier (private, protected and public) and by default it will be treated as default because for default members there is no physical existance of default access modifier.

protected :- It is an access modifier which is less restrictive than default because the members which are declared with protected access modifier will be accessible from the same package as well as from the another package but using Inheritance.

Here we have two programs from different package to the show the accessibility level of protected :

---

Test.java[com.ravi.m1]

---

```
package com.ravi.m1;

public class Test
{
  protected int x = 100;
}
```

Main.java[com.ravi.m2]

```
package com.ravi.m2;

import com.ravi.m1.Test;

public class Main extends Test
{
	public static void main(String[] args)
	{
		Main m1 = new Main();
		System.out.println(m1.x);
	}

}
```

public :- It is an access modifier which does not contain any kind of restriction because the member declared with public access

modifier will be accessible from everywhere.

Note :- According to OOP we should always declare our class and methods with public access modifier.

instance variable (private)

class variable(Genrally public)

local variable (final)

parameter variable (final)

---

JVM Architecture

---

Class Loader sub-system (Static variable + static block and static method)

Runtime Data Areas (Instance variable + instance method + instance block)

Execution Engine

---

05-12-2023

---

HAS-A Relation :

---

If we are using a reference variable as a property of another class then it is called HAS-A Relation.

Example :-

---

class Order

```
{
  private int orderId;
  private String itemName;
}

class Customer
{
   Private Order order; //HAS-A Relation
}
```

Association :

---

Association is a connection between two separate classes that can be built up through their Objects.

The association builds a relationship between the classes and describes how much a class knows about another class.

This relationship can be unidirectional or bi-directional. In Java, the association can have one-to-one, one-to-many, many-to-one and many-to-many relationships.

Example:-

One to One: A person can have only one PAN card

One to many: A Bank can have many Employees

Many to one:  Many employees can work in single department

Many to Many: A Bank can have multiple customers and a customer can have multiple bank accounts.

The folowing package contains 3 files :

---

Student.java

---

```
package com.ravi.association_demo;


public class Student
{
  private int studentId;  //111
  private String studentName;
  private long mobileNumber;


	public Student(int studentId, String studentName, long mobileNumber) {
		super();
		this.studentId = studentId;
		this.studentName = studentName;
		this.mobileNumber = mobileNumber;
	}


	@Override
	public String toString() {
		return "Student [studentId=" + studentId + ", studentName=" +
studentName + ", mobileNumber=" + mobileNumber + "]";
	}


	public int getStudentId() {
		return studentId;
	}
```

```
	public String getStudentName() {
		return studentName;
	}

	public long getMobileNumber() {
		return mobileNumber;
	}

}
```

Trainer.java

---

```
package com.ravi.association_demo;

import java.util.Scanner;

public class Trainer
{
  public static void viewStudentProfile(Student student)
  {
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter the Student Id :");
		int id = sc.nextInt();  //111

		if(id == student.getStudentId())
		{
			System.out.println(student);
		}
```

```
            else
            {
                    System.err.println("Id is not Available");
            }
            sc.close();
  }
}
```

Main.java

---

```
package com.ravi.association_demo;

public class Main
{
        public static void main(String[] args)
        {
                Student s1 = new Student(111, "Raj", 9812345678L);
                Trainer.viewStudentProfile(s1);
        }
}
```

---

Composition [Strong reference] :

---

Composition :

---

Composition relation is a restricted form of Aggregation in which two classes (or entities) are highly dependent on each other; the composed object cannot exist without the other entity. The composition can be described as a part-of HAS-A relationship.

A car has an engine. Composition makes strong relationship between the objects. It means that if we destroy the owner object(car Object), its members  will be also destroyed with it. For example, if the Car is destroyed the engine will also be destroyed as well.

This package contains 3 files :

---

Engine.java

---

```
package com.ravi.composition;


public class Engine
{
  private String engineType;
  private int horsePower;


        public Engine(String engineType, int horsePower)
        {
                super();
                this.engineType = engineType;
                this.horsePower = horsePower;
        }


        @Override
        public String toString() {
                return "Engine [engineType=" + engineType + ", horsePower=" +
horsePower + "]";
        }
```

```
}
```

Car.java

---

```
package com.ravi.composition;

public class Car
{
  private String carName;
  private double carPrice;
  private Engine engine;   //HAS-A Relation

        public Car(String carName, double carPrice)
        {
                super();
                this.carName = carName;
                this.carPrice = carPrice;
                this.engine = new Engine("Battery", 1400); //Compostion

        }

        @Override
        public String toString() {
                return "Car [carName=" + carName + ", carPrice=" + carPrice + ", engine="
+ engine + "]";
        }
```

```
}
```

Composition.java

---

```
package com.ravi.composition;

public class Composition {

	public static void main(String[] args)
	{
		Car c1 = new Car("Naxon", 1600000.89);
		System.out.println(c1);
	}

}
```

H/W :

---

Person and Heart

Laptop and Motherboard

---

Aggregation : [Weak reference]

---

Aggregation (Weak Reference) :

---

Aggregation is a relation between two classes which can be built through entity reference,  It is a weak reference type that means one object entity does not depend upon another object entity.

An aggregation is a form of Association, which is a one-way relationship or a unidirectional association.

For example, customers can have orders but the reverse is not possible, hence unidirectional in nature.

```
class Customer
{
int cid;
String cname;
Order order;//has-a relationship
}
```

---

This package contains 3 files :

---

Employee.java

---

```
package com.ravi.aggregation_demo_1;

public class Employee
{
  private int employeeId;
  private String employeeName;
  private Organisation org;  //HAS-A Relation
```

```
public Employee(int employeeId, String employeeName, Organisation org)
{
	super();
	this.employeeId = employeeId;
	this.employeeName = employeeName;
	this.org = org;
}

@Override
public String toString() {
	return "Employee [employeeId=" + employeeId + ", employeeName=" +
employeeName + ", org=" + org + "]";
}

}
```

Organisation.java

---

```
package com.ravi.aggregation_demo_1;

public class Organisation
{
  private int organisationId;
  private String organisationName;
  private String organisationAddress;

public int getOrganisationId() {
```

```
        return organisationId;
}

public void setOrganisationId(int organisationId) {
        this.organisationId = organisationId;
}

public String getOrganisationName() {
        return organisationName;
}

public void setOrganisationName(String organisationName) {
        this.organisationName = organisationName;
}

public String getOrganisationAddress() {
        return organisationAddress;
}

public void setOrganisationAddress(String organisationAddress) {
        this.organisationAddress = organisationAddress;
}
```

```
public Organisation(int organisationId, String organisationName, String
organisationAddress) {
	super();
	this.organisationId = organisationId;
	this.organisationName = organisationName;
	this.organisationAddress = organisationAddress;
}



@Override
public String toString()
{
	return "Organisation [organisationId=" + organisationId + ", organisationName=" +
organisationName
			+ ", organisationAddress=" + organisationAddress + "]";
}




}
```

Main.java

---

```
package com.ravi.aggregation_demo_1;


public class ELC {


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

```
	{
		Organisation wipro = new Organisation(1, "Wipro Limited", "Hyd");
		Organisation tcs = new Organisation(2, "TCS Limited", "Pune");

		Employee e1 = new Employee(101, "Virat", wipro);

		Employee e2 = new Employee(201, "Rohit", tcs);

		System.out.println(e1);
		System.out.println(e2);


	}

}
```

---

3 files :

---

Aadhar.java

---

```
package com.ravi.aggregation_demo3;

import java.util.Date;

public class Aadhar
{
  private long aadharNumber;
  private Date issueDate;
```

```
 private String issuingAuthority;

	public Aadhar(long aadharNumber, Date issueDate, String issuingAuthority) {
		super();
		this.aadharNumber = aadharNumber;
		this.issueDate = issueDate;
		this.issuingAuthority = issuingAuthority;
	}

	@Override
	public String toString() {
		return "Aadhar [aadharNumber=" + aadharNumber + ", issueDate=" +
issueDate + ", issuingAuthority="
				+ issuingAuthority + "]";
	}




}
```

Person.java

---

```
package com.ravi.aggregation_demo3;

public class Person
{
```

```
  private String personName;
  private String address;
  private Aadhar aadhar;

	public Person(String personName, String address, Aadhar aadhar)
	{
		super();
		this.personName = personName;
		this.address = address;
		this.aadhar = aadhar;
	}

	@Override
	public String toString() {
		return "Person [personName=" + personName + ", address=" + address +
", aadhar=" + aadhar + "]";
	}



}

package com.ravi.aggregation_demo3;

import java.util.Date;

public class ELC {

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

```
        {
          System.out.println(new Person("Virat","Ampt",new Aadhar(5678_4567_2345L,
new Date(), "uidai")));

        }

}
```

---

06-12-2023

---

Polymorphism :

---

Poly means "many" and morphism means "forms".

It is a Greek word whose meaning is "same object having different behavior".

In our real life a person or a human being can perform so many task, in the same way in our programming languages a method or a constructor can perform so many task.

Eg:-

```
void add(int a, int b)

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

void add(float a, float b)

void add(int a, float b)
```

Polymorphism can be divided into two types :

1) Static polymorphism OR Compile time polymorphism OR Early binding

2) Dynamic Polymorphism OR Runtime polymorphism OR Late binding

Static Polymorphism :

---

The polymorphism which exist at the time of compilation is called static polymorphism.

In static polymorphism, compiler has very good idea that which method is going to invoke(call) depending upon the type of parameter we have passed in the method.

This type of preplan polymorphism is called static polymorphism and we can achieve static ploymorphism by using Method Overloading.

Example:- Method Overloading

Dynamic Polymorphism :

---

The polymorphism which exist at runtime is called dynamic polymorphism.

In dynamic polymorphism, compiler does not have any idea about method calling, at runtime JVM will decide that which method is invoked depending upon the class type.

This type of polymorphism is called dynamic polymorphism.(Dynamic Method dispatched). We can achieve dynamic polymorphism by using Method

Overriding

Example:- Method Overriding

---

Note :- In static polymorphism method calling is done at the time of compilation so it is also knoan as Early Binding.

On the other hand In dynamic Polymorphism method calling is done at the time of execution so it is also known as Late Binding.

---

Method Overloading :

---

Writing two or more methods in the same class or even in the super and sub class in such a way that the method name must be same but the argument must be different.

While Overloading a method we can change the return type of the method.

Method overloading is possible in the same class as well as super and sub class.

While overloading the method the argument must be different otherwise there will be ambiguity problem.

IQ:

---

Can we overload the main method ?

---

We can overload the main method but JVM will always search the main method which takes String array as a parameter.

Example :

---

```
public static void main(String [] args)  //JVM will serach this method
{
}

public static void main(String x)
{
}

public static void main(int y)
{
}
```

---

Program on Constructor Overloading :

---

2 files :

Addition.java

---

```
package com.ravi.constructor_overloading;

public class Addition
{
  public Addition(int x, int y)
  {
            System.out.println("Sum of two integer is :"+(x+y));
  }

  public Addition(int x, int y, int z)
```

```
  {
        System.out.println("Sum of three integer is :"+(x+y+z));
  }

  public Addition(float x, float y)
  {
        System.out.println("Sum of two float is :"+(x+y));
  }
}
```

Main.java

---

```
package com.ravi.constructor_overloading;

public class Main {

      public static void main(String []  args)
      {
            new Addition(2.3f, 7.8F);
            new Addition(10, 20, 30);
            new Addition(12,90);
      }

}
```

---

Program on Constructor Overloading [Constructor Chaining]

---

2 files :

Addition.java

---

```
package com.ravi.constructor_overloading1;

public class Addition
{
  public Addition(int x, int y)
  {
        System.out.println("Sum of two integer is :"+(x+y));
  }

  public Addition(int x, int y, int z)
  {
        this(100,200);
        System.out.println("Sum of three integer is :"+(x+y+z));
  }

  public Addition(float x, float y)
  {
        this(10,20,30);
        System.out.println("Sum of two float is :"+(x+y));
  }
}
```

Main.java

---

```
package com.ravi.constructor_overloading1;
```

```
public class Main {

	public static void main(String []  args)
	{
		new Addition(2.3f, 7.8F);

	}

}
```

---

Method Overloading by changing the return type :

---

Sum.java

---

```
package com.ravi.method_overload;

public class Sum
{
  public int add(int x, int y)
  {
		int z = x+y;
		return z;
  }

  public String add(String x, String y) //data base
  {
		String z = x+y;
```

```
        return z;
 }

 public double add(double x, double y)
 {
        double z = x+y;
        return z;
 }
}
```

Main.java

---

```
package com.ravi.method_overload;

public class Main {

        public static void main(String[] args)
        {
                Sum s1 = new Sum();
                String add = s1.add("Data", "base");

                int x = s1.add(12, 12);

                double y = s1.add(12.89, 12.90);


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

```
        }
}
```

---

07-12-2023

---

Var args :

---

Var-Args :

---

It was introduced from JDK 1.5 onwards.

It stands for variable argument. It is an array variable which can hold 0 to n number of parameters of same type or different type by using Object class.

It is represented by exactly 3 dots (...) so it can accept any number of argument (0 to nth) that means now we need not to define method body again and again, if there is change in method parameter value.

var-args must be only one and last argument.(var args must be the last argument)

We can use var-args as a method parameter only.

---

Program to accept 0 to n number of parameters using var args

2 Files :

---

Test.java

---

```
package com.ravi.var_args;
```

```
public class Test
{
  public void input(int ...x)  //Array
 {
            System.out.println("Var args executed");
 }
}
```

Main.java

---

```
package com.ravi.var_args;

public class Main {

        public static void main(String ...x)
        {
                Test t1 = new Test();
                t1.input();
                t1.input(12);
                t1.input(15,19);
                t1.input(10,20,30);
                t1.input(10,20,30,40);
                t1.input(10,20,30,40,50);

        }

}
```

---

Program to add the parameter values using var args concept.

---

Test.java

---

```
package com.ravi.var_args1;


public class Test
{
  public void acceptData(int ...x)
  {
        int sum = 0;


        for(int y : x)
        {
              sum = sum + y;
        }
        System.out.println("Sum of Parameter :"+sum);
  }
}
```

Main.java

---

```
package com.ravi.var_args1;


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

```
		Test t1 = new Test();
		t1.acceptData();
		t1.acceptData(10,20);
		t1.acceptData(10,20,30);
		t1.acceptData(100,100,100,100);

	}

}
```

---

Program that describes var args must be only one and last argument.

---

2 files :

---

Test.java

---

```
package com.ravi.var_args2;

public class Test
{

	/*
	 * public void accept(float ...x, int ...y) //invalid { }
	 *
	 * public void accept(int ...x, int y) //Invalid {
	 *
	 * }
	 */
```

```
	public void accept(int x, int... y) // valid
	{
		System.out.println("x value is :"+x);
		for (int z : y)
		{
			System.out.println(z);
		}
	}
}
```

Main.java

---

```
package com.ravi.var_args2;

public class Main {

	public static void main(String[] args)
	{
		Test t1 = new Test();
		t1.accept(10, 20,30,40,50);
	}
}
```

---

Program to accept hetrogeneous types of data :

---

2 files :

---

Test.java

---

```
package com.ravi.var_args3;


public class Test
{
  public void acceptHetro(Object ...obj)
  {
            for(Object o : obj)
            {
                      System.out.println(o);
            }
  }
}
```

Main.java

---

```
package com.ravi.var_args3;


public class Main {


          public static void main(String[] args)
          {
                    new Test().acceptHetro(true,45.90,12,'A', new String("Ravi"));


          }
```

```
}
```

---

Ambiguity issues while overloading a method :(WAV)

---

Test.java (Single File Approach)

---

```
class Input
{
	public void accept(byte b)
	{
		System.out.println("byte is invoked..");
	}
	public void accept(short b)
	{
		System.out.println("short is invoked..");
	}

}
public class Test
{
	public static void main(String [] india)
	{
		Input i = new Input();
		    i.accept(9); //error
	}
}
```

Note :- The above program will generate compilation error because accept method does not contain any integer type variable.

Test.java

---

```
class Input
{
	public void accept(byte b)
	{
		System.out.println("byte is invoked..");
	}
	public void accept(short b)
	{
		System.out.println("short is invoked..");
	}

}
public class Test
{
	public static void main(String [] india)
	{
		Input i = new Input();
		    i.accept((byte)9);
			i.accept((short)15);
	}
}
```

---

class Input

```
{
	public void accept(int b)
	{
		System.out.println("int is invoked..");
	}
	public void accept(long b)
	{
		System.out.println("long is invoked..");
	}

}
public class Test
{
	public static void main(String [] india)
	{
		Input i = new Input();
		i.accept(15);
	}
}
```

Note :- Here the method which accepts int as a parameter will be invoked.

---

```
//Nearest data type
class Input
{
	public void accept(String b)
	{
		System.out.println("String is invoked..");
```

```
        }
        public void accept(Object b)
        {
                System.out.println("Object is invoked..");
        }

}
public class Test
{
        public static void main(String [] india)
        {
                Input i = new Input();
                i.accept("NIT");  //String will be executed
                i.accept(null);  //String will be executed
        }
}
```

Points to remember :

---

1) While ambiguity issues compiler will provide more priority to the nearest data type.

2) While ambiguity issues compiler will also provide the priority on the basis of following (WAV)

[Widening -> Autoboxing -> var-args]

---

```
//Nearest data type (WAV)
class Input
{
	public void accept(long b)
	{
		System.out.println("Widening invoked..");
	}
	public void accept(Integer b)
	{
		System.out.println("Autoboxing is invoked..");
	}

}
public class Test
{
	public static void main(String [] india)
	{
		Input i = new Input();
		i.accept(15);  //Widening
	}
}
```

---

```
//Nearest data type (WAV)
class Input
{
	public void accept(int ...b)
```

```
	{
		System.out.println("Var args invoked..");
	}
	public void accept(Integer b)
	{
		System.out.println("Autoboxing is invoked..");
	}

}
public class Test
{
	public static void main(String [] india)
	{
		Input i = new Input();
		i.accept(15); //Autoboxing
	}
}
```

---

```
package com.ravi.ambigity_issues;

class D
{
	 public void access(Integer x)
	 {
	   System.out.println("Autoboxing is invoked :"+x);
	 }
```

```
    public void access(String x)
    {
        System.out.println("String is invoked :"+x);
    }

}
public class Main5 {

    public static void main(String[] args)
    {
        D d1 = new D();
        //d1.access(null);   //Invalid

    }

}
```

Note :- Here It will generate compilation error becuase Integer and String both can hold null.

---

In System.out.println(), println() method is the example of Method Overloading.

---

```
class Demo
{
  static String str = "India";
}

Demo.str.length();
```

In the same way we can also understand :

```
public class System
{
  static PrintStream out;  //HAS-A Relation
}
```

```
System.out.println();
```

Program :

---

```
package com.ravi.input_data;

class Demo
{
	static String str = "Hyderabad";
}
public class EmployeeData
{
	public static void main(String[] args) throws Exception
	{
		System.out.println(Demo.str.length());

	}

}
```

08-12-2023

---

Method Overriding :

---

Writing two or more methods in the super and sub class in such a way that method signature(method name along with method parameter) of both the methods must be same in the super and sub classes.

While working with method overriding generally we can't change the return type of the method but from JDK 1.5 onwards we can change the return type of the method in only one case that is known as Co-Variant.

Without inheritance method overriding is not possible that means if there is no inheritance there is no method overriding.

---

What is the advantage of Method Overriding ?

---

The advantage of Method Overriding is, each class is specifying its own specific behavior.

---

Upcasting and Downcasting :

---

Upcasting :-

---

It is possble to assign sub class object to super class reference variable using dynamic polymorphism. It is known as Upcasting.

Example:- Animal a = new Dog(); //valid [upcasting]

Downcasting :

---

By default downcasting is not possible, Here we are trying to assign super class object to sub class reference variable but the same we can achieve by using explicit type casting. It is known as downcasting

Eg:- Dog d = new Animal(); //Invalid

Dog d =(Dog) new Animal(); //Valid because Explicit type casting

But by using above statement (Downcasting) whenever we call a method we will get a runtime exception called java.lang.ClassCastException. [Animal cann't be cast to Dog]

---

09-12-2023

---

Program on Method Overriding :

---

```
package com.ravi.method_overriding_demo;


class Animal
{
	public void eat()
	{
		System.out.println("Can't say which type...");
	}
}
class Lion extends Animal
{
	public void eat()
	{
```

```
                System.out.println("Non Veg Type Animal");
        }
}

public class AnimalDemo
{
        public static void main(String[] args)
        {
                Animal a = new Lion();
                a.eat();
        }

}
```

---

@Override Annotation :

---

In Java we have a concept called Annotation, introduced from JDK 1.5 onwards. All the annotations must be start with @ symbol.

@Override annotation is optional but it is always a good practice to write @Override annotation before the Overridden method so compiler as well as user will get the confirmation that the method is overridden method and it is available in the super class.

If we use @Override annotation before the name of the overridden method in the sub class and if the method is not available in the super class then it will generate a compilation error so it is different from comment because comment will not generate any kind of compilation error if method is not an overridden method, so this is how it is different from comment.

---

```
package com.ravi.method_overriding_demo;
```

```
class Shape
{
	public void draw()
	{
		System.out.println("We cannot say");
	}
}
class Rectange extends Shape
{
	@Override
	public void draw()
	{
		System.out.println("Drawing Rectange");
	}
}
class Square extends Shape
{
	@Override
	public void draw()
	{
		System.out.println("Drawing Square");
	}
}
public class ShapeDemo
{
	public static void main(String[] args)
	{
```

```
        Shape s;
        s = new Rectange();  s.draw(); //Dynamic Method Dispatch
        s = new Square(); s.draw();    //Dynamic Method Dispatch
    }

}
```

---

Role of Access Modifier while Overriding a method :

---

While overriding the method from super class, the access modifier of sub class method must be greater or equal in comparison to access modifier of super class method otherwise we will get compilation error.

public is greater than protected, protected is greater than default (public > protected > default)

[default < protected < public]

So the conclusion is we can't reduce the visibility while overriding a method.

Note :- private access modifier is not availble (visible) in sub class so it is not the part of method overriding.

```
package com.ravi.method_overriding_demo;

class RBI
{
    public void loan()
    {
        System.out.println("Bank should provide loan");
```

```
	}
}
class SBI extends RBI
{
	@Override
	public void loan()
	{
		System.out.println("SBI provides loan at 9.4%");
	}
}
class BOI extends RBI
{
	@Override
	public void loan()
	{
		System.out.println("BOI provides loan at 10.4%");
	}
}
public class RBIDemo
{
	public static void main(String[] args)
	{
		RBI r ;
		 r = new SBI(); r.loan(); //Dynamic Method Dispatched
		 r = new BOI(); r.loan(); //Dynamic Method Dispatched

	}
```

}

---

11-12-2023

---

Co-variant concept in method overriding :

---

In general we cann't change the return type of method while overriding a method. if we try to change it will generate compilation error as shown in the program below.

ReturnType.java

---

```
class Super
{
	public void m1()
	{
		System.out.println("Super class m1 method...");
	}
}
class Sub extends Super
{
	@Override
	public int m1()
	{
		System.out.println("Sub class m1 method...");
		return 0;
	}
}
```

```
public class ReturnType
{
	public static void main(String[] args)
	{
		Super s1 = new Sub();
		s1.m1();

	}

}
```

---

But from JDK 1.5 onwards we can change the return type of the method in only one case that the return type of both the METHODS(SUPER AND SUB CLASS METHODS) MUST BE IN INHERITANCE RELATIONSHIP called Co-Variant as shown in the program below.

Note :- Co-variant will not work with primitive data type, it will work only with classes

---

```
class Bird
{
}
class Sparrow extends Bird
{
}

class Parent
{
	public Bird show()
	{
		System.out.println("Show Method of Parent class!!");
```

```
                return new Sparrow();
        }
}
class Child extends Parent
{
        @Override
        public Sparrow show()
        {
                System.out.println("Show Method of Child class!!");
                return new Sparrow();
        }
}

public class CoVariant
{
        public static void main(String[] args)
        {
                Parent p1 = new Child();   p1.show();
        }
}
```

In the above program we have changed the return type of overridden method because Bird and Sparrow both are in IS-A relationship.

---

Using co-variant describes that Object is the super class of all the classes.

```
package com.ravi.method_return_type;
```

```
class Bird
{
	public Object fly()
	{
		System.out.println("Bird fly");
		return null;
	}
}
class Parrot extends Bird
{
	@Override
	public Bird fly()  //Bird is sub class of Object
	{
		System.out.println("Parrot fly");
		return null;
	}
}
public class CoVariant2
{
	public static void main(String[] args)
	{
		Bird b1 = new Parrot();
		b1.fly();
	}

}
```

---

* Can we override main method?

OR

Can we override static method

OR

What is method hiding in java?

Points to remember :(4 points)

---

1) We can't override static method because it is the part of the class but not the part of the Object

2) We can't overide static method with non-static (instance) method

3) We can't overide non-static method with static method.

4) We can't override main method but we can overload the main method, Here JVM will always search the main method which contains String array as a parameter.

Defination

---

We can't override static method because it is not the part of the object it is executed at the time of loading the .class file into JVM memory.

If a sub class defines a static method with the same signature as it is already defined in the super class static method , the method in the sub class is hidden by the method in the super class.

We can declare a method with the same signature in the sub class as declared in the super class which  looks like we can override static methods in Java but in reality this is method hiding.

Note :- a) we can't override static and private methods.

b) In static methods we cannot apply @Override annotation

---

Program on Method Hiding :

---

```
class Super
{
	public static void m1()
	{
		System.out.println("Super class m1 static method!!!");
	}
}
class Sub extends Super
{
	public static void m1()  //Method Hiding
	{
		System.out.println("Sub class m1 static method!!!");
	}
}
public class StaticMethod
{
	public static void main(String[] args)
	{
		Super s1 = new Sub();
		s1.m1();
	}
}
```

Note :- We cannot override static, final and private Methods

---

final keyword :

---

In java we use final keyword to provide some kind of restrictions.

We can use final keyword in three ways in java

1) To declare a class as a final (Inheritance is not possible)

2) To declare a method as a final (We can't override)

3) To declare a variable(Field) as a final (We can't perform re-assignment)

To declare a class as a final :

---

Whenever we declare a class as a final class then we cann't extend or inherit that class otherwise we will get a compilation error.

We should declare a class as a final if the composition of the class (logic of the class) is very important and we don't want to share the feature of the class to some other developer to modify the original behavior of the existing class, In that situation we should declare a class as a final.

Declaring a class as a final does not mean that the variables and methods declared inside the class will also become as a final only the class behavior is final that means we can modify the variables value as well as we can create the object for the final classes.

Note :- In java String and all wrapper classes are declared as final class.

---

For final class we cannot provide inheritance :

---

```
final class A
{
	private int x = 100;
	public void setData()
	{
		x = 120;
		System.out.println(x);
	}
}
class B extends A
{
}
public class FinalClassEx
{
	public static void main(String[] args)
	{
		B b1 = new B();
		b1.setData();
	}
}
```

---

//For final class we can create an object and modify the data as well

```
final class Test
{
	private int data = 100;


	public void setData(int data)
	{
		this.data = data;
		System.out.println("Data value is :"+data);
	}
}
public class FinalClassEx1
{
	public static void main(String[] args)
	{
		Test t1 = new Test();
		t1.setData(200);
	}
}
```

---

2) To declare a method as a final (Overriding is not possible)

---

Whenever we declare a method as a final then we can't override that method in the sub class otherwise there will be a compilation error.

We should declare a method as a final if the body of the method i.e the implementation of the method is very important and we don't want to override or change the super class method body by sub class method body then we should declare the super class method as final method.

```
class A
{
        protected int a = 10;
        protected int b = 20;

    public final void calculate()
         {
                int sum = a+b;
                System.out.println("Sum is :"+sum);
         }
}
class B extends A
{
        @Override
        public void calculate() //error
        {
                int mul = a*b;
                System.out.println("Mul is :"+mul);
        }
}
public class FinalMethodEx
{
        public static void main(String [] args)
        {
                 A a1 = new B();
                 a1.calculate();
        }
```

}

---

3) To declare a variable(field) as a final :(Re-assignment is not possible)

---

In older langugaes like C and C++ we use "const" keyword to declare a constant variable but in java const is a reserved word for future use so instead of const we should use "final" keyword.

If we declare a variable as a final then we can't perform re-assignment (i.e nothing but re-initialization) of that variable.

In java It is always a better practise to declare a final variable by uppercase letter according to the naming convention.

Some example of predefined final variables

Byte.MIN_VALUE -> MIN_VALUE is a static and final variable

Byte.MAX_VALUE -> MAX_VALUE is a static and final variable

Example:- final int DATA = 10; (Now we can not perform re-assignment )

```
class A
{
	final int A = 10;
	public void setData()
	{
		A = 10;
```

```
                System.out.println("A value is :"+A);
        }
}
class FinalVarEx
{
        public static void main(String[] args)
        {
                A a1 = new A();
                a1.setData();
        }
}
```

---

12-12-2023

---

Blank final variable :

---

If we declare a final variable without initialization then it is called Blank final variable.

1) final variables must be initialized at the time of declaration or later (only constructor), after that we can't perform re-initialization.

2) A blank final variable can't be initialized by default constructor.

3) A blank final variable must be initialized by the user as a part of constructor. If we have multiple constructor then final variable must be initialized with all the constructor to provide values for the blank final variable to all the objects.

---

```
public class BlankFinalVar
{
```

```
	final int A;  //Blank final variable


	public static void main(String[] args)
	{
		BlankFinalVar fv = new BlankFinalVar();
		System.out.println(fv.A);
	}
}
```

Note :- We will get compilation error because default constructor cannot initialize the blank final variable.

---

```
 class Demo
{
	final int A; // blank final variable


	public Demo()  //No Argument constructor
	{
		A = 15;
		System.out.println(A);
	}


	public Demo(int x)  //parameterized constructor
	{
		A = x;
		System.out.println(x);
	}
```

```
}
public class  BlankFinalVariable
{
        public static void main(String[] args)
                {
            Demo d1 = new Demo();


            Demo d2 = new Demo(8);
          }
}
```

Note :- Blank final variable must be initialized through all the constructor for all different types of Objects.

---

Method Chaining :

---

In order to write concise coding we can call a method upon another method.

Whenver we call a method upon another method then last method return type will decide which class method we can invoke further on the existing method.

The last method call will decide the final return type of the method.

Program on Method Chaining :

---

```
package com.ravi.files;


public class MethodChaining {


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

```
	{
		String str = "india";
		boolean isStart = str.concat(" is great ").toUpperCase().startsWith("I");
		System.out.println(isStart);

		String str1 = "Hyderabad";
		int length = str1.toLowerCase().length();
		System.out.println(length);
	}

}
```

---

Object class and it's Method :

---

There is a predefined class called Object available in java.lang package, this Object class is by default the super class of all the classes we have in java.

```
class Test
{

}
```

Note :- Object is the super class for this Test class. by default this Object class is super class so explicitly we need not to mention.

Since, Object is the super class of all the classes in java that means we can override the method of Object class as well as we can use the methods of Object class anywhere in java because every class is sub class of Object class.

The Object class provides some common behavior to each sub class Object like we can compare two objects (equals(Object obj)), we can create clone (duplicate) objects (clone()), we can print object properties(instance variable) by using toString(), providing a unique number to each and every object(hashCode()) and so on.

---

public native int hashCode() :

---

It is a predefined method of Object class.

Every Object contains a unique number generated by JVM at the time of Object creation is called hashCode.

we can find out the hashCode value of an Object by using hashCode() method of Object class, return type of this method is int.

```
package com.ravi.introduction;


public class Demo
{
  public static void main(String[] args)
 {
        Demo d1 = new Demo();
        System.out.println("Hashcode of demo object is :"+d1.hashCode());

        Demo d2 = new Demo();
        System.out.println("Hashcode of demo object is :"+d2.hashCode());



        }
```

}

---

public final native Class getClass() :-

---

It is a predefined method of Object class.

This method returns the runtime class of the object, the return type of this method is java.lang.Class.

This method will provide the class keyword + fully qualified name

[fully qualified name = Package Name + class name]

This getClass() method return type is java.lang.Class so further we can apply any other method of java.lang.Class class to this method.

//getClass() method of Object class

```
package com.ravi.introduction;

public class Demo1
{
	public static void main(String[] args)
	{
		Demo1 d1 = new Demo1();
		System.out.println(d1.getClass());
		System.out.println(".....");
		System.out.println(d1.getClass().getName());

	}
```

}

Note :- If we call getClass() method in s.o.p statement then it will provide class keyword + Fully Qualified Name.

On the other hand if we call getClass().getName() [getName() method is available in java.lang.Class] then it will provide only fully Qualified name.

---

public String toString() :

---

It is a predefined method of Object class.

it returns a string representation of the object. In general, the toString method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read

toString() method of Object class conatins following logic.

```
public String toString()
{
        return getClass().getName()+" @ "+Integer.toHexString(hashCode());
}
```

Please note internally the toString() method is calling the hashCode() and getClass() method of Object class.

In java whenever we print any Object reference by using System.out.println() then internally it will invoke the toString() method of Object class as shown in the following program.

---

13-12-2023

---

//calling the toString() of Object class

```
package com.ravi.introduction;

public class Demo2
{
	public static void main(String[] args)
	{
		Demo2 d2 = new Demo2();
		System.out.println(d2); //calling toString() Object class
	}

}
```

---

//Overriding the toString method()

```
package com.ravi.introduction;

public class Demo3
{
	@Override
	public String toString()
	{
		return "Overriding toString method ";
	}

	public static void main(String[] args)
	{
```

```
		Demo3 d1 = new Demo3();
		System.out.println(d1);

		Object d2 = new Demo3();
		System.out.println(d2);

	}

}
```

---

Overriding hashCode() method :

---

As we know hashCode is a unique number generated for each and every object but here, we are overriding the hashcode method and we are providing a fixed value as shown in the program.

```
//Overriding hashCode() method with a fixed value(Not a recomended approach)
package com.ravi.introduction;

class Employee
{
	private int id;

	public Employee(int id)
	{
		super();
		this.id = id;
	}
```

```
	@Override
	public int hashCode()
	{
		return 15;
	}
}

public class Demo4
{
	public static void main(String[] args)
	{
		Employee e1 = new Employee(10);
		Employee e2 = new Employee(20);

		System.out.println(e1.hashCode());
		System.out.println(e2.hashCode());
	}
}
```

It is not a recommended way to provide a fixed value for hashCode, The following program explains how to provide unique value for hashCode.

```
package com.ravi.hash_code;

class Employee
{
	private int eid;
```

```
	public Employee(int eid)
	{
		super();
		this.eid = eid;
	}

	@Override
	public int hashCode()
	{
		return this.eid;
	}
}

public class OverrideHashCode
{
	public static void main(String[] args)
	{
		Employee e1 = new Employee(12);
		Employee e2 = new Employee(22);

		System.out.println(e1.hashCode());
		System.out.println(e2.hashCode());

	}

}
```

---

public boolean equals(Object obj) :

---

It is a predefined method of Object class.

It is used to compare two Objects based on the reference i.e memory address becuase internally equals method uses == operator which always compares based on the the reference i.e the memory address so even though two objects are content wise same but still it will generate false.

---

```
package com.ravi.over_equals;


public class Product
{
  private int productId;
  private String productName;


  public Product(int productId, String productName)
  {
                super();
                this.productId = productId;
                this.productName = productName;
  }


        @Override
        public String toString()
        {
                return "Product [productId=" + productId + ", productName=" +
productName + "]";
        }
```

```
	public int getProductId() {
		return productId;
	}

	public String getProductName() {
		return productName;
	}

}

package com.ravi.equals;

public class ProductComparison
{
	public static void main(String[] args)
	{
		Product p1 = new Product(111, "Laptop");
		Product p2 = new Product(222, "Camera");
		Product p3 = new Product(111, "Laptop");

		System.out.println("Comparison using == operator");
		System.out.println(p1==p2); //false
		System.out.println(p1==p3); //false

		System.out.println("Comparison using equals method");
		System.out.println(p1.equals(p2)); //false
		System.out.println(p1.equals(p3)); //false
```

```
        }


}
```

---

Overriding the equals(Object obj) method for content comparison :

---

As we know Object class equals method meant for memory reference (memory address) comparison so even two objects are content wise same still it will provide false.

If we want to compare two objects based on their content then we should override the equals(Object obj) from Object class.

2 file :

---

Product.java

---

```
package com.ravi.over_equals;


public class Product
{
  private int productId;
  private String productName;


  public Product(int productId, String productName)
  {
                super();
                this.productId = productId;
                this.productName = productName;
  }
```

```
//Overriding the equals() method for content comparison
@Override
public boolean equals(Object obj) //obj = p2
{
  int pid1 = this.productId;
  String pname1 = this.productName;

  Product p2 = (Product) obj; //Down casting
  int pid2 = p2.productId;
  String pname2 = p2.productName;

  if(pid1 == pid2 && pname1.equals(pname2))
  {
        return true;
  }
  else
  {
        return false;
  }

}


      @Override
      public String toString()
      {
```

```
		return "Product [productId=" + productId + ", productName=" +
productName + "]";
	}

	public int getProductId() {
		return productId;
	}

	public String getProductName() {
		return productName;
	}
}
```

ProductComparison.java

---

```
package com.ravi.over_equals;

public class ProductComparison {

	public static void main(String[] args)
	{
		Product p1 = new Product(111, "Laptop");
		Product p2 = new Product(111, "Laptop");
		System.out.println(p1.equals(p2)); //true

	}

}
```

---

Comparison using instanceof operator :

---

Product.java

---

```
package com.ravi.instance_of;


public class Product
{
  private int productId;
  private String productName;


  public Product(int productId, String productName)
  {
                super();
                this.productId = productId;
                this.productName = productName;
  }


  //Overriding the equals() method for content comparison
  public boolean equals(Object obj)
  {
          if(obj instanceof Product)
          {
                  Product p2 = (Product) obj;
                  if(this.productId == p2.productId &&
this.productName.equals(p2.productName))
                  {
```

```
                return true;
            }
            else
            {
                return false;
            }
        }
        else
        {
            System.err.println("Comparison is not possible");
            return false;
        }
    }

    @Override
    public String toString()
    {
        return "Product [productId=" + productId + ", productName=" +
productName + "]";
    }

    public int getProductId() {
        return productId;
    }
```

```
	public String getProductName() {
		return productName;
	}
}
```

Manager.java

---

```
package com.ravi.instance_of;

public class Manager
{
  private int managerId;
  private String managerName;

	public Manager(int managerId, String managerName) {
		super();
		this.managerId = managerId;
		this.managerName = managerName;
	}
}
```

ProductComparison.java

---

```
package com.ravi.instance_of;

public class ProductComparison
{
```

```
	public static void main(String[] args)
	{
		Product p1 = new Product(111, "Laptop");
		Product p2 = new Product(111, "Laptop");
		Manager m1 = new Manager(111, "Virat");
		System.out.println(p1.equals(m1));
		System.out.println(p1.equals(null));
		System.out.println(p1.equals(p2));

	}

}
```

---

14-12-2023

---

enum in java :

---

An enum is class in java that is used to represent group of universal constants. It is introduced from JDK 1.5 onwards.

In order to craete an enum, we can use enum keyword and all the univarsal constants of the enum must be separeted by comma. Semicolon is optional at the end.

Example:-

```
enum Color
{
   RED, BLUE, BLACK, PINK    //public + static + final
```

}

The enum constants are by default public, static and final.

An enum we can define inside the class, outside of the class and even inside of the method.

If we define an enum inside the class then we can apply public, private, protected and static.

Every enum in java extends java.lang.Enum class so an enum can implement many interfaces but can't extends a class.

By deafult every enum is implicitly final so we can't inherit an enum.

In order to get the constant value of an enum we can use values() method which returns enum array, but this method is provided as part of enum keyword that means this method is not available inside the Enum class(java.lang.Enum).

In order to get the order position of enum constants we can use ordinal() method which is given inside the enum class and the return type of this method is int. The order position of enum constant will start from 0.

As we know an enum is just like a class so we can define any method, constructor inside an enum. Constructor must be either private or default.

*All the enum constants are by default object of type enum.

Enum constants must be decalred at the first line of enum otherwise we will get compilation error.

From java 1.5 onwards we can pass an enum in a switch statement.

---

```
public class Test1
{
	public static void main(String[] args)
	{
		enum Month
		{
			JANUARY, FEBRUARY,MARCH    //public + static + final
		}

		System.out.println(Month.MARCH);
	}
}
```

---

```
enum Month
{
	JANUARY,FEBRUARY,MARCH
}
public class Test2
{
	enum Color { RED,BLUE,BLACK }

   public static void main(String[] args)
	{
		enum Week {SUNDAY, MONDAY, TUESDAY }

		System.out.println(Month.FEBRUARY);
```

```
                System.out.println(Color.RED);
                System.out.println(Week.SUNDAY);
        }
}
```

Note :- From the above Program it is clear that we can define an enum inside a class, outside of a class and inside a method as well.

---

```
//Comapring the constant of an enum
public class Test3
{
        enum Color { RED,BLUE }

    public static void main(String args[])
    {
        Color c1 = Color.RED;
        Color c2 = Color.RED;

        if(c1 == c2)
        {
            System.out.println("==");
        }
        if(c1.equals(c2))
        {
            System.out.println("equals");
        }
    }
}
```

Here we will get true in both the cases because == operator always compares the memory address and Object class equals() method also uses == oprator.

---

15-12-2023

---

H.W (DemoTest--> com.ravi.lab)

---

Program-1

---

A class Employee is given to you. It contains the following:

Instance Variables:

name:String  private

employeeId: int  private

salary: double private

Methods: set and get methods for all.

Default constructor AND  All-fields constructor

An Enum called ManagerType is given to you. It defines the two types of managers, HR and Sales

Create two sub classes of Employee called Manager and Clerk.

The details of each class is given below.

For class Manager:

Instance Variables:

type: enum ManagerType private

Methods: set and get methods for type.

Default constructor AND  All-fields constructor

Override :

setSalary(): If the type is HR manager, add 10000 to the given salary

and for a sales manager, add 5000 to the given salary.

For class Clerk:

Instance Variables:

speed: int

accuracy:int

Methods: set and get methods for speed and accuracy.

Default constructor AND All-fields constructor.

Override:

setSalary(): If the clerk has a typing speed of greater than 70 AND accuracy greater than 80 then add 1000 to the salary. Otherwise set the same salary.

Note that any change in speed and accuracy(using setSpeed() or setAccuracy() ) should result in

recalculation of salary, as the Trainee may qualify for the extra amount.

HOWEVER, when once the extra 1000 is given for extra speed/ and accuracy it should not be given again.

For example, if a Clerk's speed is already 85 and accuracy is already 75, and the speed is changed to 90,

then the extra amount should not be added again. This extra amount should be credited to salary only the first time the Clerk qualifies for the amount.

Provide proper constructors for all classes.

A class EmployeeTester is given to you with a main method. Use this class to test your solution's classes and methods.

---

```
public class Test4
{
	private enum Season   //private, public, protected, static
	{
	SPRING, SUMMER, WINTER, RAINY;
	}

	public static void main(String[] args)
	{
		System.out.println(Season.RAINY);
	}
}
```

---

```
//Interview Question
class Hello
{
	int x = 100;
}

enum Direction extends Hello  //error
{
	EAST, WEST, NORTH, SOUTH
}
```

```
class Test5
{
	public static void main(String[] args)
	{
		System.out.println(Direction.SOUTH);
	}
}
```

---

//All enums are by default final so can't inherit

```
enum Color
{
	RED, BLUE, PINK;
}
class Test6 extends Color
{
	public static void main(String[] args)
	{
		System.out.println(Color.RED);
	}
}
```

Every enum is by default final so we cannot inherit an enum.

---

//values() to get all the values of enum

```
class Test7
{
```

```
	enum Season
	{
	SPRING, SUMMER, WINTER, FALL, RAINY
	}

	public static void main(String[] args)
	{
		Season x []=  Season.values();

		 for(Season y : x)
		  System.out.println(y);
	}
}
-------------------------------------------------------------------------
//ordinal() to find out the order position
class Test8
{
	static enum Season
	{
	SPRING, SUMMER, WINTER, FALL, RAINY
	}


	public static void main(String[] args)
	{
		Season s1[] = Season.values();

		for(Season x : s1)
```

```
                        System.out.println(x+" order is :"+x.ordinal());
        }
}
```

---

```
//We can take main () inside an enum

enum Test9
{
        TEST1, TEST2, TEST3;  //Semicolon is compulsory

        public static void main(String[] args)
        {
                System.out.println("Enum  main method");
        }
}
```

---

```
//constant must be in first line of an enum

enum Test10
{
        public static void main(String[] args)
        {
                System.out.println("Enum  main method");
        }

         HR, SALESMAN, MANAGER;
}
```

---

```
//Writing constructor in enum
enum Season
{
	WINTER, SUMMER, SPRING, RAINY;   //All are object of type enum

	Season()
	{
		System.out.println("Constructor is executed....");
	}

}
class Test11
{
	public static void main(String[] args)
	{
		System.out.println(Season.WINTER);
		System.out.println(Season.SUMMER);

	}
}
```

---

```
//Writing constructor with message
  enum Season
	{
	  SPRING("Pleasant"), SUMMER("UnPleasent"), RAINY("Rain"), WINTER;

	String msg;
```

```
    Season(String msg)
        {
         this.msg = msg;
        }

        Season()
        {
            this.msg = "Cold";
        }

        public String getMessage()
        {
            return msg;
        }
    }
class Test12
{
    public static void main(String[] args)
    {
        Season s1[] = Season.values();

        for(Season x : s1)
            System.out.println(x+"  is :"+x.getMessage());
    }
}
```

---

```
package com.ravi.enum_demo;
```

```
public enum Season
{
  WINTER
  {
        @Override
        public String toString()
        {
              return "Winter Season";
        }
  },

  SPRING
  {
        @Override
        public String toString()
        {
              return "Spring Season";
        }
  }
}

package com.ravi.enum_demo;

public class EnumTest {

      public static void main(String[] args)
      {
            System.out.println(Season.WINTER); //toString()
```

```
                System.out.println(Season.SPRING); //toString()


        }


}
```

---

We can pass enum in switch statement

```
public class Test14
{
        enum Day
                {
                  SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
SATURDAY
                }


public static void main(String args[])
{
        Day day=Day.MONDAY;


        switch(day)
        {
         case SUNDAY:
         System.out.println("Sunday");
         break;
         case MONDAY:
         System.out.println("Monday");
         break;
```

```
            default:
            System.out.println("other day");
    }


 }
}
```

---

```
enum Movie
{
            Pathaan, sir, Hit, Geme_over, Lost; //public + static + final
}


            public class Test15
            {
             public static void main(String[] args)
             {
               System.out.println(Test15.getMovie("sir"));
             }
             public static Movie getMovie(String s)
             {
               return Enum.valueOf(Movie.class, s.toLowerCase());
             }


             public static Movie getMovie(int n){
               return Movie.values()[n];
             }
}
```

---

How many ways we can Initialize the object properties

---

The following are the ways to initialize the object properties :

---

```
public class Test
{
  int x,y;
}
```

1) At the time of declaration :

Example :

```
public class Test
{
   int x = 10;
   int y = 20;
}
```

```
Test t1 = new Test();   [x = 10  y = 20]
Test t2 = new Test();   [x = 10  y = 20]
```

Here the drawback is all objects will be initialized with same value.

---

2) By using Object Reference :

```
public class Test
{
  int x,y;
}

Test t1 = new Test();   t1.x=10;   t1.y=20;
Test t2 = new Test();   t2.x=30;   t2.y=40;
```

Here we are getting different values with respect to object but here the program becomes more complex.

---

3) By using methods :

A) First Approach (Method without Parameter)

---

```
public class Test
{
  int x,y;

  public void setData()  //All the objects will be initialized with
  {                      same value
    x = 100;  y = 200;
  }
}

Test t1 = new Test();  t1.setData();  [x = 100   y = 200]
Test t2 = new Test();  t2.setData();  [x = 100   y = 200]
```

B) Second Approach (Method with Parameter)

---

```
public class Test
{
 int x,y;

 public void setData(int x, int y)
 {
   this.x = x;
        this.y = y;
 }
}
```

```
Test t1 = new Test();  t1.setData(12,78);  [x = 12   y = 78]
Test t2 = new Test();  t2.setData(15,29);  [x = 15   y = 29]
```

Here the Drawback is initialization and re-initialization both are done in two different lines so Constructor introduced.

---

4) By using Constructor

A) First Approach (No Argument Constructor)

---

```
public class Test
{
  int x,y;
```

```
  public Test()  //All the objects will be initialized with
  {                          same value
    x = 100;  y = 200;
  }
}

 Test t1 = new Test();   [x = 100   y = 200]
 Test t2 = new Test();   [x = 100   y = 200]
```

B) Second Approach (Parameterized Constructor)

---

```
 public class Test
 {
  int x,y;

  public Test(int x, int y)
  {
    this.x = x;
        this.y = y;
  }
}

 Test t1 = new Test(12,78);   [x = 12   y = 78]
 Test t2 = new Test(15,29);   [x = 15   y = 29]
```

This is the best way to initialize our instance variable because variable initialization and variable re-initialization both will be done in the same line as well as all the objects will be initialized with different values.

C) Third Approach (Copy Constructor)

---

```
public class Manager
{
  private int managerId;
      private String managerName;

      public Manager(Employee emp)
      {
       this.managerId = emp.getEmployeeId();
       this.managerName = emp.getEmployeeName();
  }
}
```

Here with the help of Object reference (Employee class) we are initializing the properties of Manager class. (Copy Constructor)

d) By using instance block (Instance Initialize)

---

```
public class Test
{
  int x,y;

 public Test()
  {
```

```
            System.out.println(x); //100
            System.out.println(y); //200
        }

    //Instance block
        {
            x = 100;
            y = 200;
        }
```

---

5) By using super keyword :

```
class Super
{
  int x,y;

  public Super(int , int y)
  {
    this.x = x;
        this.y = y;
  }
}
class Sub extends Super
{
  Sub()
  {
    super(100,200); //Initializing the properties of super class
```

```
    }
  }
```

---

18-12-2023

---

Overriding the super class method with Anonymous inner class Approach :

---

```
package com.ravi.anonymous;


class Super
{
	public void show()
	{
		System.out.println("Super class show method!!!");
	}
}


public class AnonymousClass
{
	public static void main(String[] args)
	{
		//Anonymous inner class
		Super sub = new Super()
		{
			@Override
			public void show()
			{
				System.out.println("Sub class show method!!!");
```

```
                }
        };

        sub.show();
    }

}
```

---

Another Program Overriding the super class method with Anonymous inner class Approach :

```
package com.ravi.anonymous;

class Vehicle
{
    public void run()
    {
        System.out.println("Vehicle is Running!!!");
    }
}

public class AnonymousClassDemo
{
    public static void main(String[] args)
    {
        //Anonymous inner class approach
        Vehicle car = new Vehicle()
```

```
        {
            @Override
            public void run()
            {
                System.out.println("Car is Running!!!");
            }
        };

    //Anonymous inner class approach
    Vehicle bike = new Vehicle()
        {
         @Override
            public void run()
            {
                System.out.println("bike is Running!!!");
            }
        };
        car.run();   bike.run();

    }

}
```

---

Abstract class and abstract methods :

---

A class that does not provide complete implementation (partial implementation) is defined as an abstract class.

An abstract method is a common method which is used to provide easiness to the programmer because the programmer faces complexcity to remember the method name.

An abstract method observation is very simple because every abstract method contains abstract keyword, abstract method does not contain any method body and at the end there must be a terminator i.e ; (semicolon)

In java whenever action is common but implementations are different then we should use abstract method, Generally we declare abstract method in the super class and its implementation must be provided in the sub class.

If a class contains at least one method as an abstract method then we should compulsory declare that class as an abstract class.

Once a class is declared as an abstract class we can't create an object for that class.

*All the abstract methods declared in the super class must be overridden in the sub classes otherwise the sub class will become as an abstract class hence object can't be created for the sub class as well.

In an abstract class we can write all abstract method or all concrete method or combination of both the method.

It is used to acheive partial abstraction that means by using abstract classes we can acheive partial abstraction(0-100%).

*An abstract class may or may not have abstract method but an abstract method must have abstract class.

Note :- We can't declare an abstract method as final, private and static (illegal combination of modifiers)

---

Proggram on abstract method :

---

```
abstract class Shape
{
	public private abstract void draw();
}

class Rectangle extends Shape
{
	@Override
	public void draw()
	{
		System.out.println("Drawing Rectangle!!");
	}
}

class Square extends Shape
{
	@Override
	public void draw()
	{
		System.out.println("Drawing Square!!");
	}
}

public class ShapeDemo
{
```

```
	public static void main(String[] args)
	{
		Shape s1;

		s1 = new Rectangle(); s1.draw();
		s1 = new Square(); s1.draw();
	}
}
```

---

Interview Question :

---

```
package com.ravi.abstract_demo;

abstract class Car
{
	int speed = 120;

	public Car()
	{
		System.out.println("Car class constructor Invoked");
	}

	public void getDetails()
	{
		System.out.println("It has 4 wheels!!");
	}

	public abstract void run();
```

```
}

class Audi extends Car
{
	@Override
	public void run()
	{
		System.out.println("Running Audi car");
	}
}

public class InterviewQuestion
{
	public static void main(String[] args)
	{
		Car c = new Audi();
		System.out.println("Audi Speed is :"+c.speed);
		c.getDetails();
		c.run();
	}
}
```

Note :- In abstract class, if we have a constructor then it will be executed via sub class constructor using super keyword even we cannot create an object for abstract class i.e. super class.

---

19-12-2023

---

What is the advantage of writing constructor in the abstract class ?

---

If my abstract class contains any properties (state OR Data) then we can initialize those properties of abstract class with the help of sub class object by using super keyword.

---

The following program explains that we should override all the abstract methods in the sub classes otherwise the sub class will also become as an abstract class.

```
package com.ravi.abstract_demo;


abstract class AA
{
	public abstract void show();
	public abstract void demo();
}


abstract class BB extends AA
{
  @Override
  public void show()  //+ demo();
  {
	System.out.println("Show method implemented in BB class");
  }
}
class CC extends BB
{
	@Override
	public void demo()
	{
```

```
                System.out.println("Demo method Implemented in CC class ");

        }
}

public class AbstractDemo
{
        public static void main(String[] args)
        {
          CC c1 = new CC();
          c1.show(); c1.demo();
        }

}
```

---

Overriding abstract method by using Anonymous inner class :

---

```
package com.ravi.abstract_demo;

abstract class Animal
{
        public abstract void eat();
}

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

```
        //Anonymous inner class
        Animal horse = new Animal()
        {
                @Override
                public void eat()
                {
                 System.out.println("Horse is veg type");
                }
        };
        horse.eat();

        Animal dog = new Animal()
        {
                @Override
                public void eat()
                {
                        System.out.println("Dog is Non-Veg type");
                }
        };
        dog.eat();
       }
}
```

---

Program on abstract class and abstract method which describes the common functionality :

5 files :

---

Shape.java

---

```
package com.ravi.abstract_ex;

public abstract class Shape
{
  protected int data;

  public Shape(int data)
  {
            this.data = data;
  }

  public abstract void area();
}
```

Rectangle.java

---

```
package com.ravi.abstract_ex;

public class Rectangle extends Shape
{
  protected int b;

  public Rectangle(int l , int b)
  {
            super(l);
            this.b = b;
```

```
 }

 @Override
 public void area()
 {
          System.out.println("Area of Rectangle is :"+(data*b));
 }

}
```

Circle.java

---

```
package com.ravi.abstract_ex;

public class Circle extends Shape
{
   final double PI = 3.14;
          public Circle(int radius)
          {
                  super(radius);
          }

           @Override
           public void area()
           {
                   System.out.println("Area of Circle is :"+(PI *data*data));
           }
}
```

Square.java

---

```
package com.ravi.abstract_ex;

public class Square extends Shape
{
	public Square(int side)
	{
		super(side);
	}

	 @Override
	  public void area()
	  {
		System.out.println("Area of Square is :"+(data*data));
	  }
}
```

AbstractDemo.java

---

```
package com.ravi.abstract_ex;

public class AbstractDemo
{
	public static void main(String[] args)
	{
	  Shape s1;
```

```
        s1 = new Rectangle(5, 5); s1.area();

        s1 = new Circle(2); s1.area();

        s1 = new Square(40); s1.area();
    }

}
```

Note :- All abstract method implementation is compulsory in the sub classes.

Program on abstract method with Array concept using upcasting :

---

```
package com.ravi.abstract_example;

abstract class Animal
{
    public abstract void checkup();
}

class Bird extends Animal
{
    @Override
    public void checkup()
    {
        System.out.println("Bird check up");
    }
```

```
}
class Dog extends Animal
{
	@Override
	public void checkup()
	{
		System.out.println("Dog check up");
	}
}

class Lion extends Animal
{
	@Override
	public void checkup()
	{
		System.out.println("Lion check up");
	}
}



public class AbstractDemo
{
   public void checkAnimal(Animal [] animals)
   {
	for(Animal animal : animals)
	{
		animal.checkup();
	}
```

```
  }


        public static void main(String[] args)
        {
                AbstractDemo ab = new AbstractDemo();

                Bird []birds = {new Bird(), new Bird(), new Bird()};
                Dog []dogs = {new Dog(), new Dog()};
                Lion []lions = {new Lion(), new Lion()};

                ab.checkAnimal(birds);
                System.out.println(".......");
                ab.checkAnimal(dogs);
                System.out.println(".......");
                ab.checkAnimal(lions);

        }
}
```

---

20-12-2023

---

An interface is a keyword in java which is similar to a class.

Upto JDK 1.7 an interfcae contains only abstract method that means there is a guarantee that inside an interfcae we don't have concrete or general or instance methods.

From java 8 onwards we have a facility to write default and static methods.

By using interface we can acheive 100% abstraction concept because it contains only abstract methods.

In order to implement the member of an interface, java software people has provided implements keyword.

All the methods declared inside an interface is by default public and abstract so at the time of overriding we should apply public access modifier to sub class method.

All the variables declared inside an interface is by default public, static and final.

We should override all the abstract methods of interface to the sub class otherwise the sub class will become as an abstract class hence object can't be created.

We can't create an object for interface, but reference can be created.

By using interfcae we can acheive multiple inheritance in java.

We can achieve loose coupling using inetrface.

Note :- inside an interface we can't declare any blocks (instance, static), instance variables (No properties) as well as we can't write constructor inside an interface.

---

InterfaceDemo1.java

---

```
package com.arvi.interface_demo;

interface Moveable
```

```
{
	int SPEED = 120;  //public + static + final
	void move();   //public + abstract
}

class Car implements Moveable
{
	@Override
	public void move()
	{
	 //SPEED = 150;
		System.out.println("Running with Audi Car");
	}
}

public class InterfaceDemo1
{
	public static void main(String[] args)
	{
		System.out.println("Car Speed is :"+Moveable.SPEED);
      Moveable m = new Car();
      m.move();
	}

}
```

---

//Program that describes interface defines WHAT TO DO and implementer class will describe HOW TO DO

```
package com.arvi.interface_demo;

interface Calculator
{
	void sum(int x, int y);
	void sub(int x, int y);
	void mul(int x, int y);
}

class Developer implements Calculator
{
	@Override
	public void sum(int x, int y)
	{
		System.out.println("Sum is :"+(x+y));
	}

	@Override
	public void sub(int x, int y)
	{
		System.out.println("Sub is :"+(x-y));
	}

	@Override
	public void mul(int x, int y)
	{
		System.out.println("Mul is :"+(x*y));
```

```
	}
}

public class InterfaceDemo2
{
	public static void main(String[] args)
	{
		Calculator c1 = new Developer();
		c1.sum(12, 12);
		c1.sub(200, 100);
		c1.mul(5, 5);
	}

}
```

---

21-12-2023

---

Program on Bank using interface :

---

BankApplication.java

---

```
package com.ravi.interface_demo;

interface Bank
{
```

```
	void deposit(int amount);
	void withdraw(int amount);
}
class Customer implements Bank
{
	private double balance = 10000;

	@Override
	public void deposit(int amount)
	{
		if(amount<=0)
		{
			System.out.println("Amount can't be deposited");
		}
		else
		{
			balance = balance + amount;
			System.out.println("After Deposit :"+balance);
		}
	}

	@Override
	public void withdraw(int amount)
	{
		balance = balance - amount;
		System.out.println("After Withdraw :"+balance);
	}
}
```

```
public class BankApplication
{
	public static void main(String[] args)
	{
		Bank b = new Customer();
		b.deposit(15000);
		b.withdraw(5000);

	}

}
```

---

IQ

--

How to achieve loose coupling using interfaces :

---

Loose Coupling :- If the degree of dependency from one class object to another class is very low then it is called loose coupling.

Tightly coupled :- If the degree of dependency of one class to another class is very high then it is called Tightly coupled.

According to IT industry standard we should always prefer loose coupling so the maintenance of the project will become easy.

---

Program on loose coupling using interface :

---

7 files :

---

HotDrink.java

---

```
package com.ravi.loose_coupling;

public interface HotDrink
{
  public abstract void prepare();
}
```

Tea.java

---

```
package com.ravi.loose_coupling;

public class Tea implements HotDrink
{
	@Override
	public void prepare()
	{
		System.out.println("Preparing Tea");
	}

}
```

Coffee.java

---

```
package com.ravi.loose_coupling;
```

```
public class Coffee implements HotDrink
{
	@Override
	public void prepare()
	{
		System.out.println("Preparing Coffee");
	}

}
```

Horlicks.java

---

```
package com.ravi.loose_coupling;

public class Horlicks implements HotDrink
{
	@Override
	public void prepare()
	{
		System.out.println("Preparing Horlicks");
	}

}
```

Restaurant.java

---

```
package com.ravi.loose_coupling;
```

```
public class Restaurant
{
  public static void createObject(HotDrink hd) //hd = tea object
  {
            hd.prepare();
  }
}
```

Boost.java

---

```
package com.ravi.loose_coupling;

public class Boost implements HotDrink
{
        @Override
        public void prepare()
        {
                System.out.println("Preapring Boost");

        }

}
```

Main.java

---

```
package com.ravi.loose_coupling;

public class Main {
```

```
	public static void main(String[] args)
	{
		Restaurant.createObject(new Tea());
		Restaurant.createObject(new Coffee());
		Restaurant.createObject(new Horlicks());
		Restaurant.createObject(new Boost());
	}

}
```

---

Note :- We can also take return type of the method as an interface so that method can return the object of all the sub classes which are implementing from that particular interface.

```
public HotDrink accept()
{

  return  new Tea(); OR new Coffee(); OR null OR new Horlicks(); .....(future)
}
```

---

Multiple inheritance by using interface :

---

Upto java 7, interface does not contain any method body that means all the methods are abstract method so we can achieve multiple

inheritance by providing the logic in the implementer class as shown in the below program (Diagram :- 21-DEC-23)

MultipleInheritance.java

---

```
package com.ravi.interface_mi;

interface A
{
	void m1();
}
interface B
{
	void m1();
}

class Implementer implements A,B
{
	@Override
	public void m1()
	{
		System.out.println("Multiple Inheritance using interface");
	}

}
public class MultipleInheritance
{
	public static void main(String[] args)
	{
		Implementer i = new Implementer();
		i.m1();
```

```
	}

}
```

---

Extending an interface :

---

We can extend one interface from another interface, we cannot implement one interface from another.

```
package com.ravi.interface_mi;


interface Alpha
{
	void m1();
}
interface Beta
{
	void m2();
}
interface Gamma extends Alpha, Beta
{
	void m3();
}
class Demo implements Gamma
{

	@Override
```

```
	public void m1()
	{
	System.out.println("M1");

	}

	@Override
	public void m2()
	{
		System.out.println("M2");
	}

	@Override
	public void m3()
	{
		System.out.println("M3");
	}
}

public class ExtendingInterface {

	public static void main(String[] args)
	{
		Demo d1 = new Demo();
		d1.m1(); d1.m2(); d1.m3();

	}
```

}

---

22-12-2023

---

Interface from JDK 1.8 onwards :

---

Upto JDK 1.7 we can use only abstract methods inside an interface, as we know all the abastrct methods must be overridden in the sub class otherwise the sub class will become as an abstract class.

This facility of abstract methods leads to maintenance problem because if we add any new abstract method inside an existing interface then that method has to override by all the sub classes or the classes which are implementing from that particular interface.

To avoid this boundation problem java software people has introduced default and static method inside an interface so from JDK 1.8 onwards we can define the body of the method inside an interface by declaring those method as default method and static method or both.

---

What is default Method inside an interface?

---

default method is just like concrete method which contains method body and we can write inside an interface from java 8 onwards.

default method is used to  provide  specific implementation for the implementer classes which are implmenting from interface because we can override default method inside the sub classes to provide our own specific implementation.

*By using default method there is no boundation to override the default method in the sub class, if we really required it then we can override to provide my own implementation.

by default, default method access modifier is public so at the time of overriding we should use public access modifier.

---

4 files :

---

Vehicle.java(I)

---

```
package com.ravi.java_8;


public interface Vehicle
{
   void run();
   void horn();


  default void digitalMeter() //java 1.8
  {
            System.out.println("Digital Meter");
  }
}
```

Car.java

---

```
package com.ravi.java_8;


public class Car implements Vehicle
{
```

```
	@Override
	public void run()
	{
		System.out.println("Car is Running");
	}

	@Override
	public void horn()
	{
		System.out.println("POP POP");
	}
   @Override
	public void digitalMeter() //java 1.8
	{
	 System.out.println("Car is having Digital Meter");
	}
}
```

Bike.java

---

```
package com.ravi.java_8;

public class Bike implements Vehicle
{
	@Override
	public void run()
	{
```

```
		System.out.println("Bike is Running");
	}
	@Override
	public void horn()
	{
		System.out.println("PEEP PEEP");

	}
}
```

Main.java

---

```
package com.ravi.java_8;

public class Main
{
  public static void main(String[] args)
  {
		Vehicle v;
		v = new Car(); v.run(); v.horn(); v.digitalMeter();
		v = new Bike(); v.run(); v.horn();
  }
}
```

---

The following program explains that default methods are having low priority than normal methods (Concrete Method). class is having more power than interface.

```
package com.ravi.priority;
```

```
interface A
{
	default void m1()
	{
		System.out.println("default method of A interface");
	}
}
class B
{
	public void m1()
	{
	 System.out.println("Concrete method of B class");
	}
}

class C extends B implements A
{
}

public class ClassInterfacePriority
{
	public static void main(String[] args)
	{
		C c1 = new C();
		c1.m1();

	}
```

```
}
```

---

Can we override Object class method using default method?

---

No, we cannot override object class method as a default method inside an interface.

```
interface I
{
	default String toString()
	{
		return "";
	}
}
public class Test
{
	public static void main(String[] args)
	{
		System.out.println("Hello World!");
	}
}
```

Note :- The above program will generate compilation error because we cannot override Object class method as a default method inside an interface.

---

Multiple Inheritance using default method :

---

Multiple inheritance is possible in java by using default method inside an interface, here we need to use super keyword to differenciate the super interface methods.

```
package com.ravi.multiple_inheritance;


interface A
{
	default void m1()
	{
		System.out.println("m1 method of interface A");
	}
}
interface B
{
	default void m1()
	{
		System.out.println("m1 method of interface B");
	}
}
class C implements A,B
{
 @Override
 public void m1()
 {
	A.super.m1();
	B.super.m1();
	System.out.println("Overridden");
 }
}
```

```
public class MultipleInheritance
{
	public static void main(String[] args)
	{
		C c1 = new C();
		c1.m1();
	}

}
```

---

What is static method inside an interface?

---

We can define static method inside an interface from java 1.8 onwards.

static method is only available inside the interface where it is defined that means we cannot invoke static method from the implementer classes.

It is used to provide common functionality which we can apply/invoke from any ELC class.

---

Calculator.java(I)

---

```
package com.ravi.multiple_inheritance;

public interface Calculator
{
	public static int doSum(int x, int y)
	{
		return (x+y);
```

```
	}

	public static int doSub(int x, int y)
	{
		return (x-y);
	}
}
```

Main.java(C)

---

```
package com.ravi.interface_demo;

import com.ravi.multiple_inheritance.*;
public class Main {

	public static void main(String[] args)
	{
	 int sum = Calculator.doSum(12, 12);
	 System.out.println("Sum is :"+sum);

	 int sub = Calculator.doSub(200, 100);
	 System.out.println("Sub is :"+sub);
	}
}
```

---

```
interface Callable
{
	public static void access()
```

```
	{
		System.out.println("static method available inside interface");
	}
}
public class StaticDemo2 implements Callable
{
	public static void main(String[] args)
	{
		Callable.access();

		StaticDemo2.access(); //error

	StaticDemo2 sm = new StaticDemo2();
			sm.access(); //error
	}
}
```

Note :- In the above program we will get compilation error because static method is available to Callable interface only so, implementer class cannot invoke static method access.

---

```
interface Vehicle
{
	static void move()
	{
		System.out.println("Static method of Vehicle");
	}
}
class StaticMethod1 implements Vehicle
```

```
{
        public static void main(String[] args)
        {
           Vehicle.move();
                move(); //error

                StaticMethod1.move(); //error

     StaticMethod1 sm = new StaticMethod1();
                sm.move(); //error

        }
}
```

---

23-12-2023

---

How static method of an interface is different from class static method?

Interface Static Method:

---

a) Accessible using the interface name.

b) Cannot be overridden by implementing classes.

c) Can be called using the interface name only.

Class Static Method:

---

a) Accessible using the class name.

b) Can be hidden (not overridden) in subclasses by redeclaring a static method with the same signature.

c) Can be called using the super class, sub class name as well as sub class object also as shown in the program below.

---

Overriding interface method using anonymous inner class :

---

```
package com.ravi.anonymous;


interface Student
{
	void writeExam();
}


public class AnonymousInterfaceDemo
{
	public static void main(String[] args)
	{
		//Anonymous inner class
		Student science = new Student()
		{
			@Override
			public void writeExam()
			{
				System.out.println("Science Student Exam!!");

			}

		};
```

```
        Student commerce = new Student()
        {
            @Override
            public void writeExam()
            {
                System.out.println("Commerce Student Exam!!");

            }

        };

        science.writeExam();
        commerce.writeExam();

    }

}
```

---

What is a functional interface?

---

If an interface contains exactly one abstract method then it is called Functional interface.

It is represented by @FunctionalInterface annotation.

A functional interface may conatin n number of default and static method but it must have only one abstract method (SAM) [Single Abstract Method]

```
package com.ravi.anonymous;

@FunctionalInterface
interface Vehicle
{
	void run();
}

public class AnonymousInterfaceDemo1
{
	public static void main(String[] args)
	{
		Vehicle car = new Vehicle()
		{
			@Override
			public void run()
			{
				System.out.println("Car is Running");

			}
		};

		car.run();
	}

}
```

---

Lambda Expression :

---

It is a new feature introduced in java from JDK 1.8 onwards.

It is an anonymous function i.e function without any name.

In java it is used to enable functional programming.

It is used to concise our code as well as we can remove boilerplate code.

It can be used with functional interface only.

If the body of the Lambda Expression contains only one statement then curly braces are optional.

We can also remove the variables type while defining the Lambda Expression parameter.

If the lambda expression method contains only one parameter then we can remove () symbol also.

In Lambda return statement is also optional.

Independently Lamda Expression is not a statement.

It requires a target variable i.e functional interface reference.

Lamda target can't be class or abstract class, it will work with functional interface only.

---

Lambda1.java

---

```
package com.ravi.lambda;

@FunctionalInterface
interface Printable
{
	void print();  //SAM
```

```
}

public class Lambda1
{
	public static void main(String[] args)
	{
	 Printable p1 = ()-> System.out.println("Printing");
	 p1.print();

	}

}
```

Lambda2.java

---

```
package com.ravi.lambda;

interface Calculate
{
	void doSum(int x, int y);
}
public class Lambda2
{
	public static void main(String[] args)
	{
	 Calculate c1 = (c,d)->
	 {
		System.out.println("Sum is :"+(c+d));
```

```
        };
        c1.doSum(15, 15);


    }

}
```

Lambda3.java

---

```
package com.ravi.lambda;

@FunctionalInterface
interface Length
{
    public int getLength(String str);
}

public class Lambda3
{
    public static void main(String[] args)
    {
        Length l = str -> str.length();

        System.out.println(l.getLength("India"));
    }

}
```

---

26-12-2023

---

```
@FunctionalInterface
interface Moveable
{
	void move();  //SAM (Single Abstract Method)
}

public class Lambda1
{
	public static void main(String[] args)
	{
		Moveable car = ()-> System.out.println("Car is moving ");

		Moveable bike = ()-> System.out.println("Bike is moving ");

		Moveable bus = ()-> System.out.println("Bus is moving ");

		car.move();  bike.move();  bus.move();
	}
}
```

---

```
@FunctionalInterface
interface Calculate
{
	 void add(int a, int b, double c);
}
```

```
public class Lambda2
{
	public static void main(String[] args)
	{
	  Calculate calc = (x,  y,  z) ->  System.out.println("Sum is :"+(x+y+z));

		calc.add(12,12,56.89);
	}
}
```

---

```
import java.util.Scanner;

@FunctionalInterface
interface Length
{
	 int getLength(String str);
}

public class Lambda3
{
	public static void main(String[] args)
	{
		Length l = str -> str.length();

		Scanner sc = new Scanner(System.in);
		System.out.print("Enter your Name :");
		String name = sc.next();
		System.out.println("Your Name length is :"+l.getLength(name));
```

```
	}
}
```

---

```
package com.ravi.anonymous_thread;

import java.util.Scanner;

@FunctionalInterface
interface Name
{
	 boolean isStartWithR(String str);
}

public class Lambda4
{
	public static void main(String[] args)
	{
		Name n =  str -> str.startsWith("R");

		Scanner sc = new Scanner(System.in);
		System.out.print("Enter your Name :");
		String name = sc.next();
		System.out.println("Your name starts with 'R' :"+n.isStartWithR(name));
		sc.close();
	}

}
```

---

Method reference (Introduction)

---

It is an improvement over Lambda because in lambda we need to write the body of the method at the time of writing Lambda Expression.

On the other hand by using Method reference we can simply refere to the method by using :: (double colon operator) which is already available in the project.

Worker.java

---

```
package com.ravi.method_reference;

@FunctionalInterface
public interface Worker
{
  void work();
}
```

Employee.java

---

```
package com.ravi.method_reference;

public class Employee
{
 public void work()
 {
        System.out.println("Employee is Working");
 }
```

```
}
```

Main.java

---

```
package com.ravi.method_reference;

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

		//Using Lambda
		Worker w1 = ()-> System.out.println("Employee working");
		w1.work();

		//Using Method Reference
		Worker w2 = new Employee()::work;
		w2.work();


	}

}
```

---

Working with predefined functional interfaces :

---

What is type parameter?

---

It is a technique through which we can make our application indepenedent of type. It is represented by <T>

In java we can pass Wrapper classes as well as User-defined class to this type parameter.

We cannot pass any primitive type to this type parameter.

```
package com.ravi.type_parameter;

class Accept<T>
{
	private T x; //T is type parameter and x is a variable

	public Accept(T x)
	{
		super();
		this.x = x;
	}

	public T getX()
	{
		return x;
	}
}

public class TypeParameter
```

```
{
	public static void main(String[] args)
	{
		Accept<Integer> intType = new Accept<Integer>(100);
		System.out.println("Integer Object is :"+intType.getX());

		Accept<Double> doubleType = new Accept<Double>(100.78);
		System.out.println("Double Object is :"+doubleType.getX());

		Accept<Boolean> booleanType = new Accept<Boolean>(true);
		System.out.println("Boolean Object is :"+booleanType.getX());

		Accept<Customer> custType = new Accept<Customer>(new
Customer());

   Customer customer = custType.getX();
   customer.m1();
   System.out.println(customer);
	}
}

class Customer
{
	@Override
	public String toString()
	{
		return "Batch 26";
	}
```

```
        public void m1()
        {

                System.out.println("m1 method of Customer Object");
        }
}
```

---

Working with predefined functional interfaces :

---

In order to help the java programmer to write concise  java code in day to day programming java software people has provided the following predefined functional interfaces

1) Predicate<T>

2) Consumer<T>

3) Function<T,R>

4) Supplier<T>

5) BiPredicate<T,U>

6) BiConsumer<T, U>

7) BiFunction<T,U,R>

Note :-

---

All these predefined functional interfaces are provided as a part of java.util.function sub package.

Predicate<T> funactional interface  :

---

It is a predefined functional interface available in java.util.function sub package.

It contains an abstract method test() which takes type parameter <T> and returns boolean value. The main purpose of this interface to test one argument boolean expression.

```
@FunctionalInterface
public interface Predicate<T>
{
  boolean test(T x);
}
```

Note :- Here T is a "type parameter" and it can accept any type of User defined class as well as  Wrapper class like Integer, Float, Double and so on.

We can't pass primitive type.

---

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

public class PredicateDemo
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);

    //My name starts with R or not
		Predicate<String> checkName =  str -> str.startsWith("R");
```

```
    System.out.print("Enter your Name :");
            String name = sc.nextLine();

    boolean isStart = checkName.test(name);
            System.out.println("Your name starts with R ? :"+isStart);



            //Check a number is even or odd
            Predicate<Integer> evenOrOdd =  x -> x%2==0;
            System.out.print("Enter a Number :");
            int num = sc.nextInt();
            System.out.println(num +" is Even ? :"+evenOrOdd.test(num));



        }
}
```

---

```
package com.ravi.co_variant;

import java.util.Scanner;
import java.util.function.Predicate;

public class PredicateDemo2 {

        public static void main(String[] args)
        {
         //Verify my name is Ravi or not
         Predicate<String> verifyName =  str -> str.equals("Ravi");
```

```
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter your Name :");
		String name = sc.next();
		System.out.println("Are U Ravi ?"+verifyName.test(name));
		sc.close();

	}

}
```

---

WAP to verify whether a person is eligible 4 voting or not ?

---

```
package com.ravi.predicate_interface;

import java.util.function.Predicate;

public class PredicateDemo3 {

	public static void main(String[] args)
	{
		Predicate<Integer> p = age -> age >=18;
		System.out.println("Person is eligible for vote :"+p.test(16));

	}
}
```

---

Consumer<T> functional interface :

---

It is a predefined functional interface available in java.util.function sub package.

It contains an abstract method accept() and returns nothing. It is used to accept the parameter value or consume the value.

```
@FunctionalInterface
public interface Consumer<T>
{
   void accept(T x);
}
```

---

```
package com.ravi.co_variant;

import java.util.function.Consumer;

public class ConsumerDemo {

        public static void main(String[] args)
        {
          Consumer<Integer> intType =  x -> System.out.println(x);
          intType.accept(12);

          Consumer<String> stringType = x -> System.out.println(x);
    stringType.accept("NIT");

    Consumer<Student> studentType = x -> System.out.println(x);
    studentType.accept(new Student(111, "Rohit"));
```

```
        }

}
```

---

Function<T,R> functional interface :

---

Type Parameters:

T - the type of the input to the function.

R - the type of the result of the function.

It is a predefined functional interface available in java.util.function sub package.

It provides an abstract method apply that accepts one argument(T) and produces a result(R).

Note :- The type of T(input) and the type of R(Result) both will be decided by the user.

```
@FunctionalInterface
public interface Function<T,R>
{
  R apply(T x);
}
```

---

Using Function find the square of the number.

```
package com.ravi.function;

import java.util.Scanner;
```

```
import java.util.function.Function;

public class FunctionDemo
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		Function<Integer,Integer> fn =  x -> x*x;

		System.out.print("Enter a Number :");
		int num = sc.nextInt();
		System.out.println("Square of "+num+" is :"+fn.apply(num));
		sc.close();

	}

}
```

---

Using Function<T,R> get the length of the name

```
package com.ravi.function;

import java.util.Scanner;
import java.util.function.Function;

public class FunctionDemo2 {

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

```
        {
          Scanner sc = new Scanner(System.in);
    Function<String,Integer> fn = str -> str.length();

    System.out.print("Enter your Name :");
    String name = sc.nextLine();
    System.out.println("Length of "+name+ " is :"+fn.apply(name));
    sc.close();
        }

}
```

---

Using Function verify the name starts with particular letter or not?

```
package com.ravi.function;

import java.util.function.Function;

public class FunctionDemo3 {

        public static void main(String[] args)
        {
          Function<String,Boolean> fn = str -> str.startsWith("R");
          System.out.println("Your Name starts with R ?"+fn.apply("Ravi"));

        }

}
```

---

28-12-2023

---

Supplier<T> prdefined functional interface :

---

It is a predefined functional interface available in java.util.function sub package. It provides an abstract method get() which does not take any argument but produces/supply a value of type T.

```
@FunctionalInterface
public interface Supplier<T>
{
  T get();
}
```

---

Here in the program get() method is returning String type

```
package com.ravi.supplier;

import java.util.function.Supplier;

public class SupplierDemo
{
	public static void main(String[] args)
	{
		Supplier<String>  sup = ()->  15+14+"Ravi"+20+20;
		System.out.println(sup.get());
	}
}
```

---

Here in the program get() method is returning Employee type

```
package com.ravi.supplier;

import java.util.function.Supplier;

class Employee
{
	private Integer employeeId;
	private String employeeName;
	private Double employeeSalary;

	public Employee(Integer employeeId, String employeeName, Double employeeSalary) {
		super();
		this.employeeId = employeeId;
		this.employeeName = employeeName;
		this.employeeSalary = employeeSalary;
	}

	public void getEmployeeSalary()
	{
		System.out.println("Employee Salary is :"+employeeSalary);
	}

	@Override
	public String toString() {
```

```
		return "Employee [employeeId=" + employeeId + ", employeeName=" +
employeeName + ", employeeSalary="
				+ employeeSalary + "]";
	}
}

public class SupplierDemo1
{
	public static void main(String[] args)
	{
		Supplier<Employee> empSup = ()->
		{
			Employee e1 = new Employee(1, "Raj", 40234.90);
			return e1;
		};

		Employee emp = empSup.get();
		emp.getEmployeeSalary();
		System.out.println(".....");
		System.out.println(emp);
	}

}
```

---

Here in the program get() method is returning Player type

```
package com.ravi.supplier;

import java.util.function.Supplier;
```

```
class Player
{
	private int playerId;
	private String playerName;

	public Player(int playerId, String playerName)
	{
		super();
		this.playerId = playerId;
		this.playerName = playerName;
	}

	@Override
	public String toString() {
		return "Player [playerId=" + playerId + ", playerName=" + playerName + "]";
	}
}

public class SupplierDemo2
{
	public static void main(String[] args)
	{
	 Supplier<Player> p1 = ()-> new Player(111,"Virat");
	 System.out.println(p1.get());

	}
```

}

---

BiPredicate<T,U> functional interface :

---

It is a predefined functional interface available in java.util.function sub package.

It is a functional interface in Java that represents a predicate (a boolean-valued function) OF TWO ARGUMENTS.

The BiPredicate interface has method named test, which takes two parameters and returns a boolean value, basically this BiPredicate is same with the Predicate, instead, it takes 2 arguments for the test.

```
@FunctionalInterface
public interface BiPredicate<T, U>
{
   boolean test(T t, U u);
}
```

Type Parameters:

T - the type of the first argument to the predicate

U - the type of the second argument the predicate

---

```
import java.util.function.*;
public class Lambda11
{
        public static void main(String[] args)
   {
```

```
        BiPredicate<String, Integer> filter = (x, y) ->
                        {
           return x.length() == y;
        };

        boolean result = filter.test("Ravi", 4);
        System.out.println(result);

        result = filter.test("Hyderabad", 10);
        System.out.println(result);
             }
}
```

---

```
import java.util.function.BiPredicate;

public class Lambda12
{
  public static void main(String[] args)
  {
    // BiPredicate to check if the sum of two integers is even
    BiPredicate<Integer, Integer> isSumEven = (a, b) -> (a + b) % 2 == 0;

        System.out.println(isSumEven.test(2, 3));
        System.out.println(isSumEven.test(5, 7));
    }
}
```

---

BiConsumer functional interface :

---

It is a predefined functional interface available in java.util.function sub package.

It is a functional interface in Java that represents an operation that accepts two input arguments and returns no result.

It takes a method named accept, which takes two parameters and performs an action without returning any result.

```
@FunctionalInterface
public interface BiConsumer<T, U>
{
   void accept(T t, U u);
}
```

----------JAVA ALWAYS WORKS WITH PASS BY VALUE ONLY.

---

```
import java.util.function.BiConsumer;

public class Lambda13
{
  public static void main(String[] args)
  {
    BiConsumer<Integer, String> updateVariables = (num, str) -> {
        num = num * 2;
        str = str.toUpperCase();
        System.out.println("Updated values: " + num + ", " + str);
      };
```

```
        int number = 15;
        String text = "nit";

        updateVariables.accept(number, text);

        // Values after the update (note that the original values are unchanged)
        System.out.println("Original values: " + number + ", " + text);
    }
}
```

---

BiFunction Functional interface :

---

It is a predefined functional interface available in java.util.function sub package.

It is a functional interface in Java that represents a function that accepts two arguments and produces a result.

The BiFunction interface has a method named apply that takes two arguments and returns a result.

```
@FunctionalInterface
public interface BiFunction<T, U, R>
{
   R apply(T t, U u);
}
```

---

```
import java.util.function.BiFunction;
```

```
public class Lambda14
{
  public static void main(String[] args)
 {
      // BiFunction to concatenate two strings
   BiFunction<String, String, String> concatenateStrings = (str1, str2) -> str1 + str2;

      String result = concatenateStrings.apply("Hello", " Java");
      System.out.println(result);
 }
}
```

---

Can we write private method inside an inetrafce?

Yes, From java 9 onwards we can also write private static and private non-static methods inside an interface.

These private methods will improve code re-usability inside interfaces.

For example, if two default methods needed to share common and confidential code, a private method would allow them to do so, but without exposing that private method to it's implementing classes.

Using private methods in interfaces have four rules :

1) private interface method cannot be abstract.

2) private method can be used only inside interface.

3) private static method can be used inside other static and non-static interface methods.

4) private non-static methods cannot be used inside private static methods.

```
package com.ravi.supplier;

interface I1
{
	public abstract void m1();

	public default void m2()
	{
		System.out.println("Default method");
		m4();
		m5();
	}

	public static void m3()
	{
		System.out.println("static method");

	}

	private static void m4()
	{
		System.out.println("Private static method");
	}

	private void m5()
	{
```

```
                System.out.println("Private non-static method");
        }
}
class Implementer implements I1
{
        @Override
        public void m1()
        {
                System.out.println("m1 method implemented");
        }
}

public class PrivateMethodInsideInterface {

        public static void main(String[] args)
        {
                I1 i = new Implementer();
                i.m1();
                i.m2(); //def method
                I1.m3();
        }

}
```

Note :- private methods are introduced to share the common code among number of default methods.

---

What is marker interface ?

---

An interface which does not contain any method and field is called marker interface. In other words, an empty interface is known as marker interface or tag interface.

*It describes run-time type information about objects, so the JVM have additional information about the object. [like object is clonable OR object is serializable OR Object is RandomAccess]

Example :

---

```
public interface Drawable  //Marker interface
{
}
```

Note :-In java we have Clonable and Serializable, RandomAccess are predefined marker interface.

---

Does An Interface extends Object Class In Java.?

---

No Interface does not inherits Object class,but it provide accessibility to all methods of Object class.

This is because, for every public method in Object class, there is an implicit abstract and public method declared in every interface which does not have direct super interfaces. (Java language Specification 9.2 about interface members)

```
package com.ravi.object_interface;

interface A
{
```

```
}
public class ELC
{
	public static void main(String[] args)
	{
		A a1 = null;
		a1.equals(null);
		a1.toString();
		a1.hashCode();
		a1.getClass();

	}

}
-----------------------------------------------------------------
--
//Object class non final method we can re-declare

interface A
{
   @Override
   public boolean equals(Object obj);

   @Override
   public int hashCode();

   @Override
```

```
   public String toString();


}
class Test
{
        public static void main(String[] args)
        {
                System.out.println("Hello");
        }
}
```

---

Can a functional interface contains the method of Object class?

---

Yes, Functional interface contains the method of Object class.

```
package com.ravi.functional_interface;


@FunctionalInterface
public interface Callable
{
  public abstract void call();


  public String toString();


  public int hashCode();


  public boolean equals(Object obj);
```

}

From the avove interface, It is clear that inside a Functional interface we can re-declare the method of Object class.

---

*What is difference between abstract class and interface ?

---

The following are the differences between abstract class and interface.

1) An abstract class can contain instance variables but interface variables are by default public , static and final.

2) An abstract class can have state (properties) of an object but interface can't have state of an object.

3) An abstract class can contain constructor but inside an interface we can't define constructor.

4) An abstract class can contain instance and static blocks but inside an interface we can't define any blocks.

5) Abstract class can't refer Lambda expression but using Functional interface we can refer Lambda Expression.

6) We can write concrete method inside an abstract class but inside an interface we can't write concrete public method, only abstract , default, static and private methods are allowed.

----------OOPS ENDED----------

Class loader sub system with JVM Architecture :

---

The three main components of JVM

1) class loader sub system

2) Runtime Data Areas

3) Execution engine

class loader sub system internally performs 3 task

a) Loading  b) Linking  c) Initialization  (Diagram 9th JAN)

Loading:

---

In order to load the required .class file, JVM makes a request to class loader sub system. The class loader sub system follows delegation hierarchy algorithm to load the required .class files from different areas.

To load the required .class file we have 3 different kinds of class loaders.

1) Bootstrap/Primordial class loader

2) Extension/Platform class loader

3) Application/System class loader

Bootstrap/Primordial class Loader :-

---

It is responsible to load the required .class file from java API that means all the predfined classes (provided by java software people) .class file will be loaded by Bootstrap class loader.

It is the super class of Extension class loader as well as It has the highest priority among all the class loader.

It will load the .class file from the following Path

C-> Program files -> java -> jdk -> lib -> jrt-fs.jar

Extension/Platform class Loader :-

---

It is responsible to load the required .class files from ext (extension) folder. Inside the extension folder we have jar file(Java level zip file) given by some third party or user defined jar file.

It is the super class of Application class loader as well as It has more priority than Application class loader.

Note :- Command to create the jar file

```
jar cf NIT.jar  FileName.class
```

Here FileName.class will be placed inside the jar file.

Application/System class Loader :-

---

It is responsible to load the required .class file from class path level i.e Environment variable. It has lowest priority as well as It is the sub class of Extension/Platform class loader.

Note :-

---

If all the class loaders are failed to load the .class file into JVM memory then we will get a Runtime exception i.e java.lang.ClassNotFoundException.

---

30-12-2023

---

getClassLoader() is a predefined method of class called Class available in java.lang package and it's return type is ClassLoader.

getParent() is a predefined method of ClassLoader class in java, available in java.lang package. It is an abstract class.

```
package com.ravi.oop;

public class Test
{
   public static void main(String[] args)
   {
        System.out.println("Test.class file is loaded by :"+Test.class.getClassLoader());

        System.out.println("The super class for Application class loader is
:"+Test.class.getClassLoader().getParent());
```

```
        System.out.println("The super class for Platform class loader is
:"+Test.class.getClassLoader().getParent().getParent()); //null


   }
}
```

Note :- Bootstarp class loader implementation is not provided by java software people becuase it is used to load all predefined .class file from java API.

---

The following program explains that any .class file return type is

java.lang.Class

```
package com.ravi.jvm_architecture;


class Customer{}

class Employee{}


public class Loader {


        public static void main(String[] args)

        {

                Class cls = Customer.class;

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


                cls = Employee.class;

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


        }
```

}

---

Linking :

---

verify :-

---

It ensures the correctness of the .class files, If any suspicious activity is there in the .class file then It will stop the execution immediately by throwing an exception i.e java.lang.VerifyError.

There is something called ByteCodeVerifier(Component of JVM), responsible to verify the loaded .class file i.e byte code. Due to this verify module JAVA is highly secure language.

java.lang.VerifyError is the sub class of java.lang.linkageError

---

prepare: (static variable memory allocation + Initialization )

---

It will allocate the memory for all the static data members, here all the static data member will get the default values so if we have

static int x = 100;

then for variable x  memory will be allocated and now it will initialize with default value i.e 0.

---

Resolve :-

---

All the symbolic references will be converted to direct references or actual reference.

javap -verbose  FileName.class

Note :- By using above command we can read the internal details of .class file.

---

Initialization :-

---

In Initialization, all the static data member will get their actual (Original) value as well as if any static block is present in the class then the static block will start executing from here.

01-01-2024

---

Static Block :

---

It is a very special block which is automatically executed at the time

of loading the .class file into JVM memory by class loader sub system.

Example :

```
static
{

}
```

The .class file will be loaded on the request of user into JVM memory only once so, static block will be executed only once.

The main purpose of static block to initialize the static variable of the class.

It is executed before any static method i.e before main method.

If we try to initialize the static variable inside static block even before static variable declaration then it is possible (due to prepare phase writing operation is possible). On the other hand if we try to access static variable before declaration inside the static block then we will get compilation error Illegal Forward Reference.

[Initialization is possible but accessing is not possible without declaration]

If we have multiple static blocks in the class then it will executed

from top to bottom. (According to the order)

---

```
//static block
class Foo
{
	Foo()
	{
		System.out.println("No Argument constructor..");
	}

	{
		System.out.println("Instance block..");
	}

	static
	{
		System.out.println("Static block...");
	}
```

```
}
public class StaticBlockDemo
{
	public static void main(String [] args)
	{
		System.out.println("Main Method Executed ");


	}
}
```

Note :- static block is not executed because Foo.class file is not loaded into the memory.

---

```
class Test
{
	static int x;

	static
	{
		x = 100;
		System.out.println("x value is :"+x);
	}

	static
	{
		x = 200;
		System.out.println("x value is :"+x);
	}
```

```
	static
	{
		x = 300;
		System.out.println("x value is :"+x);
	}

	static
	{
		x = 400;
		System.out.println("x value is :"+x);
	}
}
public class StaticBlockDemo1
{
	public static void main(String[] args)
	{
		System.out.println(Test.x);
	}
}
```

Note :- static blocks are executed from top to bottom.

---

```
class Foo
{
	static int x;

   static
```

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

}

public class StaticBlockDemo2
{
	public static void main(String[] args)
	{
		new Foo();
	}
}
```

Note :- Just like instance variables, static variables are also initialized with default values.

1) Instance variable : Automatically Initialized with default value at the process of Instantiation and Initialization(new Test())

2) static variable : Automatically Initialized with default value at the time of loading the .class file at prepare module.

3) Local Variable : Developer responsibility to initialize

4) Parameter Variable :- End User responsibility to pass the value.

---

static blank final variable :-

---

It must be initialized at the time of declaration OR through static block only, the prepare phase cannot initialize with default value.

instance blank final variable :-

---

It must be initialized at the time of declaration OR through constructor as well as instance block. here defualt constructor can't initialize with default value.

```
class Demo
{
	// static blank final variable
	final static int a;

	static
	{
		a = 10; //Initialization is compulsory because final blank static
		System.out.println(a);
	}
}
public class  StaticBlockDemo3
{
	public static void main(String[] args)
		{
	    System.out.println("a value is :"+Demo.a);
	  }
}
```

---

```
//IQ
class  A
```

```
{
	static
	{
		System.out.println("A");
	}


	{
		System.out.println("B");
	}

	A()
	{
		System.out.println("C");
	}
}
class B extends A
{
	static
	{
		System.out.println("D");
	}


	{
		System.out.println("E");
	}
```

```
        B()
        {
                System.out.println("F");
        }

}
public class StaticBlockDemo4
{
        public static void main(String[] args)
        {
                new B();
        }
}

//ADBCEF
-----------------------------------------------------------------------
//illegal forward reference

class Demo
{
   static
        {
                i = 10; //valid, Initialization is possible
                System.out.println(i); //Invalid compiler syntax is withot
   }             //declaration accessing is not possible

   static int i;
}
```

```
public class StaticBlockDemo5
{

   public static void main(String[] args)
         {
      System.out.println(Demo.i);
         }
}
```

Note :- We will get CE becuase we are trying to read/access the static variable value before declaration [Illegal Forward Reference]

---

```
class Demo
{
   static
         {
                  i = 100; //before defining initialization is possible
   }

   static int i;
}

public class StaticBlockDemo6
{

   public static void main(String[] args)
         {
```

```
                System.out.println(Demo.i);
        }
}
```

Note :- Without declaration, initialization is possible because prepare phase has already allocated the memory for static variable and initailized with default value.

---

```
class StaticBlockDemo7
{
        static
        {
    System.out.println("Static Block");
        }

        public static void main(String[] args)
        {
                System.out.println("Main Method");
        }
}
```

---

Can we execute a java program without main method?

---

No, It was possible upto java 1.6 but from java 1.7 onwards, JVM always verify the presence of main method at the time of loading the .class file into JVM memory.

```
//This program is possible upto java 1.6v
public class WithoutMain
{
  static
```

```
        {
        System.out.println("Hello world");
        System.exit(0);
        }
}
```

---

How many ways we can explicitly load the .class file into JVM memory?

---

There are so many ways to explicitly load the .class file into JVM memory, The common ways are as follows :

1) By using Java command

2) By using Constructor [At the time of creating the Object]

3) By accessing the static member of the class.

4) By using Inheritance.

5) By using Reflection API (java.lang.reflex)

---

1) By using Java command

```
public class Test
{
}
```

javac Test.java (Compile the Test.java)

java Test (Here java command will make a request to load Test.class file into JVM memory)

2) By Using Constructor [Object creation]

```
class Test
{
}
class ELC
{
  public static void main(String [] args)
  {
    new Test(); [Making a request to JVM to load the Test.class file]
  }
 }
```

---

---

3) By accessing the static data member of the class :

```
class Test
{
  static int x = 100;
}
class ELC
{
  public static void main(String [] args)
  {
    System.out.println(Test.x); [Making a request to JVM to load the
                                Test.class file]
  }
```

```
  }
```

Program :

---

```
class Demo
{
        static int x = 10;
        static
        {
                System.out.println("Static Block of Demo class Executed!!! :"+x);
        }
}
public class ClassLoading
{
        public static void main(String[] args)
        {
         System.out.println(Demo.x);
        }
}
```

---

4) By using Inheritance :

```
 class Alpha
 {
 }
 class Beta extends Alpha
 {
 }
```

```
class ELC
{
   public static void main(String [] args)
    {
      new Beta(); [Before Beta.class, first of all Alpha.class file will be loaded and then Beta.class file will be loaded]
    }
 }
```

Program :

---

```
class Alpha
{
        static
        {
                System.out.println("Static Block of super class A!!");
        }
}
class Beta extends Alpha
{
        static
        {
                System.out.println("Static Block of Sub class B!!");
        }
}
```

```
class InheritanceLoading
{
	public static void main(String[] args)
	{
		new Beta();
	}
}
```

Note :- java.lang.Object is the first class to be loaded into JVM memory because It is super class of all the classes we have in java.

---

5) By using Reflection API

---

In java.lang package, there is a predefined class called Class. This class called Class contains one predefined static method forName(String className) which is used to LOAD THE SPECIFIED .CLASS INTO JVM MEMORY DYNAMICALLY.

forName() method is throwing a checked execption i.e ClassNotFoundException.

This predefined static method forName(String className) returns java.lang.Class itself, the method whose rturn type is same class name

known as Factory Methods.

---

2 Files :

---

Foo.Java

---

```
package com.ravi.class_loading;
```

```
public class Foo
{
 static
 {
        System.out.println("Foo class static block");
 }
}
```

ClassLaoding.java

---

```
package com.ravi.class_loading;

public class ClassLaoding {

        public static void main(String[] args) throws Exception
        {
         //Fully Qualified Name is required
         Class.forName("com.ravi.class_loading.Foo");
        }
}
```

---

02-01-2024

---

* What is the difference java.lang.ClassNotFoundException and java.lang.NoClassDefFoundError.

java.lang.ClassNotFoundExcption :

---

In Java whenever we try to load the .class file dynamically at runtime by using Class.forName() or loadClass() method of ClassLoader class and if the required .class file is not availavle at RUNTIME then it will generate an execption i.e java.lang.ClassNotFoundException

Note :- Class.forName(String className) does not have any concern with compile time.

```
class Test
{
	static
	{
		System.out.println("Test class static block");
	}
}
public class ClassNotFoundExceptionDemo
{
	public static void main(String[] args) throws Exception
	{
		Class.forName("Demo");
	}
}
```

Here We will get java.lang.ClassNotFoundException because Demo.class file is not available.

---

java.lang.NoClassDefFoundError :

---

In this approach the class name is available at the time of compilation but after compilation the .class file (Message.class) will be deleted by user manually or It was re-located from one package to another package then at the time of execution the required .class file is not available hence we will get java.lang.NoClassDefFoundError.

Plaese notice, class was present at the time of compilation but at runtime the required .class file is not available.

```
class Message
{
	public void greet()
	{
		System.out.println("Good Afternoon All...");
	}
}

public class NoClassDefFoundErrorDemo
{
	public static void main(String[] args)
	{
		Message m = new Message();
		m.greet();
	}
}
```

Note :- After compilation, delete the Message.class file from the current directory.

---

IQ

--

What is the drawback of 'new' keyword?

OR

What is newInstance() method?

OR

How to create the object for the classes which are available at runtime through file or database?

"new" keyword is suitable for creating the object for the classes which are available at compilation time or at the time of writing the source code (.java file) but on the other hand it is not suitable for

the classes are coming from database or some file at runtime dynamically.

In order to create the object for the classes which are coming from the database or file at runtime we need to use newInstance() method available in java.lang.Class.

---

```
public class ObjectAtRuntime
{
        public static void main(String[] args) throws Exception
        {
          Object obj = Class.forName(args[0]).newInstance();
          System.out.println("Object for :"+obj.getClass());
        }
}

class Player{}
class Customer{}
class Employee{}
```

```
class Student{}
```

javac ObjectAtRuntime.java [Compilation]

java ObjectAtRuntime Player

Here we are passing Player class so it will create the object for

Player class.

---

```
class Student
{
	public void m1()
	{
		System.out.println("m1 method");
	}
}
public class ObjectAtRuntime1
{
	public static void main(String[] args) throws Exception
	{
	  Object obj = Class.forName(args[0]).newInstance();
	  Student st = (Student) obj; //Down casting
	  st.m1();
	}
}
```

newInstance() method of java.lang.Class will create the object so, with that object we can call any non-static method of that particular class.

---

Runtime Data Areas :

---

Once the .class file is loaded successfully in the JVM memory then the content of the class is divided into different memory areas which are as follows :

a) Method Area

b) HEAP Area

c) Stack Area

d) PC Register

e) Native Method Stack

a) Method Area :

---

In this area all class level information is available. Actually the .class file is dumpped here hence we have all kinds of information related to class is available like name of the class, name of the immediate super class, package name, method name , variable name, static variable, all method available in that particular class and so on.

This method area returns type java.lang.Class class , this java.lang.Class class object can hold any .class file

(Class c = AnyClass.class)

There is only one method area per JVM.

---

Methods of java.lang.Class :

---

1) public String getName() :- It is used to provide the class name

without class keyword.

2) public String getPackageName() :- It is used to provide the package

name from where the class is belonging.

3) public Method [] getDeclaredMethods() :- It will provide all the

declared methods available in the specified class. The return type of this method is Method array.

4) public Field [] getDeclaredFields() :- It will provide all the

declared fields(variable static + non static ) available in the specified class. The return type of this method is Field array.

Note :- Method and Field both are predefined classes available java.lang.reflect package. Both contain getName() method to get the name of the method and name of the field.

---

The following program explains how to get the complete information of Demo.class file

2 Files :

---

Demo.java

---

```
package com.ravi.method_area;


public class Demo
{
  int x = 100;
  static int y = 200;
  int z = 300;
```

```
 public Demo()
 {

 }

 public void m1() {}
 public void m2() {}
 public void m3() {}
 public void m4() {}


}
```

ClassDescription.java

---

```
package com.ravi.method_area;

import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class ClassDescription
{
	public static void main(String[] args) throws Exception
	{
		Class cls = Class.forName(args[0]);
		System.out.println("class Name is :"+cls.getName());
		System.out.println("Package Name is :"+cls.getPackageName());
```

```
                Field[] fields = cls.getDeclaredFields();
                System.out.println("Fields Are :");
                for(Field field : fields)
                {
                        System.out.println(field.getName());
                }
                System.out.println("Methods Are :");
                 int methodCount = 0;
                 Method[] methods = cls.getDeclaredMethods();
                 for(Method method : methods)
                 {
                        methodCount++;
                        System.out.println(method.getName());
                 }
                 System.out.println("Total Methods are :"+methodCount);
        }

}
```

---

03-01-2024

---

HEAP Area :

---

In java whenever we create the Object, all object related data like instance variable and instance methods are strored in HEAP Area.

This is the 2nd layer architecture of JVM so from this area we can access the static memeber of the class but vice versa is not possible.

We have only one Heap Area per JVM.

---

Stack Area :

---

In java all the methods are executed as a part of Stack Area. Whenever we call a method then it creates Stack Frame. Each Stack Frame contains 3 parts

a) Local variable.

b) Frame Data.

c) Operand Stack.

We have n number of stack area in one JVM.

JVM creates a separate runtime stack for Each and every thread.

---

Garbage Collector :-

---

In older languages like C++, It is the responsibility of the programmer to allocate the memory as well as to de-allocate the memory otherwise there may be chance of getting OutOfMemoryError.

But in Java a user is not responsible to de-allocate the memory that means memory allocation is the responsibility of user but memory de-allocation is automatically done by Garbage Collector.

Garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects (The object which does not contain any references).

It is an automatic memory management in java. JVM internally contains a thread called Garbage collector which is daemon thread, It is responsible to delete the unused objects or the objects which are not containing any references in the heap memory.

Note :- GC uses an algorithm mark and sweep to make an un-used objects eligible for Garbage Collection.

The garbage Collector thread is visiting our program at regular interval to delete the unused objects but as a programmer we can call garbge collector explicitly to visit our program by using the following code.

System.gc(); //explicitly calling the garbage collector

gc() is a predefined static method of System class.

---

There are 3 ways to make an Object eligible for Garbage Collector:

---

1) Assigning a null literal to reference variable

```
Employee e1 = new Employee();
e1 = null;
```

2) Creating an object inside the method

```
public void createObject()
{
   Employee e2 = new Employee();
}
```

Note :- Once the method execution is over automatically Object is eligible for Garbage Collector

3) Assigning new object to the Existing reference variable

```
    Employee e3 = new Employee();


    e3 = new Employee();
```

---

HEAP and Stack diagram for Employee.java

---

```
public class Employee
{
        int id = 100;


        public static void main(String[] args)
        {
                int val = 200;


                Employee e1 = new Employee();


                e1.id = val;


                update(e1);


                System.out.println(e1.id);


     Employee e2 = new Employee();


                e2.id = 500;


                switchEmployees(e2,e1);     //3000x , 1000x
```

```
                    //GC [2 objects are eligible 4 GC 2000x and 4000x]


                            System.out.println(e1.id);
                    System.out.println(e2.id);
              }


            public static void update(Employee e)
            {
       e.id = 500;
                        e = new Employee();
                        e.id = 400;
            }


            public static void switchEmployees(Employee e1, Employee e2)
            {
                        int temp = e1.id;
                        e1.id = e2.id; //500
                        e2 = new Employee();
                        e2.id = temp;
            }
  }
```

---

HEAP and STACK diagram for Beta.java

---

```
class Alpha
{
            int val;
```

```
        static int sval = 200;
        static Beta b = new Beta();

        public Alpha(int val)
        {
    this.val = val;
        }
}

public class Beta
{
        public static void main(String[] args)
        {
                Alpha am1 = new Alpha(9);
                Alpha am2 = new Alpha(2);

                Alpha []ar = fill(am1, am2);

                ar[0] = am1;
     System.out.println(ar[0].val);
     System.out.println(ar[1].val);
        }

        public static Alpha[] fill(Alpha a1, Alpha a2)
        {
                a1.val = 15;

     Alpha fa[] = new Alpha[]{a2, a1};
```

```
            return fa;
        }
}
```

---

PC Register :

---

It stands for Program counter Register.

In order to hold the current executing instruction of running thread we have separate PC register for each and every thread.

Native Method Stack :

---

Native method means, the java methods which are written by using native languages like C and C++. In order to write native method we need native method library support.

Native method stack will hold the native method information in a separate stack.

---

Execution Engine :

---

Interpreter

---

In java, JVM is an interpreter which executes the program line by line. JVM (Interpreter) is slow in nature because at the time of execution if we make a mistake at line number 9 then it will throw the execption at line number 9 and after solving the execption again it will start the execution from line number 1 so it is slow in execution that is the reason to boost up the execution java software people has provided JIT compiler.

JIT Compiler :

---

It stands for just in time compiler. The main purpose of JIT compiler to boost up the execution so the execution of the program will be completed as soon as possible.

JIT compiler holds the repeated instruction like method signature, variables and make it available to JVM at the time of execution so the overall execution becomes very fast.

---

04-01-2024

---

Exception Handling :

---

What is an execption ?

---

An execption is a runtime error.

An execption is an abnormal situation or un-expected situation in a noraml execution flow.

An exception encounter due to dependency, if one part of the program is dependent to another part then there might be a chance of getting Exception.

AN EXCEPTION ALSO ENCOUNTER DUE TO WRONG INPUT GIVEN BY THE USER.

Exception Hierarchy :

---

This Exception hierarchy is available in the diagram (Exception_Hierarchy.png)

Criteria for Exception in Java :

---

1) java.lang.ArrayStoreException

In the following program we have String Array object and we want to insert an integer value (90) so, it will generate an exception

ArrayStoreException as shown in the program

```
public class Main
{
        public static void main(String[] args)
        {
          Object obj[] = new String[3];
          obj[0] = "Ravi";
          obj[1] = "Raj";
          obj[2] = 90;
          System.out.println(Arrays.toString(obj));

        }

}
```

2) java.lang.ArithmeticException

Whenever we divide a number by 0 (an int value) then we will get

java.lang.ArithmeticException

```
int x = 10 /0;
```

3) java.lang.ArrayIndexOutOfBoundsException

If we want to access the index of array is out of the range then we will get java.lang.ArrayIndexOutOfBoundsException

```
int []arr = {12, 90,67};
System.out.println(arr[3]);
```

4) java.lang.NumberFormatException

If we try to convert String value which contains alphabets (not numbers) into corresponding integer then we will get

java.lang.NumberFormatException

```
String str = "ravi";
int no Integer.parseInt(str);
```

5) java.lang.NullPointerException

Upon null if we want to call any method then we will get

java.lang.NullPointerException

```
String str = null;
System.out.println(str.toUpperCase());
```

6) java.util.InputMismatchException

```
Scanner sc = new Scanner(System.in);
```

```
System.out.print("Enter your Roll");
int roll = sc.nextInt(); //user is providing some character
```

---

Note :- As a developer we are responsibe to handle the Exception. System admin is responsibe to handle the error because we cannot recover from error.

---

Exception is the super class of all the execptions whether it is a predefined exception or user-defined exception in Java.

---

WAP that describes that Exception is the super class of all the exceptions we have in java :

```
package com.ravi.exception_demo;


public class ExceptionSuper
{

	public static void main(String[] args)
	{
		Exception e = new ArithmeticException("Divide by zero");
		System.out.println(e); //Fully Qualified name : error msg

		Exception e1 = new ArrayIndexOutOfBoundsException();
		System.out.println(e1);

		Exception e2 = new NullPointerException();
		System.out.println(e2);
	}
```

}

---

WAP that describes that whenever an exception encounter in the program then program will be terminated in the middle.

```
package com.ravi.exception_demo;

import java.util.Scanner;

public class ExceptionHalt {

	public static void main(String[] args)
	{
		System.out.println("Main method started");

		Scanner sc = new Scanner(System.in);
		System.out.print("Enter the value of x :");
		int x = sc.nextInt();

		System.out.print("Enter the value of y :");
		int y = sc.nextInt();

		int result = x/y;  //if y is 0 then program will Halt here

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

```
                System.out.println("Main method ended");
                sc.close();

        }

}
```

Note :- In the above program if the value of y will be 0 then our program will halt in the middle, it is called abnormal termination so, JVM is having default exception handler, which will terminate the program and provide the appropriate message.

---

In order to work with Exception, Object Oriented Programming has provided the following keywords :

1) try
2) catch
3) finally (try with resources [java 1.7])
4) throw
5) throws

---

05-01-2024

---

try block :

---

Whenever our statement is error suspecting statement OR Risky statement then we should write that statement inside the try block.

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

*try block is responsible to trace our code line by line, if any execption encounter then TRY BLOCK WILL CREATE APPROPRIATE EXECPTION OBJECT, AND THROW THIS EXCEPTION OBJECT to the nearest catch block.

After the execption in the try block, the remaining code of try block will not be executed because control will directly transfer to the catch block.

In between try and catch block we cannot write any kind of statement.

catch block :

---

The main purpose of catch block to handle the exception which is thrown by try block.

catch block will only executed if there is an exception in the try block.

---

```
package com.ravi.basic;

import java.util.Scanner;

public class TryDemo {

	public static void main(String[] args)
	{
		System.out.println("Main method started....");
		Scanner sc = new Scanner(System.in);

		try
		{
```

```
                System.out.print("Enter the value of x :");
                int x = sc.nextInt();

                System.out.print("Enter the value of y :");
                int y = sc.nextInt();

                int result = x /y;

                System.out.println("Result is :"+result);
                System.out.println("End of try");
            }
            catch(Exception e) //e = new AE();
            {
                System.out.println("Inside catch");
                System.err.println(e);
            }
            sc.close();

            System.out.println("Main method ended....");
        }
}
```

The above program in the protection of try-catch so, even we have an exception (y=0) but program will be terminated normally.

---

```
package com.ravi.basic;

public class ThrowException
```

```
{
	public static void main(String[] args)
	{
		System.out.println("Main method started");
		try
		{
	 throw new ArithmeticException("Ravi is dividing by zero");

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

		System.out.println("Main method ended");
	}

}
```

The above program describes how to create and throw an exception object explicitly. try block is doing the same implicitly.

---

Program that describes we should provide user-friendly message to our client.

```
package com.ravi.basic;

import java.util.Scanner;
```

```
public class CustomerDemo
{
	public static void main(String[] args)
	{
		System.out.println("Hello Client, Welcome to my application");
		Scanner sc = new Scanner(System.in);

		 try
		  {
			System.out.print("Please enter the value of x :");
			int x = sc.nextInt();

			System.out.print("Please enter the value of y :");
			int y = sc.nextInt();

			int result = x /y;

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

		  }
		 catch(Exception e)
		 {
			System.err.println("Don't put zero here");
		 }
		 sc.close();
		 System.out.println("Thank you 4 visiting my application");
	}
```

}

---

Throwable class method :

---

Throwable class has provided the following three methods :

1) public String getMessage() :- It will provide only error message.

2) public void printStackTrace() :- It will provide the complete details regarding exception like exception class name, exception message, exception class location, exception method name and exception line number.

3) public String toString() :- It will convert the exception into
String representation.

```
package com.ravi.basic;

public class PrintStackTrace
{
	public static void main(String[] args)
	{
		System.out.println("Main method started...");
		try
		{
			String x = "Ravi";
			int y = Integer.parseInt(x);
			System.out.println(y);
		}
		catch(Exception e)
```

```
		{
			e.printStackTrace(); //For complete Exception details
			System.out.println("--------------------------");
			System.out.println(".........................");
			System.err.println(e.getMessage()); //only for Exception message

		}
		System.out.println("Main method ended...");


	}

}
```

---

06-01-2024

---

Working with Specific Exception :

---

Working with Specific Exception :

---

While working with exception, in the corresponding catch block we can take Exception (super class) which can handle any type of Exception.

On the other hand we can also take specific type of exception (ArithmetiException, NullPointerException and so on) which will handle only one type i.e specific type of exception.

```
package com.ravi.basic;


import java.util.InputMismatchException;
```

```
import java.util.Scanner;

public class SpecificException
{
	public static void main(String[] args)
	{
		System.out.println("Main started");

		Scanner sc = new Scanner(System.in);

		try
		{
			System.out.print("Enter your Roll :");
			int roll = sc.nextInt();
			System.out.println("Your Roll is :"+roll);
		}
		catch(InputMismatchException e)
		{
			System.err.println("Input is not in proper format");
		}
		sc.close();
		System.out.println("Main ended");
	}
}
```

---

```
package com.ravi.basic;

public class SpecificException1
```

```
{
	public static void main(String[] args)
	{
		try
		{
			throw new OutOfMemoryError();
		}
		catch(Exception e)
		{
			System.out.println("Inside catch block");
			System.err.println(e);
		}
	}

}
```

Note :- catch block will not be executed because Exception is the super class of all the Exception but it cannot handle error.

```
	package com.ravi.basic;

public class SpecificException1
{
	public static void main(String[] args)
	{
		try
		{
```

```
                throw new OutOfMemoryError();
            }
            catch(Throwable e)
            {
                System.out.println("Inside catch block");
                System.err.println(e);
            }
        }

}
```

Now catch block will be executed.

---

Working with Infinity and Not a number(NaN) :

---

10/0  -> Infinity (Java.lang.ArithmeticException)

10/0.0  -> Infinity  (POITIVE_INFINITY)

0/0  -> Undefined (Java.lang.ArithmeticException)

0/0.0  -> Undefined  (NaN)

While working with Integral literal in both the cases i.e Infinity (10/0) and Undefined (0/0) we will get java.lang.ArithmeticException because java software people has not provided any final, static variable support to deal with Infinity and Undefined.

On the other hand while working with floating point literal in the both cases i.e Infinity (10/0.0) and Undefined (0/0.0) we have final, static variable support so the program will not be terminated in the middle which are as follows

10/0.0 = POSITIVE_INFINITY

-10/0.0 = NEGATIVE_INFINITY

0/0.0 = NaN

InfinityFloatingPoint.java

---

```
package com.ravi.basic;


public class InfinityFloatingPoint
{
	public static void main(String[] args)
	{
	  System.out.println("Main Started");
	  System.out.println(10/0.0);
	  System.out.println(-10/0.0);
	  System.out.println(0/0.0);
	  System.out.println(10/0);
	  System.out.println("Main Ended");
	}


}
```

---

Working with multiple try catch :

---

According to our application requirement we can provide multiple try-catch in my application to work with multiple execptions.

```
package com.ravi.basic;

public class MultipleTryCatch
{
	public static void main(String[] args)
	{
	 System.out.println("Main method started!!!!");


	 try
	 {
		int arr[] = {10,20,30};
		System.out.println(arr[3]);
	 }
	 catch(ArrayIndexOutOfBoundsException e)
	 {
		System.err.println("Array is out of limit!!!");
	 }


	 try
	 {
		String str = null;
		System.out.println(str.length());
	 }
	 catch(NullPointerException e)
	 {
		System.err.println("ref variable is pointing to null");
	 }


	 System.out.println("Main method ended!!!!");
```

```
	}
}
```

---

* Single try with multiple catch block :

---

According to industry standard we should write try with multiple catch block so we can provide proper information for each and every exception.

While working with multiple catch block always the super class catch block must be last catch block.

From java 1.7 this multiple catch block we can also represent by using | symbol.

---

```
package com.ravi.basic;
public class MultyCatch
{
	public static void main(String[] args)
	{
		System.out.println("Main Started...");
		try
		{
			int c = 10/0;
			System.out.println("c value is :"+c);

			int []x = {12,78,56};
			System.out.println(x[5]);
		}
```

```
		catch(ArrayIndexOutOfBoundsException e1)
		{
			System.err.println("Array is out of limit...");
		}
		catch(ArithmeticException e1)
		{
			System.err.println("Divide By zero problem...");
		}
		catch(Exception e1)
		{
			System.out.println("General");
		}
		System.out.println("Main Ended...");
	}
}
```

While working with multiple catch block, the try block will entertain only the first exception, even we have multiple exceptions in the try block.

---

```
package com.ravi.basic;


public class MultyCatch1
{
	public static void main(String[] args)
	{
		System.out.println("Main method started!!!");
		try
		{
```

```
            String str1 = null;
            System.out.println(str1.toUpperCase());

            String str2 = "Ravi";
            int x = Integer.parseInt(str2);
            System.out.println("Number is :"+x);
        }
        catch(NumberFormatException | NullPointerException  e)
        {
                e.printStackTrace();
        }

        System.out.println("Main method ended!!");
    }

}
```

---

07-01-2024

---

finally block :

---

finally is a block which is meant for Resource handling purposes.

According to Software Engineering, the resources are memory creation, buffer creation, opening of a database, working with files, working with network resourses and so on.

Whenever the control will enter inside the try block always the finally block would be executed.

We should write all the closing statements inside the finally block because irrespective of exception finally block will be executed every time.

If we use the combination of try and finally then only the resources will be handled but not the execption, on the other hand if we use try-catch and finally then execption and resourses both will be handled.

---

//Programthrough which we can handle only resource but not exception

```
package com.ravi.basic;


public class FinallyBlock
{
	public static void main(String[] args)
	{
		System.out.println("Main method started");


		try
		{
			System.out.println(10/0);
		}
		finally
		{
			System.out.println("Finally Block");
		}


		System.out.println("Main method ended");
	}


}
```

Here finally block will be executed, even we don't have catch block and program will be terminated in the middle.

---

```
package com.ravi.basic;


public class FinallyWithCatch
{
	public static void main(String[] args)
	{
		try
		{
		   int []x = new int[-2];//We can't pass negative size of an array in negative
		   x[0] = 12;
		   x[1] = 15;
		   System.out.println(x[0]+" : "+x[1]);
		}
		catch(NegativeArraySizeException e)
		{
			System.err.println("Array Size is in negative value...");
		}
		finally
		{
		  System.out.println("Resources will be handled here!!");
		}
		System.out.println("Main method ended!!!");
	}
}
```

Here we are handling the Exception and resource both.

---

Limitation of finally block :

---

The following are the limitations of finally block :

1) User is responsible to close the resources manually.
2) Due to finally block the length of the code will be increased.
3) While using finally block we should declare all our resources
   outside of the try block otherwise the resourses will become
   block level variable.

```
package com.ravi.basic;

import java.util.InputMismatchException;
import java.util.Scanner;

public class FinallyLimitation
{
	public static void main(String[] args)
	{
		Scanner sc = null;
		try
		{
			sc = new Scanner(System.in);
			System.out.print("Enter your Employee Number :");
			int eid = sc.nextInt();
```

```
                System.out.println(eid);
            }
            catch(InputMismatchException e)
            {
                System.err.println("Input is not in a proper format");
            }
            finally
            {
                sc.close();
                System.out.println("Scanner resource handled");
            }
        }

}
```

---

try with resources :

---

To avoid all the limitation of finally block, Java software people introduced a separate concept i.e try with resources from java 1.7 onwards.

Case 1:

---

```
try(resource1 ; resource2)  //Only the resources will be handled
{
}
```

Case 2 :

---

```
try(resource1 ; resource2) //Resources and Exception both will be
{                    //handled
}
catch(Exception e)
{
}
```

There is a predefined interface available in java.lang package called AutoCloseable which came from java 7 onwards which contains predefined abstract method i.e close() which throws Exception.

There is another predefined interface available in java.io package called java.io.Closeable came from 1.5 (re-modified from java 1.7), this Closeable interface is the sub interface for Auto Closeable interface.

```
public interface java.lang.AutoCloseable
{
  public abstract void close() throws Exception;
}
public interface java.io.Closeable extends java.lang.AutoCloseable
{
public abstract void close() throws IOException;
}
```

Whenever we pass any resourse class as part of try with resources then that class must implements either Closeable or AutoCloseable interface so, try with resourses will automatically call the respective class

close() method even an exception is encountered in the try block.

```
try(ResourceClass rc = new ResourceClass())  //This ResourceClass must
```

{ implements either

Closeable or AutoCloseable interface

} so, try block will

catch(Exception e) automatically call the

{ close() method.

}

The following program explains how try block is invoking the close() method available in DatabaseResource class and FileResource class.

---

3 Files :

---

DatabaseResource.java

---

```
package com.ravi.auto_close;

public class DatabaseResource implements AutoCloseable
{
	@Override
	public void close() throws Exception
	{
		System.out.println("Database resource closed");
	}

}
```

FileResource.java

---

```
package com.ravi.auto_close;

import java.io.Closeable;
import java.io.IOException;

public class FileResource implements Closeable
{
	@Override
	public void close() throws IOException
	{
		System.out.println("File Resource closed");
	}

}
```

Main.java

---

```
package com.ravi.auto_close;

public class Main
{
	public static void main(String[] args) throws Exception
	{
		DatabaseResource dr = new DatabaseResource();
```

```
                FileResource fr = new FileResource();


                try(dr ; fr)
                {
                 System.out.println(10/0);
                }
                catch(ArithmeticException e)
                {
                        System.err.println("Don't provide zero");
                }


        }

}
```

---

```
package com.ravi.try_with_resource_1;

import java.util.InputMismatchException;
import java.util.Scanner;

public class TryWithResource
{
        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                try(sc)
                {
                        System.out.print("Enter your Age :");
```

```
                int age = sc.nextInt();
                System.out.println("Your Age is :"+age);
            }
            catch(InputMismatchException e)
            {
                System.err.println("Input Mismatched Exception");

            }
            System.out.println("Reached Destination");
        }
}
```

---

08-01-2024

---

Nested try block :

---

If we write a try block inside another try block then it is called Nested try block.

```
try  //Outer try
{
 statement1;
   try  //Inner try
   {
     statement2;
   }
   catch(Exception e) //Inner catch
   {
   }
```

```
}
catch(Exception e) //Outer Catch
{
}
```

The execution of inner try block depends upon outer try block that means if we have an exception in the Outer try block then inner try block will not be executed.

---

```
package com.ravi.basic;


public class NestedTryBlock
{
	public static void main(String[] args)
	{
	    try  //outer try
	    {
			String x = "India";
			System.out.println("It's length is :"+x.length());


				try  //inner try
				{
					String y = "NIT";
					int z = Integer.parseInt(y);
					System.out.println("z value is :"+z);

				}
				catch(NumberFormatException e)
				{
```

```
                    System.err.println("Number is not in a proper format");
                }
            }
            catch(NullPointerException e)
            {
                System.err.println("Null pointer Problem");
            }
        }
}
```

---

Writing try-catch inside catch block :

---

We can write try-catch inside catch block but this try-catch block will be exceuted if the catch block will be executed.

```
package com.ravi.basic;

import java.util.InputMismatchException;
import java.util.Scanner;

public class TryWithCatchInsideCatch
{

    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        try(sc)
        {
```

```
                System.out.print("Enter your Roll number :");
                int roll = sc.nextInt();
                System.out.println("Your Roll is :"+roll);

            }
            catch(InputMismatchException e)
            {
                System.err.println("Provide Valid input!!");

                try
                {
                    System.out.println(10/0);
                }
                catch(ArithmeticException e1)
                {
                  System.err.println("Divide by zero problem");
                }

            }
        }

}
```

---

Working with return statement in try-catch :

---

try-catch with return statement

---

If we write try-catch block inside a method and that method is returning some value then we should write return statement in both the places i.e inside the try block as well as inside the catch block.

We can also write return statement inside the finally block only, if the finally block is present. After this finally block we cannot write any kind of statement. (Unrechable)

```
package com.ravi.advanced;

public class ReturnExample
{
   public static void main(String[] args)
   {
     System.out.println(methodReturningValue());
   }



         public static int methodReturningValue()
   {
      try
      {
         System.out.println("Try block");
         return 10/0;
      }
      catch (Exception e)
      {
         System.out.println("catch block");
         return 10/0;
      }
```

```
   }
}
```

---

Initialization of a variable in try and catch :

---

A local variable must be initialized inside try block as well as catch block OR at the time of declaration otherwise we will get compilation error if we want to use the local variable.

```
package com.ravi.basic;


public class VariableInitialization
{
        public static void main(String[] args)
        {
          int x;
    try
    {
          x = 12;
          System.out.println(x);
    }
    catch(Exception e)
    {
          x = 12;  //Variable initialization is compulsory here
          System.out.println(x);
    }

    System.out.println("Main completed!!!");
        }
```

}

---

* Difference between Checked Exception and Unchecked Exception :

---

Checked Exception :

---

In java some exceptions are very common exceptions are called Checked excption here compiler takes very much care and wanted the clarity regarding the exception by saying that by using this code you may face some problem at runtime and you did not report me how would you handle this situation at runtime are called Checked exception, so provide either try-catch or declare the method as throws.

All the checked exceptions are directly sub class of java.lang.Exception

Eg:

---

FileNotFoundException, IOException, InterruptedException,ClassNotFoundException, SQLException and so on

Unchecked Exception :-

---

The exceptions which are rarely occurred in java and for these kinds of exception compiler does not take very much care are called unchecked exception.

Unchecked exceptions are directly entertain by JVM because they are rarely occurred in java.

All the un-checked exceptions are sub class of RuntimeException

Eg:

---

ArithmeticException, ArrayIndexOutOfBoundsException, NullPointerException, NumberFormatException, ClassCastException, ArrayStoreException and so on.

Checked Exception :

---

1) Common Exception
2) Compiler takes care (Will not compile the code)
3) Handling is compulsory (try-catch OR throws)
4) Directly the sub class of java.lang.Exception

Unchecked Exception :

---

1) Rare Exception
2) Comiler will not take any care
3) Handling is not Compulsory
4) Sub class of RuntimeException

---

When to provide try-catch or declare the method as throws :-

---

We should provide try-catch if we want to handle the exception by own as well as if we want to provide user-defined messages to the client but on the other hand we should declare the method as throws when we are not interested to handle the exception and try to send it to the JVM for handling purpose.

Note :- It is always better to use try catch so we can provide appropriate user defined messages to our client.

---

Why compiler takes very much care regarding the checked Exception ?

---

As we know Checked Exceptions are very common exception so in case of checked exception "handling is compulsory" because checked Exception depends upon other resources as shown below.

IOException  (we are depending upon System Keyboard )

FileNotFoundException(We are depending upon the file)

InterruptedException (Thread related problem)

ClassNotFoundException (class related problem)

SQLException (SQL related or database related problem)

---

throw :

---

In case of predefined exception try block is responsible to create the exception object and throw the exception object to the nearest catch block but it works with predefined exception only.

If a user wants to throw an exception based on his own requirement and specification by using userdefined exception then we should write throw keyword to throw the user defined exception object explicitly. (throw new LowBalanceException())

THROWING THE EXCEPTION OBJCET EXPLICITLY.

throws :-

---

In case of checked Exception if a user is not interested to handle the exception and wants  to throw the exception to JVM, wants to skip from the current situation then we should declare the method as throws.

It is mainly used to work with Checked Exception.

---

Types of exception in java :

---

Exception can be divided into two types :

1) Predefined Exception OR Built-in Exception

2) Userdefined Exception OR Custom Exception

Predefined Exception :-

---

The Exceptions which are already defined by Java software people for some specific purposes are called predefined Exception or Built-in exception.

Ex :

---

IOException, ArithmeticException and so on

Userdefined Exception :-

---

The exceptions which are defined by user according to their own use and requirement are called User-defined Exception.

Ex:-

---

InvalidAgeException, GreaterMarksException

---

Steps to create userdefined exception :

---

In order to create user defined exception we should follow the following steps.

1) A userdefined exception class must extends either Exception(Checked Exception) Or RuntimeException(Unchecked Exception) as a super class.

a) If our userdefined class extends RuntimeException that menas we are creating UncheckedException.

b) If our userdefined class extends Exception that menas we are creating checkedException and exception handling is compulsory here.

2) The userdefined class must contain No argument constructor as well as parameterized construtor(in case we want to pass some userdefined error message).

We should take No argument constructor if we don't want to send any error message where as we should take parameterized constructor with super keyword if we want to send the message to the super class.

3) We should use throw keyword to throw the Exception object explicitly.

---

Program on Checked Exception :

---

```
package com.ravi.user_defined_exception;

import java.util.Scanner;

class InvalidAgeException extends Exception
{
	public InvalidAgeException()
	{
	}
```

```
	public InvalidAgeException(String message)
	{
		super(message);
	}
}


public class CustomExceptionDemo1
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		try(sc)
		{
			System.out.print("Enter your Age :");
			int age = sc.nextInt();

			if(age<18)
			{
			  throw new InvalidAgeException("Age is Invalid");
			}
			else
			{
				System.out.println("Go for a movie");
			}
		}
		catch(InvalidAgeException e)
		{
```

```
                    e.printStackTrace();
                }

        }

}
```

---

10-01-2024

---

Assignment :

---

Develop one program on Unchecked Exception :

---

```
package com.ravi.exception;

import java.io.IOException;
import java.util.Scanner;

class GreaterMarksException extends RuntimeException
{
   public GreaterMarksException()
   {

   }

   public GreaterMarksException(String message)
   {
```

```
        super(message);
    }
}

public class CustomException
{
        public static void main(String[] args) throws IOException
        {
         System.out.println("Main method started");
         Scanner sc = new Scanner(System.in);
         try(sc)
         {
                System.out.print("Enter the Marks :");
                int marks = sc.nextInt();

                if(marks > 100)
                {
                        throw new GreaterMarksException("Marks is Invalid");
                }
                else
                {
                        System.out.println("Your Marks is :"+marks);
                }
         }
         catch(GreaterMarksException e)
         {
                System.err.println(e.getMessage());
         }
```

```
        System.out.println("Main method Ended");
    }

}
```

---

Note :- If we throw an execption using throw keyword for user defined class then that class must be sub class of Throwable otherwise compilation error will be generated as shown below.

```
class Ravi
{
}

public class Test
{
    public static void main(String[] args)
    {
      throw new Ravi();  //error Ravi is not the sub class of
    }              Throwable

}
```

---

Some important points to remember :

---

a) If the try block does not throw any checked exception then in the corresponding catch block we can't handle checked exception.It will generate compilation error i.e "exception never thrown from the corresponding try statement"

Example :-

```
try
{
  //try block is not throwing any checked Exception
}
catch(IOException e) //Error
{

}
```

---

```
package com.ravi.method_related_rule;

import java.io.IOException;

public class CatchingCheckedWithoutThrow
{
	public static void main(String[] args)
	{
		try
		{
			//throw new IOException();
		}
		catch(IOException e)  //error
		{
			e.printStackTrace();
		}

	}
```

```
}
```

Note :- The above rule is not applicable for Unchecked Exception

```
	try
		{


		}
		catch(ArithmeticException e)  //Valid
		{
			e.printStackTrace();
		}
```

---

b) If the try block does not throw any exception then in the corresponding catch block we can write Exception, Throwable because both are the super classes for all types of Exception whether it is checked or unchecked.

```
package com.ravi.method_related_rule;


import java.io.IOException;
import java.sql.SQLException;


public class CatchingWithSuperClass
{
	public static void main(String[] args)
	{
		try
```

```
		{
			//throw new IOException();
		}
		catch(Exception e)
		{
	e.printStackTrace();
		}

	}

}
```

---

c) At the time of method overriding if the super class method does not reporting or throwing checked exception then the overridden method of sub class not allowed to throw checked exception. otherwise it will generate compilation error but overridden method can throw Unchecked Exception.

```
package com.ravi.method_related_rule;

import java.io.FileNotFoundException;
import java.io.IOException;

class Super
{
	public void show()
	{
		System.out.println("Super class method not throwing checked Exception");
	}
```

```
}
class Sub extends Super
{
	@Override
	public void show()//throws IOException
	{
		System.out.println("Sub class method should not throw checked Exception");
	}
}

public class MethodOverridingWithChecked {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}

}
```

---

d) If the super class method declare with throws keyword to throw a checked exception, then at the time of method overriding, sub class method may or may not use throws keyword.

If the Overridden method is also using throws keyword to throw checked exception then it must be either same exception class or sub class, it should not be super class as well as we can't add more exceptions in the overridden method.

```
package com.ravi.method_related_rule;
```

```
import java.io.FileNotFoundException;
import java.io.IOException;

class Base
{
	public void show() throws FileNotFoundException
	{
		System.out.println("Super class method ");
	}
}
class Derived extends Base
{
	//throws is applicable but must be equal or sub class
	public void show() throws IOException
	{
		System.out.println("Sub class method ");
	}
}

public class MethodOverridingWithThrows
{
	public static void main(String[] args)
	{
	  System.out.println("Overridden method may or may not throw checked exception but if it is throwing then must be same or sub class");
	}
```

```
}
```

e) Initializer must be complete normally (In static block abnormal termination is not possible)

```
class Check
{
  static{
            throw new Exception(); //error
           }
}
public class Test
{
 public static void main(String [] args)
 {



 }
}
```

---

Exception Propagation :

---

Exception propagation :- [Exception object will shift from callee to caller]

---

Whenever we call a method and if the the callee method contains any kind of exception and if callee method doesn't contain any kind of exception handling mechanism (try-catch) then JVM will propagate the exception to caller method for handling purpose. This is called Exception Propagation.

If the caller method also does not contain any exception handling mechanism then JVM will terminate the method from the stack frame hence the remaining part of the method(m1 method) will not be executed even if we handle the exception in another caller method like main.

If any of the the caller method does not contain any exception handling mechanism then exception will be handled by JVM, JVM has default exception handler which will provide the exception message and terminates the program abnormally.

```
package com.ravi.advanced;


public class ExceptionPropagation
{
	public static void main(String[] args)
	{
	  System.out.println("main started!!!");
	  try
		{
			m1();
		}
		catch(Exception e)
		{
			System.out.println("Handled in main");
		}
	  System.out.println("main Ended!!!");
	}

	public static void m1()
```

```
	{
		System.out.println("m1 started!!!");
		  m2();
	   System.out.println("m1 Ended!!!"); //This line is not Executed
	}

	public static void m2()
	{
		System.out.println(10/0);
	}
}
```

---

Rules :

---

```
package com.ravi.method_related_rule;

class Parent
{
	public void m1() throws InterruptedException
	{
		System.out.println("Parent class m1 method");
	}
}

class Child extends Parent
{
	public void m1()
	{
```

```
        super.m1();    //error
            System.out.println("Child class m1 method");
    }
}


public class CallingSuperClassMethodWithThrows {

    public static void main(String[] args) {
            // TODO Auto-generated method stub

    }

}
```

Note :- In the above program we are calling super class method which is throwing a checked Exception but the caller method does not have any protection so compilation error hence provide either try-catch or declare the method as throws. (Same will not be applicable for Un checked Exception)

The following program describes how to call a method which is throwing Checked Exception

```
package com.ravi.method_related_rule;

import java.io.IOException;

class Parent1
{
```

```
	public void m1() throws IOException
	{
		System.out.println("Parent class m1 method");
	}
}

class Child1 extends Parent1
{
	public void m1() throws IOException
	{
	  super.m1();
		System.out.println("Child class m1 method");
	}
}



public class CallingSuperClassMethodWithThrows1 {

	public static void main(String[] args)
	{
		System.out.println("Main");


	}


}
```

---

```
package com.ravi.method_related_rule;
```

```
import java.util.Scanner;

class Parent2
{
	public void m1() throws InterruptedException
	{
		System.out.println("Parent class m1 method");
	}
}

class Child2 extends Parent2
{

	public void m1()
	{
	  try
	  {
		super.m1();
	  }
	  catch(InterruptedException e)
	  {
		e.printStackTrace();
	  }
	}
}
public class CallingSuperClassMethodWithThrows2 {

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

```
	{
		System.out.println("main");


	}


}
```

So the Conclusion is, if we are calling any method and that method is throwing any checked exception then the caller method must have either try-catch or throws (Protection is reqd).

---

11-01-2024

---

String Handling in java :

---

A string literal in Java is basically a sequence of characters. These characters can be anything like alphabets, numbers or symbols which are enclosed with double quotes. So we can say String is a collection of alpha-numeric character.

How we can create String in Java :-

---

In java String can be created by using 3 ways :-

1) By using String Literal

```
String x = "Ravi";
```

2) By using new keyword

```
String y = new String("Hyderabad");
```

3) By using character array

```
char z[] = {'H','E','L','L','O'};
```

---

Immutability in String (Diagram 11-JAN-24)

---

In java Strings Objects are immutable means unchanged so, whenever we create a String object in java it can't be modifiable.

Strings literals are created in a very special memory of HEAP called String Constant Pool(SCP) and it is not eligible for garbage collection.

String once created can't be modifiable.

---

Facts about String and memory :-

---

In java Whenever we create a new String object by using String literal, first of all JVM will verify whether the String we want to create is pre-existing (already available ) in the String constant pool or not.

If the String is pre-existing (already available) in the String Constant pool then JVM will not create any new String object, the same old existing String object would be refer by new reference variable as shown in the diagram(11-JAN)

Note :- In SCP area we can't have duplicate String Object.

---

* Why String objects are immutable :

---

As we know a String object in the String constant pool can be refer by multiple reference variables, if any of the reference variable will modify the String Object value then it would be very difficult for the another reference variables pointing to same String object to get the original value, what they have defined earlier as shown in the diagram.(11-JAN)

That is the reason Strings are immutable in java.

---

WAP in java to show String objects are not eligible for GC.

---

```
package com.ravi.string_test;


public class StringGC
{
	public static void main(String[] args) throws InterruptedException
	{
		String str1 = "india";
		System.out.println(str1 +" : "+str1.hashCode());

		str1 = null;
		System.gc(); //Calling the GC explicitly

		Thread.sleep(5000);

		String str2 = "india";
		System.out.println(str2 +" : "+str2.hashCode());

	}
```

}

From the above program it is clear that String objects are not eligible for GC.

---

12-01-2024

---

* What is the difference between the following two statements ?

---

String x = "Naresh";  [Creating String by Literal]

String y = new String("Hyderabad"); [Creating String Object by new keyword]

---

String x = "Naresh";

It will create one String object and one reference variable and String object will be created in the String constant pool.

---

String y = new String("Hyderabad");

It will create two String objects one is inside the heap memory(non SCP area) which will be refered by y reference variable and the same String object will be placed in the String constant pool area, if it is not available there.

Hence two String Objects and one reference variable will be created.

---

Program in java that describes whenever we create String object by using new keyword then two String objects are created with same content with same hash code.

```
package com.ravi.method_parameter_local;


public class Test {


	public static void main(String[] args)
	{
    String str1 = new String("Hyderabad");


			String str2 = "Hyderabad";


			System.out.println(str1.hashCode());
			System.out.println(str2.hashCode());


			System.out.println(str1 == str2);  //false
	}
}
```

---

The String object created by using new keyword, how they are automatically placed inside SCP area.

In Java whenever we create the String Object by using new keyword then one object will be created in the non SCP area and same object will be placed inside SCP area, Actually Here JVM internally performs intern process. By using this intern process, JVM is placing the String into SCP Area.

The following program explains how to use intern() method explicitly (By user) to place the String created by new keyword into SCP Area.

```
public class Test {

	public static void main(String[] args)
	{
    String str1 = "India";
    String str2 = new String("India");

    System.out.println(str1==str2);

    str2 = str2.intern();

    System.out.println(str1==str2);

	}
}
```

---

Working with method of String class :

---

```
//Three Ways to create the String Object
public class Test
{
	public static void main(String[] args)
	{
		String s1 = "Hello World";	//Literal
		System.out.println(s1);

		String s2 = new String("Hi"); //Using new Keyword
		System.out.println(s2);
```

```
		char s3[] = {'H','E','L','L','O'}; //Character Array
		System.out.println(s3);
	}
}
-----------------------------------------------------------------
//Immutability

public class Test1
{
	public static void main(String[] args)
	{
      String x = new String("india"); //immutable
		x.toUpperCase();
		System.out.println(x); //will print india in small letter


	}
}
-----------------------------------------------------------------
//Solution of immutability [assigning to reference variable]
class Test2
{
	public static void main(String[] args)
	{
		String x = new String("india");
		String y  = x.toUpperCase();
		System.out.println(x);
		System.out.println(y);
```

```
	}
}
```

---

```
//String is collection of alpha-numeric character
public class Test3
{
	public static void main(String[] args)
	{
		String x="B-61 Hyderabad";
		System.out.println(x);

		String y = "123@$5";
		System.out.println(y);

		String z = "67.90";
		System.out.println(z);

		String p = "A";
		System.out.println(p);
	}
}
```

---

Working with methods of String :-

---

String class has provided number of predefined methods to work with String which are as follows :-

1) public char charAt(int indexPosition) :-

---

It is a predefined method available in the String class. The main purpose of this method to extract or fetch or retrieve a single character from the given String.

We need to pass the index position as a parameter to the method and based on the index position it will extract the character. The return type of this method is char.

---

```
//Program on charAt(int indexPosition)
//Program on charAt(int indexPosition)
public class Test4
{
	public static void main(String[] args)
	{
	   String x = "Hello Hyderabad";

    char ch1 =  x.charAt(6);
                System.out.println(ch1); //H

                ch1 = x.charAt(4);
                System.out.println(ch1); //o

                ch1 = x.charAt(9);
                System.out.println(ch1); //e
	}
}
```

---

public String concat(String str) :-

---

It is a predefined method available in the String class. The main purpose of this method to concat or append two Strings. This can be also done by using concatenation operator '+'.

This method takes String as a parameter and the return type of this method is String.

---

```
//Program on concat(String str)
public class Test5
{
	public static void main(String[] args)
	{
		String s1 = "Data";
		String s2 = "base";
		String s3 = s1.concat(s2);
		System.out.println("String after concatenation :"+s3);

	String s4 = "Tata";
		String s5 = "Nagar";
		String s6 = s4+s5;
		System.out.println("String after concatenation :"+s6);

	String s7 = "Naresh";
		System.out.println(s7.concat(" Technology"));
	}
}
```

---

public boolean equals(Object obj) : [Content Comparison]

---

It is a predefined method available in the String class. The main purpose of this method to verify whether two Strings are equal or not based on the content.

If both the Strings are equal it will return true otherwise it will return false.It is case sensitive method.

It takes Object as a parameter because it is an overridden method.It is overridden from Object class.

---

```
public class EqualsMethod {

	public static void main(String[] args)
	{
		String x = new String("india");
		String y = "india";

		System.out.println(x==y);
		System.out.println(x.equals(y));

	}
}
```

---

19-01-2024

---

equlas(Object obj) method implementation by using Predicate

---

```
package com.ravi.equals_predicate;

import java.util.InputMismatchException;
```

```
import java.util.Scanner;
import java.util.function.Predicate;


public class PredicateEquals
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		try(sc)
		{
		System.out.print("Enter Country Name :");
		String name = sc.next();
		Predicate<String> p1 = str -> str.equals("India");
		System.out.println("Country Name is India ?:"+p1.test(name));
		}
		catch(InputMismatchException e)
		{
			e.printStackTrace();
		}
	}


}
```

---

public boolean equalsIgnoreCase(String str) :-

---

It is a predefined method available in the String class. The main purpose of this method to Compare two Strings based on the content by ignoring the case.

This method takes String as a parameter and return type of this method is boolean. It comapres two Strings by ignoring the case so it is not a case sensitive method.

Hence for this method 'A' and 'a' both are same.

```
//Program on boolean equalsIgnoreCase(String s)
public class Test7
{
	public static void main(String[] args)
	{
		String username = args[0];

		if(username.equalsIgnoreCase("Raviinfotech"))
		{
			System.out.println("Welcome to Raviinfotech channel");
		}
		else
		{
			System.out.println("Sorry! wrong username /Password");
		}
	}
}
```

---

IQ

What is differenec b/w == operator and equals(Object obj) method of String class

equals(object obj) method of String class compares two strings based on the content because it is an overriden method where as == operator compares two Strings based on the reference i.e memory address.

```
//IQ
```

```
public class Test8
{
	public static void main(String[] args)
	{
		String s1="India";
		String s2="India";
		String s3=new String("India");

		System.out.println(s1==s2); //true
		System.out.println(s1==s3); //false

		System.out.println(s1.equals(s2)); //true
		System.out.println(s1.equals(s3));  //true
	}
}
```

Note :- String class has overridden equals(Object obj) method from Object class because Object class equals(Object obj) method meant for memory address comparison but this overridden String class equals(Object obj) meant for content comparison.

Note :- String is a final class in java.

---

public int length() :-

---

It is a predefined method available in the String class. The main purpose of this method to find out the length of the given String. The return type of this method is int.

Note :-

---

Length and Size always start from 1 where as index of the character String always starts from 0.

```
//Program on public int length()
class Test9
{
	public static void main(String[] args)
	{
		String x = "Naresh Tech";
      int len = x.length();
		System.out.println("The length of "+x+" is :"+len);
	}
}
```

---

public String replace(char old, char new) :-

---

It is a predefined overloaded method available in the String class. The main purpose of this method to replace a character or a String with another character or String. The return type of this method is String.

By using this method we can replace a single character or a complete String from the given String.

```
//public String replace(char old, char new)
public class Test10
{
public static void main(String [] args)
	{
```

```
    String x = "oxoxoxox";
                System.out.println("String before replacement :"+x);

                System.out.println("String after replacement :"+x.replace('x','X'));

                String y="Manager";
                System.out.println(y.replace("Man","Dam"));
  }
}
```

---

public int compareTo(String s) :-

---

It is a predefined method available in the String class. The main purpose of this method two compare two String based on character by character, comparison of two Strings chracter by chracter based on the UNICODE values are called Lexicographical comparison or dictionary comparison or alphabetical comparison(String case).

The return type of this method is int.It takes String as a parameter.

If s1 and s2 are two valid Strings

if s1==s2 -> 0

if s1>s2 -> +ve

if s1<s2 -> -ve

//public int compareTo(String s)

```
public class Test11
{
    public static void main(String [] args)
   {
      String s1="Sachin";     //PQRS    S > R
      String s2="Sachin";
      String s3="Ratan";

            System.out.println(s1.compareTo(s2)); //0
            System.out.println(s1.compareTo(s3)); //1   [Sachin   Ratan]
            System.out.println(s3.compareTo(s1)); //-1  [Ratan Sachin]

     String s4 = "Apple";
            String s5 = "apple";
            System.out.println(s4.compareTo(s5));  // -32
            System.out.println(s5.compareTo(s4));  // 32

            String s6 = "Ravi";
            String s7 = "Raj";
            System.out.println(s6.compareTo(s7)); //

   }
}
```

---

20-01-2024

---

public String substring(int startIndex) :-

public String substring(int startIndex, int endIndex) :-

---

It is a predefined method available in the String class. The main purpose of this method to extract the part of the specified string based on the index position.

In this method the startIndex starts from 0 whereas endIndex starts from 1.

Both index will be inclusive for extracting the value.

If end index will be less than start index then we will get an exception i.e StringIndexOutOfBoundsException

substring(5,2);

If start index and end index both are equal, nothing will print.

Nither start index nor end index will accept (-ve) value otherwise StringIndexOutOfBoundsException.

---

```
public class Test12
{
        public static void main(String []  args)
        {
    String x="HYDERABAD";
                System.out.println(x.substring(2,7)); //DERAB

                System.out.println(x.substring(3)); //ERABAD

                System.out.println(x.substring(3,3));
```

```
        //java.lang.StringIndexOutOfBoundsException
                //System.out.println(x.substring(6,3));


                //java.lang.StringIndexOutOfBoundsException
                System.out.println(x.substring(6, -3));
        }
}
```

---

public boolean isEmpty() :-

---

It is a predefined method available in the String class. The main purpose of this method to check whether a String is empty or not. This method returns true if the String is empty that means length is 0, otherwise it will return false.

The return type of this method is boolean.

---

```
//public boolean isEmpty()


//public boolean isEmpty()


public class Test13
{
        public static void main(String args[])
        {
                String str1 = "Java by James Gosling";
                String str2 = "";


                System.out.println(str1.isEmpty());
                System.out.println(str2.isEmpty());
```

```
	}
}
```

---

public String intern() :

---

It is a predefined method available in the String class. The main purpose of this method to return canonical representation for the string object that means String interning ensures that all strings having the same contents use the same memory location.

```
//public String intern()
//Returns the Canonical representation for the String object.
public class Test14
{
	public static void main(String args[])
	{
		String s1 = new String("india");
			String s2 = new String("india");
			System.out.println(s1 == s2);

			s1 = s1.intern();
			s2 = s2.intern();
			System.out.println(s1 == s2);

			String s3 = "Hyd";
			String s4 = new String("Hyd");
			System.out.println(s3 == s4);
			s4 = s4.intern();
			System.out.println(s3 == s5);
```

```
	}
}
```

Note :- Java automatically interns the string literals but we can manually use the intern() method on String object created by new keyword so all the Strings which are having same content will get the same String and return the same memory address(Canonical representation for the String ).

From the above program it is clear that, All that String having same content will represent the same memory address so the hashcode value will be same as shown in the program below.

---

The following explains all the Strings having same content will provide

same hashcode.

```
public class Demo
{
	public static void main(String[] args)
	{
		String s1 = "India";
		String s2 = new String("India");
		String s3 = new String("India");

		System.out.println(s1.hashCode()+" : "+s2.hashCode()+" : "+s3.hashCode());

	}

}
```

```
public class Demo
{
	public static void main(String[] args)
	{
		String s1 = "India";
		String s2 = new String("India");
		String s3 = new String("India");

		System.out.println(s1.hashCode()+" : "+s2.hashCode()+" :
"+s3.hashCode());

	}

}
```

---

```
//IQ
public class Test15
{
	public static void main(String args[])
	{
		String x = "india";
		System.out.println("it's length is :"+x.length); //error

		String [] y = new String[10];
		System.out.println("it's length is :"+y.length());//error
	}
}
```

With array variable we have length property where as with String ref variable we have length() method.

---

//public boolean startsWith(String prefix)

//public boolean endsWith(String suffix)

Both the methods are available in String class.

startsWith() is used to verify that the given String is Starting with prefix String or not, if yes it will return true otherwise it will return false.

endsWith() is used to verify that the given String is ending with suffix String or not, if yes it will return true otherwise it will return false.

Both the methods are case-sensitive.

Both methods take String as a  parameter and return type is boolean.

//public boolean startsWith(String prefix)

//public boolean endsWith(String suffix)

//public boolean startsWith(String prefix)

//public boolean endsWith(String suffix)

```
public class Test16
{
	public static void main(String args[])
	{
		String s="Sachin Tendulkar";
```

```
                System.out.println(s.startsWith("Sa"));
                System.out.println(s.endsWith("r"));
        }
}
```

---

public int indexOf(String str) :-

---

It is a predefined method available in the String class. The main purpose of this method to find out the index position of the specified String in the existing String.

It will serach the index position of the first occurrance of the specified String as a parameter.

It takes String as a parameter and return type of this method is int.

```
//public int indexOf(String x)
public class Test17
{
        public static void main(String args[])
        {
                String  str = "India is my country and It is in Asia";
    int index = str.indexOf("is");
                System.out.println("First Occurrance of is :"+index);
        }
}
```

---

public int lastIndexOf(String x) :-

---

It is a predefined method available in the String class. The main purpose of this method to find out the last index position of the Specified String in the existing String.

It will serach the index position of the last occurrance of the String.

It takes String as a parameter and return type of this method is int.

```
//public int lastIndexOf(String x)
public class Test18
{
        public static void main(String args[])
        {
      String s1 = "it is a nice city";
                int lastIndex = s1.lastIndexOf("it");
                System.out.println("Last occurrance of it, is :"+lastIndex+ "th position");
   }
}
```

---

```
//public String toUpperCase() :- converts to upper case letter
public class Test19
{
        public static void main(String args[])
        {
     String str = "india";
                System.out.println(str.toUpperCase());
  }
}
```

---

```
//public String toLowerCase()  converts to lower case.
public class Test20
{
	public static void main(String args[])
	{
		String str = "INDIA";
		System.out.println(str.toLowerCase());  //india
	}
}
```

---

public String trim() :-

---

It is a predefined method available in the String class. The main purpose of this method to remove the white spaces from the begning (heading) and end (trailing) from the String.

It will not remove any white space in the between the String. The return type of this method is String.

---

```
//program on trim()
public class Test21
{
public static void main(String args[])
	{
		String s1="   Tata    ";
		System.out.println(s1+"Nagar"); //      Tata     Nagar

    s1 = "        Hello      Data       ";
		System.out.println(s1.trim() +"Base"); //Hello    DataBase
```

```
        }
}
```

---

public String []  split (String delimiter)  :

---

It is a predefined method available in the String class. The main purpose of this method to split or break the given String based on specified delimiter(Criteria OR Pattern OR Regular Expression).

The return type of this method is String array because It returns the collection of String tokens or multiple Strings.

```
public class  Demo
{
        public static void main(String ...x)
        {
                String str = "My batch is batch 26";

    String words []=   str.split("c");

    for(String word : words)
    {
                System.out.println(word);
    }
        }
}
```

---

```
import java.util.StringTokenizer;
```

```
public class StringTokenizerDemo {

	public static void main(String[] args)
	{
		String str = "Hyderabad is a nice city";

		StringTokenizer st = new StringTokenizer(str, "e");

		System.out.println("Total number of Tokens :"+st.countTokens());

		while(st.hasMoreTokens())
		{
			System.out.println(st.nextToken());
		}

	}

}
```

There is a predefined class called StringTokenizer available in java.util package, is also used to split the String

```
public int countTokens()
public boolean hasMoreTokens()
public String nextToken()
```

---

22-01-2024

---

public char[] toCharArray() :

---

It is a predefined method available in the String class. The main purpose of this method to convert the given string into a sequence of characters. The returned array length is equal to the length of the string.

This method does not take any parameter and return type is character array.

---

```
//public char[] toCharArray()
public class Test23
{
public static void main(String args[])
	{
		String str = "Java technology";
		char ch [] = str.toCharArray();

		for(char c : ch)
		{
			System.out.print(c+"  ");
		}
		System.out.println();
	}
}
```

---

public byte [] getBytes() :-

---

It is a predefined method available in the String class. The main purpose of this method to encode the string into bytes. It converts the string into a sequence of bytes and returns an array of bytes.

String data we cann't write to the byte oriented Stream files so, it must be converted into byte array.

```
public class Test24
{
	public static void main(String[] args)
	{
	  String str = "ABCDEF";
	  byte []b = str.getBytes();

	  for(byte c : b)
		{
		  System.out.println(c);
		}
	}
}
```

---

StringBuffer :-

---

While working with String class the drawback is memory consumption is very high because it is immutable so whenever we want to perform some operation on the existing String Object, a new String object will be created.

In order to solve the problem of immutability as well as high memory consumption, java software people has introdued a separate class called StringBuffer available in java.lang packge from 1.0 onwards.

StringBuffer is a mutable class so we can modify the existing String hence automatically the memory consumption will be low but we have some performance issue because almost all the methods of StringBuffer class are synchronized so at a time only one thread can access the method of StringBuffer hence it is Thread-safe.

In order to solve this performance issue problem java software people has introduced StringBuilder class from 1.5v onwards.

StringBuilder :-

---

It is a predefined class available in java.lang packge. It is also mutable class. The only difference between StringBuffer and StringBuilder is, almost all the methods of StringBuffer are synchronized where as all the methods of StringBuilder are non-synchronized hence performance wise StringBuilder is more better than StringBuffer.

Both the classes are sharing same API so, method name, return type, parameter list all are same.

* What is the difference String, StringBuffer and StringBuilder?

---

Diagram is available in paint window

---

```
//String, StringBuffer and StringBuilder Objects comparison
public class Test25
{
public static void main(String args[])
        {
```

```
                StringBuilder sb1=new StringBuilder("Data"); //mutable
                sb1.append("Base");
                System.out.println(sb1);

                StringBuffer sb2=new StringBuffer("Data"); //mutable
                sb2.append("Base");
                System.out.println(sb2);

                String sb3 = new String("Data");  //immutable
          sb3.concat("Base");
     System.out.println(sb3);
         }
}
```

---

public int capacity() :

---

StringBuffer class contains capacity method() through which we can find out the initial capacity of StringBuffer class in the form of Characters.

StringBuffer sb = new StringBuffer(); //default capacity is 16

new capacity = (current capacity * 2) + 2

new Capacity = (16 * 2) + 2 = 34

---

```
//public int  capacity()
//new capacity = ( current capacity*2)+2.
public class Test26
```

```
{
public static void main(String args[])
        {
                StringBuffer sb1 = new StringBuffer();
                System.out.println(sb1.capacity()); //16

          StringBuffer sb2 = new StringBuffer("India");  //21 (16+5)
          System.out.println(sb2.capacity());

     sb2.append("is great. It is in Asia"); //44  (21*2)+2 = 44
                System.out.println(sb2.capacity());

        }
}
---------------------------------------------------------------------
//public StringBuffer insert(int position, String str)
//Based on the index position we can insert the String
public class Test27
{
public static void main(String args[])
        {
                StringBuffer sb1=new StringBuffer("Hello");
                sb1.insert(1,"Java");
                System.out.println(sb1);  //HJavaello

                StringBuilder sb2=new StringBuilder("Hello");
                sb2.insert(1,"Java");
                System.out.println(sb2);
```

```
	}
}
```

---

```
//public AbstractStringBuilder reverse()
//Used to reverse the given String
class Test28
{
	public static void main(String[] args)
	{
		StringBuffer sb1=new StringBuffer("Hello");
		sb1.reverse();
		System.out.println(sb1);

		StringBuilder sb2=new StringBuilder("Java");
		sb2.reverse();
		System.out.println(sb2);
		}
}
```

---

23-01-2024

---

Program that describes StringBuilder is more better than StringBuffer in performance.

```
package com.ravi.performance;

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

```
    {
            long startTime =  System.currentTimeMillis();

      for(int i=1; i<=1000000; i++)
      {
       StringBuffer sb = new StringBuffer(" Java ");
       sb.append(" technology ");
      }

      long endTime =  System.currentTimeMillis();

      System.out.println("The total time taken by StringBuffer class to execute this
loop :"+(endTime-startTime)+" ms");

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

      for(int i=1; i<=1000000; i++)
      {
       StringBuilder sb = new StringBuilder(" Java ");
       sb.append(" technology ");
      }

      endTime =  System.currentTimeMillis();

      System.out.println("The total time taken by StringBuilder class to execute this
loop :"+(endTime-startTime)+" ms");

  }
```

}

Note :- System is a predefined class available in java.lang package and it contains a predefined static method currentTimeMillis() , the return type of this method is long, actually it returns the current time of the system in ms.

public static long currentTimeMillis()

---

String class Constructor :

---

String class is containing 15 constructors (some are deprecated) the commonly used constructors are :

1) String s1 = new String();

2) String s2 = new String("India");

3) String s3 = new String(byte []b);

4) String s4 = new String(char []b);

5) String s5 = new String(StringBuffer sb);

6) String s6 = new String(StringBuilder sb);

Some operation on String methods :

---

//String class Constructor

```
//WAP in java to reverse a String
import java.util.Scanner;
public class Test1
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter a String to reverse :");
		String str = sc.nextLine();  //nit

    for(int i=str.length()-1; i>=0; i--)  //i =0
		{
			System.out.print(str.charAt(i)); //tin
		}
    System.out.println();
	}
}
```

---

```
//WAP in java to reverse a String
import java.util.Scanner;
public class Test2
{
  public static void main(String[] args)
	{
    Scanner sc = new Scanner(System.in);
		System.out.println("Enter a String to reverse :");
		String input = sc.nextLine();
```

```
      StringBuilder sb = new StringBuilder();
      sb.append(input);
      System.out.println(sb.reverse());
   }
}
```

---

```
//Program to find out the duplicate characters in String as well as count it in a String
import java.util.*;
public class Test3
{
public static void main(String ...x)
{
 Scanner sc = new Scanner(System.in);
 System.out.println("Enter a String :");
 String str = sc.nextLine(); //ravishankar

 int count = 0;
 char[] arr = str.toCharArray();

 System.out.println("Duplicate Characters are:");

 for (int i = 0; i < str.length(); i++)    //i=2
 {
   for (int j = i + 1; j < str.length(); j++) //j=3
           {
    if (arr[i] == arr[j])   //r == r
                       {
         System.out.println(arr[j]);
```

```
          count++;
          break;
        }
    }
 }
  System.out.println("Total duplicate characters are :"+count);
}
}
```

---

```
//Remove a specified character from the given String
import java.util.*;
public class Test4
{
  public static void main(String[] args)
  {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter a String :");
    String str = sc.nextLine();  //ravi

          System.out.println("Enter a character you want to remove :");
    char removeChar = sc.next().charAt(0); //v

    StringBuilder result = new StringBuilder(); //rai

    for (char c : str.toCharArray()) // {  'i'}
                    {
        if (c != removeChar)    // i != v
                     {
```

```
          result.append(c);
        }
      }
    System.out.println(result);
  }
}
```

---

```
//Program to check whether a String contains vowels or not?
import java.util.*;
public class Test5
{
  public static void main(String[] args)
  {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter a String :");
    String str = sc.nextLine(); //SKY

    boolean containsVowel = false;

    for (char c : str.toLowerCase().toCharArray())
           {
     if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
            {
       containsVowel = true;
       break;
     }
    }
```

```
   if (containsVowel)
           {
     System.out.println("The string contains a vowel.");
   }
           else
           {
     System.out.println("The string does not contain a vowel.");
   }
 }
}
```

---

```
//How to sort a String data
import java.util.Scanner;
public class Test6
{
  public static void main(String[] args)
  {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter a String :");
    String str = sc.nextLine();  //mango

    char[] chars = str.toCharArray();

    for (int i = 0; i < chars.length; i++)
           {
     for (int j = i + 1; j < chars.length; j++)
            {
       if (chars[i] > chars[j])
```

```
                    {
        char temp = chars[i];
        chars[i] = chars[j];
        chars[j] = temp;
      }
     }
   }
   System.out.println(new String(chars));
 }
}
```

---

```
//count the occurrence of a given character in the existing String
import java.util.*;

public class Test7
{
public static void main(String[] args)
{
   Scanner sc = new Scanner(System.in);
   System.out.println("Enter a String :");
   String str = sc.nextLine(); //apple    p

          System.out.println("Enter a character :");
   char target = sc.next().charAt(0); //p

   int count = 0;

   for (int i = 0; i < str.length(); i++) // i = 5     length = 5
```

```
          {
        if (str.charAt(i) == target)   //e == p
                                {
           count++;  //2
        }
   }
     System.out.println("The character '" + target + "' appears " + count + " times in the string '" + str + "'");
   }
}
```

---

Character class in java :

---

It is a predefined Wrapper class available in java.lang package. It contains the following static methods to check whether a chracter is digit or uppercase or lowecase or not?

```
public static boolean isDigit(char ch);  //ravi1ui

public static boolean isUpperCase(char ch);

public static boolean isLowerCase(char ch);
```

---

//Program to find out a String contains digit or not

//public static boolean isDigit(char ch)

```
import java.util.*;
public class Test8
{
public static void main(String[] args)
```

```
{
   Scanner sc = new Scanner(System.in);
   System.out.println("Enter a String :");
   String str = sc.nextLine();

   boolean containsDigits = false;

   for (int i = 0; i < str.length(); i++)   ///Ravi123
          {
    if (Character.isDigit(str.charAt(i)))
           {
      containsDigits = true;
      break;
    }
   }

   if (containsDigits)
          {
    System.out.println("The string contains digits.");
   }
          else
   {
    System.out.println("The string does not contain digits.");
   }
 }
}
```

---

//program to count capital and small letter from the given String

```
//public static boolean isUpperCase(char ch)
//public static boolean isLowerCase(char ch)

import java.util.Scanner;
public class Test9
{
public static void main(String[] args)
{
 Scanner sc = new Scanner(System.in);
 System.out.print("Enter a string: ");
 String str = sc.nextLine();  //RaVi

 int upperCase = 0, lowerCase = 0;

  for (int i = 0; i < str.length(); i++)
         {
   char ch = str.charAt(i);

          if (Character.isUpperCase(ch))
          {
    upperCase++;
   }
          else if (Character.isLowerCase(ch))
          {
    lowerCase++;
   }
  }
  System.out.println("Uppercase letters: " + upperCase);
```

```
   System.out.println("Lowercase letters: " + lowerCase);
 }
}
```

---

```
//Program to count the consonants and vowels in the given String
import java.util.Scanner;
public class Test10
{
public static void main(String[] args)
{
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a string: ");
    String str = sc.nextLine();

   int vowels = 0, consonants = 0;

   for (int i = 0; i < str.length(); i++)
          {
     char c = str.charAt(i);

     if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ||
        c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U')
           {
      vowels++;
     }
           else
           {
      consonants++;
```

```
    }
   }
   System.out.println("Vowels: " + vowels);
   System.out.println("Consonants: " + consonants);
 }
}
```

---

```
//check a String  is palindrome or not
import java.util.Scanner;
public class Test11
{
public static void main(String[] args)
{
  Scanner sc = new Scanner(System.in);
  System.out.print("Enter a string: ");
  String str = sc.nextLine();          // madam

   boolean isPalindrome = true;

   for(int i = 0; i < str.length() / 2;   i++)    //i=2   length = 2
           {
      if (str.charAt(i) != str.charAt(str.length() - i - 1)) //a != a
                      {
        isPalindrome = false;
        break;
      }
   }
```

```
   if (isPalindrome)
           {
     System.out.println(str + " is a palindrome.");
   }
           else
           {
     System.out.println(str + " is not a palindrome.");
   }
 }
}
```

---

24-01-2024

---

Multithreading

---

Uniprocessing :-

---

In uniprocessing, only one process can occupy the memory So the major drawbacks are

1) Memory is westage

2) Resources are westage

3) Cpu is idle

To avoid the above said problem, multitasking is introduced.

In multitasking multiple task can concurrently work with CPU so, our task will be completed as soon as possible.

Multitasking is further divided into two categories.

a) Process based Multitasking

b) Thread based Multitasking

Process based Multitasking :

---

If a CPU is switching from one subtask(Thread) of one process to

another subtask of another process then it is called Process based Multitasking.

Thread based Multitasking :

---

If a CPU is switching from one subtask(Thread) to another subtask within the same process then it is called Thread based Multitasking.

---

Thread :-

---

A thread is light weight process and it is the basic unit of CPU which can run concurrently with another thread within the same context (process).

It is well known for independent execution. The main purpose of multithreading to boost the execution sequence.

A thread can run with another thread at the same time so our task will be completed as soon as possible.

In java whenever we define main method then JVM internally creates a thread called main thread.

---

Program that describes that main is a Thread :

---

Whenever we define main method then JVM will create main thread internally, the purpose of this main thread to execute the entire main method.

In java there is a predefined class called Thread available in java.lang package, this class contains a predefined static method currentThread() which will provide currently executing Thread.

Thread t = Thread.currentThread(); //Factory Method

Thread class has provided predefined method getName() to get the name of the Thread.

MainThread.java

---

```
package com.ravi.thread;

public class MainThread
{
        public static void main(String[] args)
        {
    String name = Thread.currentThread().getName();
    System.out.println("Running thread name is :"+name);
        }
```

}

---

How to create a userdefined Thread in java ?

---

As we know whenever we define the main method then JVM internally creates a thread called main thread.

The purpose of main thread to execute the entire main method so at the time of execution of main method a user can create our own userdefined thread.

In order to create the userdefined Thread we can use one of the following two ways :-

1) By extending java.lang.Thread class

2) By implementing java.lang.Runnable interface

Note :- Thread is a predefined class available in java.lang package where as Runnable is a predefined interface available in java.lang Package.

---

public synchronized void start() :

---

start() is a predefined method of Thread class and this method internally performs two tasks

1) It makes a request to opearting system to assign a new thread to perform concurrent execution.

2) It internally invokes the run() method as a part a separate Stack.

Note :- For every individual thread, JVM creates a separate runtime stack.

The following program explains how to create a userdefined Thread by extending Thread approach.

```
package com.ravi.thread_demo;


class Test extends Thread
{
	@Override
	public void run()
	{
		System.out.println("Child Thread is running");
	}
}

public class MyThread
{
	public static void main(String[] args) throws InterruptedException
	{
		System.out.println("Main thread started...");

		Test t1 = new Test();
		t1.start();

		System.out.println("Main thread Ended...");

	}
```

}

In the above program main thread and Thread-0 threads are created both threads are executing their own Stack Memory as shown in the diagram

(25-JAN-24)

---

public boolean isAlive() :-

---

It is a predefined method of Thread class through which we can find out whether a thread has started or not ?

As we know a new thread is created after calling start() method so if we use isAlive() method before start() method, it will return false but if the same isAlive() method if we invoke after the start() method, it will return true.

We can't restart a thread in java if we try to restart then It will generate an exception i.e java.lang.IllegalThreadStateException

```
package com.ravi.thread_demo;

class Test extends Thread
{
	@Override
	public void run()
	{

		System.out.println("Child Thread is running ");
	}
}
```

```
public class MyThread
{
	public static void main(String[] args) throws InterruptedException
	{

		System.out.println("Main thread started...");

		Test t1 = new Test();

		System.out.println("Child thread started ? "+t1.isAlive());
		t1.start();
		System.out.println("Child thread started ? "+t1.isAlive());

		//t1.start(); [java.lang.IllegalThreadStateException]
		System.out.println("Main thread Ended...");

	}

}
```

---

```
//Exception while executing the thread

package com.ravi.basic;

class Stuff extends Thread
{
	@Override
```

```
	public void run()
	{
		System.out.println("Child Thread is Running!!!!");
	}
}
public class ExceptionDemo
{
	public static void main(String[] args)
	{
		System.out.println("Main Thread Started");

		Stuff s1 = new Stuff();
		Stuff s2 = new Stuff();



		s1.start();
		s2.start();

		System.out.println(10/0);

		System.out.println("Main Thread Ended");
	}

}
```

Note :- Here main thread is interrupted due to AE but still child thread will be executed because child thread is executing with separate Stack

---

26-01-2024

---

Loop program by using Multithreading :

---

```
package com.ravi.basic;


class Sample extends Thread
{
	@Override
	public void run()
	{
		String name = Thread.currentThread().getName();


	  for(int i = 1; i<=10; i++)
	  {
	  System.out.println("i value is :"+i+" by "+name+" thread" );
	  }
	}
}
public class ThreadLoop
{
	public static void main(String[] args)
	{
	  System.out.println("Main thread started.....");


	  Sample s = new Sample();
	  s.start();//child thread is created


	  String name = Thread.currentThread().getName();
```

```
        for(int i = 1; i<=10; i++)
        {
        System.out.println("i value is :"+i+" by "+name+ " thread");
        }



        int x =1;
         do
         {
               System.out.println("Hello");
               x++;
         }
         while(x<=10);
      }
}
```

---

How to set and get the name of the Thread :

---

Whenever we create a userdefined Thread in java then by default JVM assigns the name of thread is Thread-0, Thread-1, Thread-2 and so on.

If a user wants to assign some user defined name of the Thread, then Thread class has provided a predefined method called setName(String name) to set the name of the Thread.

On the other hand we want to get the name of the Thread then Thread class has provided a predefined method called getName().

```
public void setName(String name)  //setter


public String getName()  //getter
```

---

```
//Getting the name of the Thread


package com.ravi.basic;
class Test extends Thread
{
	@Override
	public void run()
	{
		String name = Thread.currentThread().getName();
		System.out.println(name +" thread is running Here!!!!");
	}
}
public class ThreadName
{
	public static void main(String[] args)
	{
		Test t1 = new Test();
		Test t2 = new Test();
		t1.start();
		t2.start();
	System.out.println(Thread.currentThread().getName()+" thread is running.....");
	}
}
```

Note :- In the above program we have not assigned userdefined name to Thread so by default the name of the Thread woud be Thread-0 and Thread-1

---

```
package com.ravi.basic;

class Demo extends Thread

{

	@Override

	public void run()

	{

		System.out.println(Thread.currentThread().getName()+" thread is running.....");

	}

}

public class ThreadName1

{

	public static void main(String[] args)

	{

		 Thread t = Thread.currentThread();

	   t.setName("Parent");//changing the main thread name


	   Demo d1 = new Demo();

	   Demo d2 = new Demo();


	   d1.setName("Child1");

		d2.setName("Child2");


	   d1.start();

	   d2.start();
```

```
            System.out.println(Thread.currentThread().getName()+" thread is running!!!!");
        }
}
```

In the above program we are setting the name of the Thread.

---

Thread.sleep(long milisecond) :

---

If we want to put a thread into temporarly waiting state then we should use sleep() method.

The waiting time of the Thread depends upon the time specified by the user as parameter to sleep() method.

Thread.sleep(1000); //Thread will wait for 1 second

It is a static method of Thread class.

It is throwing a checked Exception i.e InterruptedException because there may be chance that this sleeping thread may be interrupted by some another thread.

```
package com.ravi.basic;

class Sleep extends Thread
{
 @Override
 public void run()
 {
```

```
		for(int i=1; i<=10; i++)
		{
			System.out.println("i value is :"+i);
			try
			{
				Thread.sleep(1000);
			}
			catch (InterruptedException e)
			{
				System.err.println("Catch block :"+e);
			}
		}
	 }

}
public class SleepDemo
{
		public static void main(String[] args)
		{
			System.out.println("Main Thread started...");
	Sleep s = new Sleep();
	s.start();

			System.out.println("Main Thread ended...");
		}
}
```

---

```
package com.ravi.basic;
```

```
class MyTest extends Thread
{
	@Override
	public void run()
	{
		for (int i = 1; i <= 5; i++)
		{
		    //Child 1  and Chiild2
			try
			{
				Thread.sleep(1000);
			}
			catch(Exception e)
			{
			 System.err.println("thread has interrupted");
			}

			System.out.println(i + " by " + Thread.currentThread().getName());
		}
	}
}
public class SleepDemo1
{
	public static void main(String[] args)
	{
		System.out.println(Thread.currentThread().getName() + " thread");
```

```
        MyTest t1 = new MyTest();
        MyTest t2 = new MyTest();

        t1.setName("Child1");
        t2.setName("Child2");

        t1.start();
        t2.start();
    }
}
```

---

Thread life cycle in java :

---

As we know a thread is well known for Independent execution and it contains a life cycle which internally contains 5 states (Phases).

During the life cycle of a thread, It can pass from thses 5 states. At a time a thread can reside to only one state of the given 5 states.

1) NEW State (Born state)

2) RUNNABLE state (Ready to Run state) [Thread Pool]

3) RUNNING state

4) WAITING / BLOCKED state

5) EXIT/Dead state

New State :-

---

Whenever we create a thread instance(Thread Object) a thread comes to new state OR born state. New state does not mean that the Thread has started yet only the object or instance of Thread has been created.

Runnable state :-

---

Whenever we call start() method on thread object, A thread moves to Runnable state i.e Ready to run state. Here Thread schedular is responsible to select/pick a particular Thread from Runnable state and sending that particular thread to Running state for execution.

Running state :-

---

If a thread is in Running state that means the thread is executing its own run() method.

From Running state a thread can move to waiting state either by an order of thread schedular or user has written some method(wait(), join() or sleep()) to put the thread into temporarly waiting state.

From Running state the Thread may also move to Runnable state directly, if user has written Thread.yield() method explicitly.

Waiting state :-

---

A thread is in waiting state means it is waiting for it's time period to complete. Once the time period will be completed then it will re-enter inside the Runnable state to complete its remaining task.

Dead or Exit :

---

Once a thread has successfully completed its run method then the thread will move to dead state. Please remember once a thread is dead we can't restart a thread in java.

IQ :- If we write Thread.sleep(1000) then exactly after 1 sec the Thread will re-start?

Ans :- No, We can't say that the Thread will directly move from waiting state to Running state.

The Thread will definetly wait for 1 sec in the waiting mode and then again it will re-enter into Runnable state which is control by Thread Schedular so we can't say that the Thread will re-start just after 1 sec.

---

27-01-2024

---

join() method of Thread class :

---

The main purpose of join() method to put the one thread into waiting mode until the other thread finish its execution.

Here the currently executing thread stops its execution and the thread goes into the waiting state. The current thread remains in the wait state until the thread on which the join() method is invoked has achieved its dead state.

It also throws checked exception i.e InterruptedException so better to use try catch or declare the method as throws.

It is an instance method so we can call this method with the help of Thread object reference.

---

JoinDemo.java

---

```
package com.ravi.basic;


class Join extends Thread
{
	@Override
	public void run()
	{
		for(int i=1; i<=5; i++)
		{
			System.out.println(i);
			try
			{
				Thread.sleep(500);
			}
			catch(InterruptedException e)
			{
				e.printStackTrace();
			}
		}
	}
}


public class JoinDemo
{
	public static void main(String[] args) throws InterruptedException
	{
```

```
        System.out.println("Main Thread Started!!!!!");

        Join j1 = new Join();
        Join j2 = new Join();
        Join j3 = new Join();

        j1.start();

        j1.join(); //putting the main thread into Waiting mode

        j2.start();
        j3.start();



        System.out.println("Main thread completed!!!!");

            }
}
```

---

JoinDemo1.java

---

```
package com.ravi.basic;

public class JoinDemo1
{
        public static void main(String[] args) throws InterruptedException
        {
```

```
		System.out.println("Main thread started");
		Thread thread = Thread.currentThread();
		String name = thread.getName();

		for(int i=1; i<=10; i++)
		{
			System.out.println(i + " by "+name+ " thread ");

		}
		thread.join();

		System.out.println("Main thread ended");

	}

}
```

Here in the above program the main thread is waiting for main thread only so it is a deadloack state.

---

JoinDemo2.java

---

```
package com.ravi.basic;

class Alpha extends Thread
{
	@Override
	public void run()
```

```
        {
                Thread t = Thread.currentThread();
                String name = t.getName();  //Alpha_Thread

                Beta b1 = new Beta();
                b1.setName("Beta_Thread");
    b1.start();
    try
    {
                        b1.join(); //Alpha thread is in Halt mode
                }
    catch (InterruptedException e)
    {
                        e.printStackTrace();
                }

                for(int i=1; i<=10; i++)
                {
                        System.out.println(i+" by "+name);
                }

        }
}

public class JoinDemo2 {

        public static void main(String[] args)
        {
```

```
		Alpha a1 = new Alpha();
		a1.setName("Alpha_Thread");
		a1.start();
	}
}

class Beta extends Thread
{
	@Override
	public void run()
	{
		Thread t = Thread.currentThread();
		String name = t.getName();
		for(int i=1; i<=20; i++)
		{
			System.out.println(i+" by "+name);
		}
		System.out.println(".....");
	}
}
```

---

Creating Thread by implements Runnable approach :

---

```
package com.ravi.basic;

class Foo2 implements Runnable
{
	@Override
```

```
	public void run()
	{
		String name = Thread.currentThread().getName();
		System.out.println("Child Thread is running :"+name);
	}
}
public class RunnableDemo
{
	public static void main(String[] args)
	{
	  Foo2 f1 = new Foo2();

	  Thread t1 = new Thread(f1);
	  t1.start();

	}

}
```

---

29-01-2024

---

Anonymous approach :

---

1) Anonymous Thread class with Thread Reference :

---

```
package com.ravi.anonymous;
```

```
public class AnonymousThreadWithReference
{
	public static void main(String[] args)
	{
		//Anonymous inner class
		Thread t1 = new Thread()
		{
			@Override
			public void run()
			{
				String name = Thread.currentThread().getName();
				System.out.println("Child Thread :"+name);
			}

		};

		String name = Thread.currentThread().getName();
		System.out.println("Parent Thread :"+name);
		t1.setName("Child1");
		t1.start();



	}

}
```

---

2) Anonymous Thread class without Thread Reference :

---

```
package com.ravi.anonymous;

public class AnonymousThreadWithoutReference
{
	public static void main(String[] args)
	{
		//Anonymous inner class without reference
		new Thread()
		{
			@Override
			public void run()
			{
				Thread thread = Thread.currentThread();
				thread.setName("Child1");
				String name = Thread.currentThread().getName();
				System.out.println("Child Thread :"+name);
			}
		}.start();
	}

}
```

---

3) Anonymous inner class with Runnable interface :

---

```
package com.ravi.anonymous;

public class AnonymousInnerClassWithRunnable
{
```

```
	public static void main(String[] args)
	{
		String name = Thread.currentThread().getName();
		System.out.println("Thread name is :"+name);

		Runnable r1 = new Runnable()
		{
			@Override
			public void run()
			{
				String name = Thread.currentThread().getName();
				System.out.println("Thread name is :"+name);
			}

		};
		Thread t1 = new Thread(r1,"Child1");
		t1.start();

	}

}
```

---

4) Runnable by using Lambda :

---

```
package com.ravi.anonymous;

public class RunnableWithLambda
{
```

```
	public static void main(String[] args)
	{
		Runnable r1 =   ()->
		{
		 String name = Thread.currentThread().getName();
		 for(int i=1; i<=10; i++)
		 {
			System.out.println(i+" by "+ name);
		 }

		};

		Thread t1 = new Thread(r1,"Child1");
		t1.start();
			//Runnable r, String name
		Thread t2 = new Thread(new Ravi(),"Child2");
		t2.start();
	}

}
class Ravi implements Runnable
{
	public void run()
	{
		for(int i=1; i<=10; i++)
		{
			System.out.println("Java");
		}
```

```
        }
}
```

---

*In between extends Thread and implements Runnable, which one is better and why?

In between extends Thread and implements Runnable approach, implements Runnable is more better due to the following reasons

1) When we use extends Thread, all the methods and properties of Thread class is available to sub class so it is heavy weight but this is not the case while implementing Runnable interface.

2) As we know Java does not support multiple inheritance using classes so in the extends Thread approach we can't extend another class but if we use implments Runnable interface still we have chance to extend another class(Only one class) and we can also implement one or more interfaces.

3) implements Runnable is a better approach to create multiple threads on a single sub class object.

4) We can implement Lambda for Runnable interface (Functional interface) only but not for Thread class.

---

Thread class Constructor :

---

We have so many constructor in the Thread class but the following are commonly used constructor in the Thread class.

1) Thread t1 = new Thread();

2) Thread t2 = new Thread(String name);

3) Thread t3 = new Thread(Runnable target);

4) Thread t3 = new Thread(Runnable target, String name);

5) Thread t3 = new Thread(ThreadGroup tg, Runnable target, String name);

---

Problem with multithreading :-

---

Multithreading is very good to complete our task as soon as possible but in some situation, It provides some wrong data or wrong result.

In Data Race or Race condition, all the threads try to access the resource at the same time so the result will be corrupted.

In multithreading if we want to perform read operation and data is not updatable then multithreading is good but if the data is updatable data (modifiable data) then multithreading may produce some wrong result or wrong data as shown in the diagram.(29-JAN-24)

---

Program that provide the problems of Thread while booking the ticket

in Railway Reservation System.

```
package com.ravi.limitation_of_multithreading;

class Customer implements Runnable
{
	private int availableSeat = 1;
```

```
        private int wantedSeat;

        public Customer(int wantedSeat)
        {
                super();
                this.wantedSeat = wantedSeat;
        }

@Override
public void run()
{
 String name = null;

        if(availableSeat >= wantedSeat)
        {
         name = Thread.currentThread().getName();
         System.out.println(wantedSeat +" seat is reserved for :"+name);
         availableSeat = availableSeat - wantedSeat;
        }
        else
        {
                name = Thread.currentThread().getName();
                System.err.println("Sorry !"+name+" ticket is not available");
        }

        }

}
```

```
public class RailwayReservation
{
	public static void main(String[] args)
	{
		Customer c1 = new Customer(1);

		Thread t1 = new Thread(c1,"Scott");
		Thread t2 = new Thread(c1,"John");

		t1.start();
		t2.start();
	}

}
```

---

Thread is also not suitable for performing parallel task as shown in the program below

```
package com.ravi.limitation_of_multithreading;

class MyThread implements Runnable
{
	private String str;

	public MyThread(String str)
	{
		this.str=str;
	}
```

```
	@Override
	public void run()
	{
		for(int i=1; i<=10; i++)
		{
			System.out.println(str+ " : "+i);
			try
			{
				Thread.sleep(100);
			}
			catch (Exception e)
			{
				e.printStackTrace();
			}
		}
	}
}
public class CinemaHall
{
	public static void main(String [] args)
	{
	MyThread obj1 = new MyThread("Cut the Ticket");
	MyThread obj2 = new MyThread("Show the Seat");

		Thread t1 = new Thread(obj1);
		Thread t2 = new Thread(obj2);
```

```
                t1.start();
                t2.start();

        }
}
```

---

Problem with multithreading in banking Application

---

```
package com.ravi.banking_application;

public class Banking
{
  private double balance = 5000;
  private double withdrawAmount;

  public Banking(double withdrawAmount)
  {
        this.withdrawAmount = withdrawAmount;
  }

  public static void main(String[] args)
  {
        Banking b = new Banking(5000);

        Runnable r1 = ()->
        {
              String name = null;
              if(b.balance >= b.withdrawAmount)
```

```
                {
                        name = Thread.currentThread().getName();
                        System.out.println(b.withdrawAmount+" Amount withdrawn by
:"+name);
                        b.balance = b.balance - b.withdrawAmount;
                }
                else
                {
                        name = Thread.currentThread().getName();
                        System.out.println("Sorry "+ name +" Balance is insufficiant");
                }
                System.out.println("Your Account balance is :"+b.balance);
        };

        Thread t1 = new Thread(r1,"Virat");
        Thread t2 = new Thread(r1,"Rohit");

        t1.start(); t2.start();
  }

}
```

---

* Synchronization :

---

In order to solve the problem of multithreading java software people has introduced synchronization concept.

In order to acheive synchronization in java we have a keyword called "synchronized".

It is a technique through which we can control multiple threads but accepting only one thread at all the time.

Synchronization allows only one thread to enter inside the synchronized area for a single object.

Synchronization can be divided into two categories :-

1) Method level synchronization

2) Block level synchronization

Method level synchronization :-

---

In method level synchronization, the entire method gets synchronized so all the thread will wait at method level and only one thread will enter inside the synchronized area as shown in the diagram.(30-JAN-24)

Block level synchronization :-

---

In block level synchronization the entire method does not get synchronized, only the part of the method gets synchronized so all the thread will enter inside the method but only one thread will enter inside the synchronized block as shown in the diagram (30-JAN-24)

Note :- In between method level synchronization and block level synchronization, block level synchronization is more preferable because all the threads can enter inside the method so only the PART OF THE METHOD GETS synchronized so only one thread will enter inside the synchronized block.

---

How synchronization controls multiple threads :

---

Every Object has a lock(monitor) in java environment and this lock can be given to only one Thread at a time.

The thread who acquires the lock from the object will enter inside the synchronized area, it will complete its task without any disturbance because at a time there will be only one thread inside the synchronized area(for single Object). *This is known as Thread-safety in java.

The thread which is inside the synchronized area, after completion of its task while going back will release the lock so the other threads (which are waiting outside for the lock) will get a chance to enter inside the synchronized area by again taking the lock from the object and submitting it to the synchronization mechanism.

This is how synchronization mechanism controls multiple Threads.

Note :- Synchronization logic can be done by senior programmers in the real time industry because due to poor synchronization there may be chance of getting deadlock.

---

31-01-2024

---

Program on Method level synchronization :

---

```
package com.ravi.custom_exception;


class Table
{
	public synchronized void printTable(int num)
	{
		for(int i=1; i<=10; i++)
```

```
                {
                        System.out.println(num+" X "+i+" = "+(num*i));
                        try
                        {
                                Thread.sleep(1000);
                        }
                        catch(InterruptedException e)
                        {
                                e.printStackTrace();
                        }
                }
                System.out.println(".............");
        }
}

public class MethodLevelSynchronization
{
        public static void main(String[] args)
        {
                Table obj = new Table(); //lock is created

                Thread t1 = new Thread()
                {
                  @Override
                  public void run()
                  {
                          obj.printTable(5);
                  }
```

```
                };

                Thread t2 = new Thread()
                {
                  @Override
                  public void run()
                  {
                        obj.printTable(10);
                  }
                };

                t1.start();  t2.start();
        }

}
```

---

Program on Block level synchronization :

---

```
package com.ravi.advanced;

//Block level synchronization

class ThreadName
{
        public void printThreadName()
        {
          //This area is accessible by all the threads
          String name = Thread.currentThread().getName();
```

```
            System.out.println("Thread inside the method is :"+name);

                synchronized(this)  //synchronized Block
                {
                        for(int i=1; i<=9; i++)
                        {
                                System.out.println("i value is :"+i+" by :"+name);
                        }
                        System.out.println(".....");
                }
        }
}
public class BlockSynchronization
{
        public static void main(String[] args)
        {
                ThreadName obj1 = new ThreadName(); //lock is created

                Runnable r1 = () -> obj1.printThreadName();

                Thread t1 = new Thread(r1,"Child1");
                Thread t2 = new Thread(r1,"Child2");
                t1.start(); t2.start();
        }
}
```

---

Problem with Object Level Synchronization :

---

From the given diagram it is clear that there is no interference between t1 and t2 thread because they are passing throgh Object1 where as on the other hand there is no interferenec even in between t3 and t4 threads because they are also passing through Object2 (another object).

But there may be chance that with t1 Thread, t3 or t4 thread can enter inside the synchronized area at the same time, simillarly it is also possible that with t2 thread, t3 or t4 thread can enter inside the synchronized area so the conclusion is synchronization mechanism does not work with multiple Objects.(Diagram 31-JAN-24)

ProblemWithObjectLevelSynchronization

---

```
package com.ravi.advanced;
class PrintTable
{
	public synchronized void printTable(int n)
	{
	  for(int i=1; i<=10; i++)
	  {
		System.out.println(n+" X "+i+" = "+(n*i));
		try
		{
			Thread.sleep(500);
		}
		catch(Exception e)
		{
		}
	  }
	  System.out.println(".....");
	}
```

```
}

public class ProblemWithObjectLevelSynchronization
{
	public static void main(String[] args)
	{
		PrintTable pt1 = new PrintTable(); //lock1
		PrintTable pt2 = new PrintTable(); //lock2

		Thread t1 = new Thread()  //Anonymous inner class concept
			{
			@Override
			public void run()
			{
				pt1.printTable(2);    //lock1
			}
			};

		Thread t2 = new Thread()
			{
			@Override
			public void run()
			{
				pt1.printTable(3);    //lock1
			}
			};

		Thread t3 = new Thread()
```

```
                {
            @Override
            public void run()
            {
                pt2.printTable(6);    //lock2
            }
                };

        Thread t4 = new Thread()
                {
            @Override
            public void run()
            {
                pt2.printTable(9); //lock2
            }
                };
                t1.start();     t2.start();     t3.start();  t4.start();
    }
}
```

So, from the above program it is clear that synchronization will not work with multiple objects.

Now, to avoid this Static Synchronization is came into the picture.

---

Static Synchronization :

---

If We declare a synchronized method as a static method then it is called static synchronization.

Now with static synchronization lock will be available at class level but not Object level.

To call the static synchronized method, object is not required so we can call the static method with the help of class name.

Unlike objects we can't create multiple classes for the same application.

```
package com.ravi.advanced;
class MyTable
{
	public static synchronized void printTable(int n)  //static synchronization
	 {
	   for(int i=1; i<=10; i++)
	   {
		try
		{
			Thread.sleep(100);
		}
		catch(InterruptedException e)
		{
			System.err.println("Thread is Interrupted...");
		}
		System.out.println(n+" X "+i+" = "+(n*i));
	   }
	   System.out.println("------------------------");
	 }
```

```
}
public class StaticSynchronization
{
	public static void main(String[] args)
	{
				Thread t1 = new Thread()
					{
				@Override
				public void run()
				{
					MyTable.printTable(5);
				}
					};

					Thread t2 = new Thread()
					{
				@Override
				public void run()
				{
					 MyTable.printTable(10);
				}
					};


					Runnable r3 = new Runnable()
					{
						@Override
						public void run()
```

```
                {
                        MyTable.printTable(15);

                }
            };
            Thread t3 = new Thread(r3);

            t1.start();
            t2.start();     t3.start();

        }
}
```

---

*Inter Thread Communication (ITC) :

---

It is a mechanism to communicate two synchronized threads within the context to achieve a particular task.

In ITC we put a thread into wait mode by using wait() method and other thread will complete its corresponding task, after completion of the task it will call notify() method so the waiting thread will get a notification to complete its remaining task.

ITC can be implemented by the following method of Object class.

1) public final void wait() throws InterruptedException

2) public native final void notify()

3) public native final void notifyAll()

public  final void wait() throws InterruptedException :-

---

It will put a thread into temporarly waiting state and it will release the lock.

It will wait till the another thread invokes notify() or notifyAll() for this object.

public native final void notify() :-

---

It will wake up the single thread that is waiting on the same object.

public native final void notifyAll() :-

---

It will wake up all the threads which are waiting on the same object.

*Note :- wait(), notify() and notifyAll() methods are defined in Object class but not in Thread class because these methods are related to lock(because we can use these methods from the synchronized area ONLY) and Object has a lock so, all these methods are defined inside Object class.

*What is the difference between sleep() and wait()

---

(Given in the diagram 01-FEM-24)

---

//Program that describes if we don't use ITC then the problem is ...

//Program that describes if we don't use ITC then the problem is ...

class Test implements Runnable

```
{
	int var = 0;
	@Override
	public void run()
	{
		for(int i=1; i<=100; i++)
		{
			var = var + i;    //var = 1 3 6 10 15 21 28
			try
			{
				Thread.sleep(200);
			}
			catch (Exception e)
			{
			}
		}

	}
}
public class ITCProblem
{
	public static void main(String[] args)
	{
		Test t = new Test();
		Thread t1 = new Thread(t);
		t1.start();
		try
		{
```

```
                Thread.sleep(200);
            }
            catch (Exception e)
            {
            }

            System.out.println(t.var);
        }
}
```

---

```
//Communication between main thread and child thread using ITC

class SecondThread extends Thread
{
        int x = 0;

        @Override
            public void run()
            {
                //Child Thread is waiting here for the lock
                synchronized(this)
                {
                    for(int i=1; i<=10; i++)
                    {
                        x =  x + i;
                    }
                    System.out.println("Sending notification");
                    notify();  //will give notification to waiting thread
```

```
                }
        }
}
public class InterThreadComm
{
public static void main(String [] args)
{
        SecondThread b = new SecondThread();
        b.start();

        synchronized(b)  //lock is taken by main thread
                {
                        //suspended
                        try
                        {
                                System.out.println("Waiting for b to complete...");
                                b.wait();  // after releasing the lock, waiting here
                                System.out.println("Main thread wake up");
                        }
                        catch (InterruptedException e)
                        {
                        }
                        System.out.println("Value is: " + b.x);
                }
        }
}
```

---

```
class Customer
```

```
{
    int balance=10000;

        public synchronized void withdraw(int amount)   //amount = 15000
        {
                System.out.println("going to withdraw...");
                if(balance < amount)
                        {
                                System.out.println("Less balance; waiting for deposit...");
                                        try
                                        {
                                                wait();  //waiting and releasing the lock
                                        }
                                        catch(Exception e){}
                        }
                balance  = balance - amount;
                System.out.println("withdraw completed..."+balance+" is remaining
balance");
        }

        public synchronized void deposit(int amount)   //amount = 9000
                {
                        System.out.println("going to deposit...");
                        balance = balance + amount;
                        System.out.println("Balance after deposit is :"+balance);
                        System.out.println("deposit completed... ");
                        notify();
                }
```

```
}
public class InterThreadBalance
{
public static void main(String args[])
        {
    Customer c = new Customer();    //lock is created here

                Thread son = new Thread()  //anonymous class concept
                {
                        @Override
                        public void run()
                        {
                                c.withdraw(15000);
                        }
                };
                son.start();

                Thread father = new Thread()
                {
                        public void run()
                        {
                                c.deposit(9000);
                        }
                };

                father.start();
  }
}
```

---

Thread Priority :

---

It is possible in java to assign priority to a Thread.

Thread class has provided two predefined methods setPriority(int newPriority) and getPriority() to set and get the priority of the thread respectively.

In java we can set the priority of the Thread in number from 1- 10 only where 1 is the minimum priority and 10 is the maximum priority.

Whenever we create a thread in java by default its priority would be 5 that is normal priority.

The userdefined thread created as a part of main thread will acquire the same priority of main Thread.

Thread class has also provided 3 final static variables which are as follows :-

Thread.MIN_PRIORITY :- 01

Thread.NORM_PRIORITY : 05

Thread.MAX_PRIORITY :- 10

Note :- We can't set the priority of the Thread beyond the limit(1-10) so if we set the priority beyond the limit (1 to 10) then it will generate an exception java.lang.IllegalArgumentException.

---

```
package com.ravi.advanced;
```

```
public class MainPriority
{
	public static void main(String[] args)
	{
		Thread t = Thread.currentThread();

		System.out.println("Main thread priority is :"+t.getPriority());

		Thread t1 = new Thread();
		System.out.println("User thread priority is :"+t1.getPriority());
	}

}
```

Any thread which is created as a part of main thread will get the

priority of main thread.

---

```
package com.ravi.advanced;

class ThreadP extends Thread
{
	@Override
	public void run()
	{
	 int priority = Thread.currentThread().getPriority();

	 System.out.println("Child Thread priority is :"+priority);
```

```
	}
}
public class MainPriority1
{
	public static void main(String[] args)
	{
		Thread t = Thread.currentThread();
		t.setPriority(8);

		// t.setPriority(11); Invalid java.lang.IllegalArgumentException

		System.out.println("Main thread priority is :"+t.getPriority());

		ThreadP t1 = new ThreadP();
		t1.start();
	}
}
----------------------------------------------------------------------
package com.ravi.advanced;

class ThreadPrior1 extends Thread
{
	@Override
	public void run()
	{

		int count = 0;
```

```
            for(int i=1; i<=1000000; i++)
            {
             count++;
            }

        System.out.println("Thread name is:"+Thread.currentThread().getName());
        System.out.println("Thread priority is:"+Thread.currentThread().getPriority());
        }

        public static void main(String args[])
        {
            ThreadPrior1 m1 = new ThreadPrior1();
            ThreadPrior1 m2 = new ThreadPrior1();

            m1.setPriority(Thread.MIN_PRIORITY);//1
            m2.setPriority(Thread.MAX_PRIORITY);//10

            m1.setName("Last");
            m2.setName("First");

            m1.start();
            m2.start();
        }
}
```

---

Thread.yield() :

---

It is a static method of Thread class .

It will send a notification to thread schedular to stop the currently executing Thread (In Running state) and provide a chance to Threads which are in Runnable state to enter inside the running state having same priority or highest priority. Here The running Thread will directly move from Running state to Runnable state.

The Thread schedular can ignore this notification message given by currently executing Thread.

Here there is no guarantee that  after using yield() method the running Thread will move to Runnable state and from Runnable state the thread can move to Running state.

If the thread which is in runnable state is having low priority then the same running thread will continue its execution.

---

```
class Test implements Runnable
{
	@Override
	public void run()
	{
		for(int i=1; i<=10; i++)
		{
			String name = Thread.currentThread().getName();

			System.out.println("i value is :"+i+" by thread :"+name);

			 if(name.equals("Child1"))
			{
				Thread.yield();  //Give a chance to Child2 Thread
```

```
                        }


                }
  }
}
public class ThreadYieldMethod
{
        public static void main(String[] args)
        {
                Test obj = new Test();


                Thread t1 = new Thread(obj, "Child1");
                Thread t2 = new Thread(obj, "Child2");


                t1.start();  t2.start();
        }
}
```

Note :- In real time if a thread is acquiring more time of CPU then to release that Thread we call yield() method the currently executing Thread.

---

interrupt() method of Thread class :

---

It is a predefined method of Thread class. The main purpose of this method to disturb the execution of the Thread, if the thread is in waiting or sleeping state.

Whenever a thread is interupted then it throws InterruptedException so the thread (if it is in sleeping or waiting mode) will get a chance to come out from a particular logic.

Points :-

---

If we call interrupt method and if the thread is not in sleeping or waiting state then it will behave normally.

If we call interrupt method and if the thread is in sleeping or waiting state then we can stop the thread gracefully.

*Overall interrupt method is mainly used to interrupt the

thread safely so we can manage the resources easily.

Methods :

---

1) public void interrupt () :- Used to interrupt the Thread but the thread must be in sleeping or waiting mode.

2) public boolean isInterrupted() :- Used to verify whether thread is interrupted or not.

---

```
class Interrupt extends Thread
{
  @Override
  public void run()
        {
          Thread t = Thread.currentThread();
          System.out.println(t.isInterrupted());

          for(int i=1; i<=10; i++)
              {
                System.out.println(i);
```

```
                try
                {
                        Thread.sleep(1000);
                }
                catch (Exception e)
                {
                        System.err.println("Thread is Interrupted ");
                        e.printStackTrace();
                }
              }
        }
}
public class  InterruptThread
{
        public static void main(String[] args)
        {
                Interrupt it = new Interrupt();
                System.out.println(it.getState());  //NEW STATE
                it.start();
                it.interrupt();  //main thread is interrupting the child thread
        }
}
---------------------------------------------------------------------
class Interrupt extends Thread
{
  public void run()
        {
          try
```

```
            {
             Thread.currentThread().interrupt();

            for(int i=1; i<=10; i++)
                    {
                      System.out.println("i value is :"+i);
                      Thread.sleep(1000);
                    }

            }
                    catch (InterruptedException e)
                    {
                            System.err.println("Thread is Interrupted :"+e);
                    }
                    System.out.println("Child thread completed...");
            }
}
public class  InterruptThread1
{
            public static void main(String[] args)
            {
                    System.out.println("Main thread is started");
                    Interrupt it = new Interrupt();
                    it.start();
                    System.out.println("Main thread is ended");
            }
}
---------------------------------------------------------------------
```

```
public class InterruptThread2
{
   public static void main(String[] args)
         {
      Thread thread = new Thread(new MyRunnable());
      thread.start();

      try
                  {
        Thread.sleep(5000);  //main thread is waiting for 2 sec
      }
                  catch (InterruptedException e)
                  {
        e.printStackTrace();
      }


      thread.interrupt();
   }
}

class MyRunnable implements Runnable
{
   @Override
   public void run()
         {
      try
                  {
```

```
        while (!Thread.currentThread().isInterrupted())
                                {
          System.out.println("Thread is running...");
          Thread.sleep(500);
        }
      }
                    catch (InterruptedException e)
                    {
        System.out.println("Thread interrupted gracefully.");
      }
                    finally
                    {
        System.out.println("Thread resource can be release here.");
      }
    }
}
```

Note :- Here main thread is in sleeping mode for 5 sec, after wake up main thread is interrupting child thread so child thread will come out from infinite loop and if any resource is attached with child thread that will be released because child thread execution completed.

finally block is there to close the resource.

---

Thread Group :-

---

There is a predefined class called ThreadGroup available in java.lang package.

In Java it is possible to group multiple threads in a single object so, we can perform a particular operation on a group of threads by a single method call.

The Thread class has the following constructor for ThreadGroup

new Thread(ThreadGroup groupName, Runnable target, String name);

---

```
public class ThreadGroupDemo1
{
  public static void main(String[] args)
        {
    ThreadGroup myThreadGroup = new ThreadGroup("NIT_Thread");

    // Create and start threads within the ThreadGroup

    Thread thread1 = new Thread(myThreadGroup, new MyRunnable(), "Thread 1");

    Thread thread2 = new Thread(myThreadGroup, new MyRunnable(), "Thread 2");

    Thread thread3 = new Thread(myThreadGroup, new MyRunnable(), "Thread 3");




    thread1.start();
    thread2.start();
                thread3.start();
```

```
    // Display information about the ThreadGroup and its threads
    System.out.println("ThreadGroup Name: " + myThreadGroup.getName());

    System.out.println("Active Count: " + myThreadGroup.activeCount());
}



static  class MyRunnable implements Runnable //static nested inner
        {
    @Override
    public void run()
                {
      for (int i = 0; i < 3; i++)
                        {
        System.out.println(Thread.currentThread().getName() + ": " + i);
        try
                                {
          Thread.sleep(1000);
        }
                                catch (InterruptedException e)
                                {
          e.printStackTrace();
        }
      }
    }
}
```

}

---

03-02-2024

---

Deadlock :

---

It is a situation where two or more than two threads are in blocked state forever, here threads are waiting to acquire another thread resource without releasing it's own resource.

This situation happens when multiple threads demands same resource without releasing its own attached resource so as a result we get Deadlock situation and our execution of the program will go to an infinite state as shown in the diagram. (03-FEB-24)

```
public class DeadlockExample
        {
 public static void main(String[] args)
         {
   String resource1 = "Ameerpet";
   String resource2 = "Hyderabad";

  // t1 tries to lock resource1 then resource2

  Thread t1 = new Thread()
                  {
          @Override
    public void run()
                   {
```

```
                synchronized (resource1)
                        {
                System.out.println("Thread 1: locked resource 1");
                try
                        {
                        Thread.sleep(1000);
                        }
                        catch (Exception e)
                        {}

                synchronized (resource2) //Nested synchronized block
                {
                        System.out.println("Thread 1: locked resource 2");
                }
        }
 }
};



// t2 tries to lock resource2 then resource1
Thread t2 = new Thread()
        {
 @Override
 public void run()
         {
  synchronized (resource2)
                        {
    System.out.println("Thread 2: locked resource 2");
```

```
        try
                        {
                        Thread.sleep(1000);
                        }
                        catch (Exception e)
                       {}

      synchronized (resource1) //Nested synchronized block
                {
       System.out.println("Thread 2: locked resource 1");
      }
     }
    }
  };
  t1.start();
  t2.start();
 }
}
```

---

```
class Demo
{
        public static void main(String[] args)  throws InterruptedException
        {
                Thread t = Thread.currentThread();
        for(int i =1; i<=10; i++)
                {
                 System.out.println(i);
                 t.join();
```

```
                }
        }
}
```

In the above program main thread is blocking it self.

---

Daemon Thread [Service Level Thread]:

---

Daemon thread is a low- priority thread which is used to provide background maintenance.

The main purpose of of Daemon thread to provide services to the user thread.

JVM can't terminate the program till any of the non-daemon (user) thread is active, once all the user thread will be completed then JVM will terminate all Daemon threads, which are running in the background to support user threads.

The example of Daemon thread is Garbage Collection thread, which is running in the background for memory management.

In order to make a thread as a  Daemon thread , we should use setDaemon(true)

---

```
public class DaemonThreadDemo1
{
  public static void main(String[] args)
        {
           System.out.println("Main Thread Started...");
     Thread daemonThread = new Thread(() ->
                {
         while (true)
```

```
                {
    System.out.println("Daemon Thread is running...");
    try
                {
      Thread.sleep(1000);
    }
                catch (InterruptedException e)
                {
      e.printStackTrace();
    }
  }
});

daemonThread.setDaemon(true);
daemonThread.start();


Thread userThread = new Thread(() ->
            {
  for (int i = 1; i <= 19; i++)
                {
    System.out.println("User Thread: " + i);
    try
                {
      Thread.sleep(2000);
    }
                catch (InterruptedException e)
                {
```

```
                e.printStackTrace();
            }
        }
    });

    userThread.start();

    System.out.println("Main Thread Ended...");
  }
}
```

---

I/O : 4 days

Inner classes : 2 days

Wrapper classes : 1 day

Collection Framework : 24 days

- Generics
- Optional class
- New Date and Time (Java 8)
- Sealed class
- Record class
- Stream API
  - Intermediate Opeartion
    - Terminal Opeartion

---

Remaining method of Object class :

---

Object cloning in java :

---

Object cloning is the process of creating an exact copy of an existing object in the memory.

Object cloning can be done by the following process :

1) Creating Shallow copy

2) Creating Deep copy

3) Using clone() method of java.lang.Object class

4) Passing Object reference to the Constructor.

Shallow Copy :

---

In shallow copy, we create a new reference variable which will point to same old existing object so if we make any changes through any of the reference variable then original object content will be modified.

Here we have one object and multiple reference variables.

Hashcode of the object will be same.

ShallowCopy.java

---

```
package com.ravi.clone_method;

class Student
{
```

```
	int id;
	String name;

	@Override
	public String toString()
	{
		return "Id is :" + id + "\nName is :" + name ;
	}

}
public class ShallowCopy
{
	public static void main(String[] args)
	{
		Student s1 = new Student();
		s1.id = 111;
		s1.name = "Ravi";

		System.out.println(s1);

		System.out.println("After Shallow Copy");

		Student s2 = s1; //shallow copy
		s2.id = 222;
		s2.name = "Shankar";

		System.out.println(s1);
		System.out.println(s2);
```

```
		System.out.println(s1.hashCode());
		System.out.println(s2.hashCode());
	}

}
```

Note :- hashcode value will be same for the objects

---

Deep Copy :

---

In deep copy, We create a copy of object in a different memory location. This is called a Deep copy.

Here objects are created in two different memory locations so if we modify the content of one object it will not reflect another object.

```
package com.ravi.clone_method;

class Employee
{
	int id;
	String name;

	@Override
	public String toString()
	{
		return "Employee [id=" + id + ", name=" + name + "]";
	}
```

```
}

public class DeepCopy
{
	public static void main(String[] args)
	{
		Employee e1 = new Employee();
		e1.id = 111;
		e1.name = "Ravi";

		Employee e2 = new Employee();
		e2.id = e1.id;
		e2.name = e1.name;

		System.out.println(e1 +" : "+e2);

		e2.id = 222;
		e2.name = "shankar";
		System.out.println(e1 +" : "+e2);

		System.out.println(e1.hashCode() +" : "+e2.hashCode());
	}

}
```

Note :- hash code of both the obejcts will be different.

---

protected native Object clone() throws CloneNotSupportedException

---

Object cloning in Java is the process of creating an exact copy of the original object. In other words, it is a way of creating a new object by copying all the data and attributes from the original object.

The clone method of Object class creates an exact copy of an object.

In order to use clone() method , a class must implements Clonable interface because we can perform cloning operation on Cloneable objects only [JVM must have additional information].

We can say an objeect is a Cloneable object if the corresponding class implements Cloneable interface.

It throws a checked Exception i.e CloneNotSupportedException

Note :- clone() method is not the part of Clonable interface[marker interface], actually it is the method of Object class.

clone() method of Object class follow deep copy concept so hashcode will be different.

```
package com.ravi.clone_method;

class Customer implements Cloneable
{
	int id;
	String name;

	@Override
	protected Object clone() throws CloneNotSupportedException
	{
```

```
        return super.clone();
    }


    @Override
    public String toString()
    {
        return "Customer [id=" + id + ", name=" + name + "]";
    }
}

public class CloneMethod
{
    public static void main(String[] args) throws CloneNotSupportedException
  {
        Customer c1 = new Customer();
        c1.id = 222;
        c1.name = "Rahul";

        Customer c2  = (Customer) c1.clone();
        c2.id = 333;
        c2.name = "Rohit";

        System.out.println(c1);
        System.out.println(c2);

        System.out.println(c1.hashCode());
        System.out.println(c2.hashCode());
```

```
        }

}
```

---

protected void finalize() throws Throwable :

---

It is a predefined method of Object class.

Garbage Collector automatically call this method just before an object is eligible for garbage collection to perform clean-up activity.

Here clean-up activity means closing the resources associated with that object like file connection, database connection, network connection and so on we can say resource de-allocation.

Note :- JVM calls finalize method only one per object.

```
package com.ravi.finalize_method;

public class Student
{
   int id;
   String name;

   public Student(int id, String name)
```

```
{
        this.id= id;
        this.name = name;
}

        @Override
        public String toString()
        {
                return "Id is :"+id+"\nName is :"+name;
        }

        @Override
        protected void finalize()
        {
 System.out.println("JVM call this finalize method...");
        }


        public static void main(String[] args) throws InterruptedException
        {
 Student s1 = new Student(111,"Ravi");
 System.out.println(s1.hashCode());
 System.out.println(s1);

 s1 = null;
 System.gc(); //Explicitly calling Garbage Collector
 Thread.sleep(3000);
 System.out.println(s1);
```

```
        }
}
```

---

*What is the difference between final, finally and finalize

final :- It is a keyword which is used to provide some kind of restriction like class is final, Method is

final,variable is final.

finally :- if we open any resource as a part of try block then

that particular resource must be closed inside

finally block otherwise program will be terminated ab-normally and the corresponding resource will not be closed (because the remaining lines of try block will not be executed)

finalize() :- It is a method which JVM is calling automatically just before object destruction so if any resource

(database, file and network) is associated with

that particular object then it will be closed

or de-allocated by JVM by calling finalize().

---

05-02-2024

---

Wrapper classes in java :

---

In java we have 8 primitive data types i.e byte, short, int, long, float, double, char and boolean.

Except these primitives, everything in java is an Object.

If we remove these 8 data types from java then Java will become pure Object Oriented language.

Wrapper class is a technique through which we can convert the primitives to corresponding object. Now it can be divided into two types from 1.5 onwards

a) Autoboxing

b) Unboxing

Autoboxing

---

When we convert the primitive data types into corresponding wrapper object then it is called Autoboxing as shown below.

| Primitive type | | Wrapper Object |
|---|---|---|
| byte | - | Byte |
| short | - | Short |
| int | - | Integer |
| long | - | Long |
| float | - | Float |
| double | - | Double |

char - Chracter

boolean - Boolean

Note :- Wrapper classes has provided valueOf() method through which we can convert the primitive into Object.

---

```
//Integer.valueOf(int);
public class AutoBoxing1
{
	public static void main(String[] args)
	{
		int a = 12;
		Integer x = Integer.valueOf(a); //Upto 1.4 version
		System.out.println(x);


    int y = 15;
		Integer i = y;   //From 1.5 onwards compiler takes care
		System.out.println(i);
	}
}
```

Note :- Integer class has provided predefined static method called

valueOf(int x), the return type of this method is Integer class.

---

```
public class AutoBoxing2
```

```
	{
		public static void main(String args[])
			{
			  int y = 12;
				Integer x = y;
				System.out.println(x);


				double e = 45.90;
				Double d = e;
				System.out.println(d);


		  boolean a = true;
				Boolean b = a;
				System.out.println(b);
			}
}
```

---

WAP to convert a String into integer using base 2

---

```
public class Test
{
	public static void main(String[] args)
	{
		String str = "111";
		int x = Integer.parseInt(str , 2);


		System.out.println(x);
```

```
	}
}
```

---

Integer class has provided the following overloaded methods which are as follows :

public Integer valueOf(int x) :- Primitive to Integer Wrapper

public Integer valueOf(String x) :- String to Integer Wrapper

public Integer valueOf(String x, int radix/base) :- String to Integer object by using specified base/radix.

```
//Integer.valueOf(String str)
//Integer.valueOf(String str, int radix/base)
public class AutoBoxing3
{
	public static void main(String[] args)
	{
			Integer a = Integer.valueOf(15);

			Integer b = Integer.valueOf("25");

	Integer c = Integer.valueOf("111",2); //Here Base we can take upto 36

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

```
	}
}
```

Note :- In the above progarm we can pass base OR radix upto 36

i.e A to Z (26) + 0 to 9 (10) -> [26 + 10 = 36], It can be

calculated by using Character.MAX_RADIX.

Output will be generated on the basis of radix

---

```
public class AutoBoxing4
{
	public static void main(String[] args)
	{
		Integer i1 = new Integer(100);
		Integer i2 = new Integer(100);
		System.out.println(i1==i2);

		Integer a1 = Integer.valueOf(15);
		Integer a2 = Integer.valueOf(15);
		System.out.println(a1==a2);
	}
}
```

We will get false and true, new keyword always create different memory location but valueOf() method will return Integer object and according to above program a1 and a2 both will point to same memory location.

---

```
//Converting integer value to String
public class AutoBoxing5
{
```

```
	public static void main(String[] args)
	{
		int x = 12;
		String str = Integer.toString(x);
		System.out.println(str+2);
	}
}
```

---

```
package com.ravi.stream_intermediate;

public class Autoboxing6 {

	public static void main(String[] args)
	{
		char ch = 'A';
		Character chObj = Character.valueOf(ch);
		System.out.println(chObj);

		boolean b = true;
	   Boolean boolObj = Boolean.valueOf(b);
	   System.out.println(boolObj);

	}

}
```

---

Unboxing :

---

Converting wrapper object to corresponding primitive type is called Unboxing.

| Wrapper Object | | Primitive type |
|---|---|---|
| Byte | - | byte |
| Short | - | short |
| Integer | - | int |
| Long | - | long |
| Float | - | float |
| Double | - | double |
| Chracter | - | char |
| Boolean | - | boolean |

---

```
//Converting Wrapper object into primitive
public class AutoUnboxing1
{
  public static void main(String args[])
            {
        Integer obj = 15;   //Upto 1.4
                        int x = obj.intValue();
```

```
                System.out.println(x);
        }
}
```

---

```
public class AutoUnboxing2
{
    public static void main(String[] args)
    {
                Integer x = 25;
                int y = x;      //JDK 1.5 onwards
                System.out.println(y);
    }
}
```

---

Byte, Short, Integer, Long, Float and Double all these 6 wrapper classes are the sub class of java.lang.Number class. All these 6 classes are providing the following 6 common methods

1) public byte byteValue()

2) public short shortValue()

3) public int intValue()

4) public long longValue()

5) public float floatValue()

6) public dopuble doubleValue()

```
---------------------------------------------------------------------
public class AutoUnboxing3
{
	public static void main(String[] args)
	{
			Integer i = 15;
			System.out.println(i.byteValue());
			System.out.println(i.shortValue());
			System.out.println(i.intValue());
			System.out.println(i.longValue());
			System.out.println(i.floatValue());
			System.out.println(i.doubleValue());
	}
}
---------------------------------------------------------------------
public class AutoUnboxing4
{
	public static void main(String[] args)
	{
		Character c1 = 'A';
		char ch = c1.charValue();
		System.out.println(ch);
	}
}
---------------------------------------------------------------------
public class AutoUnboxing5
{
	public static void main(String[] args)
```

```
	{
		Boolean b1 = true;
		boolean b = b1.booleanValue();
		System.out.println(b);
	}
}
```

---

While working with Primitive type we can easily convert from one type to another type by using Widening.

```
long l = 12;

double d = 12;
```

```
package com.ravi.stream_intermediate;

public class Unboxing
{
 public static void main(String[] args)
 {
	Byte b = 12;

	Short s = 90;

	Integer i = 100;

	Long l = 12L;
```

```
        Double d = 15.0;

        Float f = 12.7F;



 }
}
------------------------------------------------------------
class BufferTest
{
        public static void main(String[] args)
        {
                Integer i1 = 127;
                Integer i2 = 127;
                System.out.println(i1==i2); //true

                Integer i3 = 128;
                Integer i4 = 128;
                System.out.println(i3==i4); //false

                Integer i5 = 128;
                Integer i6 = 128;
                System.out.println(i5.equals(i6)); //true
        }
}
```

equals() method is overridden from Object class to Integer class.

To compare two String Object equals(Object obj) method (Overridden method) is preferable.

---

06-02-2024

---

Collection Framework in java (40 - 45% IQ):

---

Collections framework is nothing but handling individual Objects(Collection Interface) and Group of objects(Map interface).

We know only object can move from one network to another network.

A collections framework is a class library to handle group of Objects.

It is implemented by using java.util package.

It provides an architecture to store and manipulate group of objects.

All the operations that we can perform on data such as searching, sorting, insertion and deletion can be done by using collections framework because It is the data structure of Java.

The simple meaning of collections is single unit of Objects.

---

It provides the following sub interfaces :

1) List (Accept duplicate elements)

2) Set (Not accepting duplicate elements)

3) Queue (Storing and Fetching the elements based on some order i.e FIFO)

Collection is an interface in java.util package where as Collections is predefined class which contains various static method available in java.util package.

---

Methods of Collection inetrafce :

---

Methods of Collection interface :

---

a) public boolean add(Object element) :- It is used to add an item/element in the collection.

b) public boolean addAll(Collection c) :- It is used to insert the specified collection elements in the existing collection(For merging the Collection)

c) public boolean retainAll(Collection c) :- It is used to retain all the elements from existing element. (Common Element)

d) public boolean removeAll(Collection c) :- It is used to delete all the elements from the existing collection.

e) public boolean remove(Object element) :- It is used to delete an element from the collection.

f) public int size() :- It is used to find out the size of the Collection.

g) public void clear() :- It is used to clear all the elements at once from the Collection.

---

List interface :

---

1) It is sub interface of Collection, introduced from 1.2V

2) It stores the element based on the index.

3) It can accept duplicate elements.

4) We can perform sorting operation on List interface manully.

---

Hierarchy of List interface :

---

Available in the diagram [7th FEB]

---

Behavior of List iterface Specific classes :

---

1) It stores the element beased on the index.

2) Duplicates elements are allowed.

3) We can perform sorting operation.

4) It can accept hetrogeneous and null value.

5) With generic (<>) now we don't have any compilation warning
and we can also take herogeneous by using Object class.

6) IT IS DYNAMICALLY GROWABLE ARRAY so, once the capacity is

full then one new object will be created and all the previous data will be copied into new location and old location is

eligible for GC.[07-FEB-24]

---

08-02-2024

---

Methods of List interface :

---

1) public boolean isEmpty() :- Verify whether List is empty or not.

2) public void clear() :- Will clear all the elements

3) public int size() :- To get the size of the Collections

4) public void add(int index, Object o) :- Insert the element based on the index position.

5) public boolean addAll(int index, Collection c) :- Insert the Collection based on the index position

6) public Object get(int index) :- To retrieve the element based on the index position

7) public Object set(int index, Object o) :- To override or replace the existing element based on the index position

8) public Object remove(int index) :- remove the element based on the index position

9) public boolean remove(Object element) :- remove the element based on the object element, It is the Collection interface method extended by List interface

10) public int indexOf() :- index position of the element

11) public int lastIndex() :- last index position of the element

12) public Iterator iterator() :- To fetch or iterate or retrieve the elements from Collection in forward direction only.

13) public ListIterator listIterator() :- To fetch or iterate or retrieve the elements from Collection in forward and backward direction

---

How many ways we can fetch the object from Collection :

---

There are 7 ways to fetch the Collection object which are as

follows :

1) By using Enumeration interface (JDK 1.0)

2) By using Ordinary for loop (JDK 1.0)

3) By using for-Each loop (JDK 1.5)

4) By using Iterator interface (JDK 1.2)

*5) By using ListIterator interface (JDK 1.2)

*6) By using For Each method (JDK 1.8)

*7) By using Method Reference (JDK 1.8)

Among all these 7 ways Enumeration, Iterator and ListIterator

are the cursor(can move from one direction to another direction)

---

Enumeration :

---

It is a predefined interface available in java.util package from JDK 1.0 onwards.

We can use Enumeration interface to fetch or retrieve the Objects one by one from the Collection because it is a cursor.

We can create Enumeration object by using elements() method of the respective Collection class.

public Enumeration elements();

Enumeration interface contains two method :

---

1) public boolean hasMoreElements() :- It will return true if the Collection is having more elements.

2) public Object nextElement() :- It will return collection object so return type is Object.

---

Iterator interface :

---

It is a predefined interface available in java.util package available from 1.2 version.

It is used to fetch/retrieve the elements from the Collection in forward direction only because it is also a cursor.

public Iterator iterator();

Example :

---

Iterator itr = v.iterator();

Now, Iterator interface has provided two methods

public boolean hasNext() :-

It will verify, the element is available in the next position or not, if available it will return true otherwise it will return false.

public Object next() :- It will return the collection object.

---

ListIterator interface :

---

It is a predefined interface available in java.util package and it is the sub interface of Iterator.

It is used to retrieve the Collection object in both the direction i.e in forward direction as well as in backward direction.

public ListIterator listIterator();

Example :

---

ListIterator lit =  v.listIterator();

1) public boolean hasNext() :-

It will verify the element is available in the next position or not, if available it will return true otherwise it will return false.

2) public Object next() :- It will return the next position collection object.

3) public boolean hasPrevious() :-

It will verify the element is available in the previous position or not, if available it will return true otherwise it will return false.

4) public Object previous () :- It will return the previous position collection object.

Note :- Apart from these 4 methods we have add(), set() and remove() method in ListIterartor interface

By using forEach() method :

---

From java 1.8 onwards every collection class provides a method forEach() method, this method takes Consumer functional interface as a  parameter.

---

WAP to fetch the Collection object by using all 7 ways :

---

```
package com.ravi.colllection;

import java.util.Collections;

import java.util.Enumeration;

import java.util.Iterator;
```

```
import java.util.ListIterator;
import java.util.Vector;

public class CollectionObjectRetrieval
{
  public static void main(String[] args)
  {
        Vector<String> fruits = new Vector<>();
        fruits.add("Orange");
        fruits.add("Mango");
        fruits.add("Apple");
        fruits.add("Grapes");
        fruits.add("Kiwi");

        Collections.sort(fruits);

        System.out.println("FETCHING THROUGH ENUMERATION INTERFACE :");

        Enumeration<String> ele = fruits.elements();

        while(ele.hasMoreElements())
        {
              System.out.println(ele.nextElement());
        }

        System.out.println("FETCHING THROUGH ORDINARY FOR LOOP :");

        for(int i=0; i<fruits.size(); i++)
```

```
	{
		System.out.println(fruits.get(i));
	}

	System.out.println("FETCHING THROUGH FOR EACH LOOP :");

	for(String fruit : fruits)
	{
		System.out.println(fruit);
	}

	System.out.println("FETCHING THROUGH ITERATOR INTERFACE :");

Iterator<String> itr = fruits.iterator();
while(itr.hasNext())
{
	System.out.println(itr.next());
}


System.out.println("FETCHING THROUGH LISTITERATOR INTERFACE :");
ListIterator<String> ltr = fruits.listIterator();

System.out.println("FETCHING THE DATA IN FORWARD DIRECTION");
while(ltr.hasNext())
{
	System.out.println(ltr.next());
}
```

```
   System.out.println("FETCHING THE DATA IN BACKWARD DIRECTION");

   while(ltr.hasPrevious())
   {
         System.out.println(ltr.previous());
   }

  System.out.println("FETCHING THE ELEMENTS THROUGH FOR EACH METHOD");
  fruits.forEach(fruit -> System.out.println(fruit));

  System.out.println("FETCHING THE ELEMENTS THROUGH METHOD REFERENCE");
  fruits.forEach(System.out::println);
 }
}
```

---

How forEach() method works internally ?

---

```
package com.ravi.colllection;

import java.util.Vector;
import java.util.function.Consumer;

public class ForEachMethodInternalWorking {

      public static void main(String[] args)
      {
            Vector<String> fruits = new Vector<>();
```

```
		fruits.add("Orange");
		fruits.add("Mango");
		fruits.add("Apple");

		Consumer<String> cons = new Consumer<String>()
		{
			@Override
			public void accept(String t)
			{
				System.out.println(t.toUpperCase());

			}
		};

		//fruits.forEach(cons);

		fruits.forEach(t-> System.out.println(t.toUpperCase()));

		//new Thread(()-> System.out.println(Thread.currentThread().getName()));

	}

}
```

---

09-02-2024

---

ArrayList :

---

public class ArrayList<E>  extends AbstractList<E> implements List<E>, Serializable, Clonable, RandomAccess

It is a predefined class available in java.util package under List interface.

It accepts duplicate elements and null values.

It is dynamically growable array.

It stores the elements on index basis so it is simillar to dynamic array.

Initial capacity of ArrayList is 10. The new capacity of Arraylist can be calculated by using the  formula

new capacity = (current capacity * 3)/2 + 1

*All the methods declared inside an ArrayList is not synchronized so multiple thread can access the method of ArrayList.

*It is highly suitable for fetching or retriving operation when duplicates are allowed and Thread-safety is not required.

It implements List,Serializable, Clonable, RandomAccess interfcaes

Constructor of ArrayList :

---

In ArrayList we have 3 types of Constructor:

Constructor of ArrayList :

---

We have 3 types of Constructor in ArrayList

1) ArrayList al1 = new ArrayList();

Will create ArrayList object with default capacity 10.

2) ArrayList al2 = new ArrayList(int initialCapacity);

Will create an ArrayList object with user specified Capacity

3) ArrayList al3 = new ArrayList(Collection c)

We can copy any Collection interface implemented class data to the current object reference (Coping one Collection data to another)

---

```
package com.ravi.arraylist;


import java.util.*;
public class ArrayListDemo
{
	public static void main(String... a)
	{
		ArrayList<String> arl = new ArrayList<>();//Generic type
		arl.add("Apple");
		arl.add("Orange");
		arl.add("Grapes");
		arl.add("Mango");
		arl.add("Guava");
		arl.add("Mango");

		Collections.sort(arl);
```

```
                arl.forEach(System.out::println);



        }
}
```

---

```
import java.util.*;
public class ArrayListDemo
{
        public static void main(String... a)
        {
                ArrayList<String> arl = new ArrayList<>();//Generic type
                arl.add("Apple");
                arl.add("Orange");
                arl.add("Grapes");
                arl.add("Mango");
                arl.add("Guava");
                arl.add("Mango");

                System.out.println("Contents :"+arl); //toString()  [Apple,....]

                arl.remove(2); //based on the index position
                arl.remove("Guava"); //based on the Object

                System.out.println("Contents After Removing :"+arl);
                System.out.println("Size of the ArrayList:"+arl.size());

                Collections.sort(arl);
```

```
            arl.forEach(System.out::println);
        }
}
```

---

Record class in java :

---

public abstract class Record extends Object.

It is a new feature introduced from java 17.(In java 14 preview version)

As we know only objects are moving in the network from one place to another place so we need to write BLC class with nessacery requirements to make BLC class as a Data carrier class.

Records are immutable data carrier so, now with the help of record we can send our immutable data from one application to another application.

It is also known as DTO (Data transfer object) OR POJO classes.

It is mainly used to concise our code as well as remove the boiler plate code.

In record, automatically constructor will be generated which is known as canonical constructor and the variables which are known as components are by default final.

In order to validate the outer world data, we can write our own constructor which is known as compact constructor.

Record will automatically generate the implemenation of toString(), equals(Object obj) and hashCode() method.

We can define static and non static method as well as static variable inside the record.
We cannot define instance variable inside the record.

We cann't extend or inherit records because by default every record is implicilty final. It is extending from java.lang.Reocrd class

We can implement an interface by using record.

We don't have setter facility in record because by default variables are final.

---

//Program that compare class and record

3 files :

---

EmployeeClass.java(C)

---

```
package com.ravi.record;

import java.util.Objects;

public class EmployeeClass {
	private Integer employeeId;
	private String employeeName;
	private Double employeeSalary;

	public EmployeeClass(Integer employeeId, String employeeName, Double employeeSalary) {
		super();
```

```
        this.employeeId = employeeId;
        this.employeeName = employeeName;
        this.employeeSalary = employeeSalary;
}

public Integer getEmployeeId() {
        return employeeId;
}

public void setEmployeeId(Integer employeeId) {
        this.employeeId = employeeId;
}

public String getEmployeeName() {
        return employeeName;
}

public void setEmployeeName(String employeeName) {
        this.employeeName = employeeName;
}

public Double getEmployeeSalary() {
        return employeeSalary;
}

public void setEmployeeSalary(Double employeeSalary) {
        this.employeeSalary = employeeSalary;
}
```

```
	@Override
	public int hashCode() {
		return Objects.hash(employeeId, employeeName, employeeSalary);
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		EmployeeClass other = (EmployeeClass) obj;
		return Objects.equals(employeeId, other.employeeId) &&
Objects.equals(employeeName, other.employeeName)
				&& Objects.equals(employeeSalary,
other.employeeSalary);
	}

	@Override
	public String toString() {
		return "EmployeeClass [employeeId=" + employeeId + ",
employeeName=" + employeeName + ", employeeSalary="
				+ employeeSalary + "]";
	}
```

```
}
```

EmployeeRecord.java(R)

---

```
package com.ravi.record;

        //Canonical Constructor
public record EmployeeRecord(int id, String name, double salary)
{
 //Compact Constructor
 public EmployeeRecord
 {
          if(id<0)
                  throw new IllegalArgumentException("Invalid ID");

 }
}
```

EmployeeComparison.java

---

```
package com.ravi.record;

public class EmployeeComparison
{
        public static void main(String[] args)
        {
                EmployeeClass e1 = new EmployeeClass(1, "A", 23.90);
```

```
                EmployeeClass e2 = new EmployeeClass(1, "A", 23.90);
                System.out.println(e1.equals(e2));
                System.out.println(e1);
                System.out.println(e1.getEmployeeName());
                e1.setEmployeeName("AA");
                System.out.println(e1.getEmployeeName());

                System.out.println(".....");
                EmployeeRecord r1 = new EmployeeRecord(2, "B", 25.90);
                EmployeeRecord r2 = new EmployeeRecord(2, "B", 25.90);
                System.out.println(r1.equals(r2));
                System.out.println(r1);
                System.out.println(r1.id());


        }

}
```

---

2 files :

---

Product.java(R)

---

```
package com.ravi.colllection;

public record Product(int id, String name, double price)
{
```

```
}
```

ArrayListDemo.java

---

```
package com.ravi.colllection;

import java.util.ArrayList;

public class ArrayListDemo
{
	public static void main(String[] args)
	{
    ArrayList<Product> al = new ArrayList<>();
    al.add(new Product(1, "Laptop", 78890.90));
    al.add(new Product(2, "Camera", 75890.90));
    al.add(new Product(3, "Mobile", 79890.90));
    al.add(new Product(4, "Iphone", 73890.90));

    al.forEach(p1 -> System.out.println(p1));



	}

}
```

---

```
package com.ravi.arraylist;
```

```
//Program to merge and retain of two collection
import java.util.*;
public class ArrayListDemo2
	{
		public static void main(String args[])
		{
		 ArrayList<String> al1=new ArrayList<>();
		 al1.add("Ravi");
		 al1.add("Rahul");
		 al1.add("Rohit");

		 ArrayList<String> al2=new ArrayList<>();
		 al2.add("Pallavi");
		 al2.add("Sweta");
		 al2.add("Puja");

		 al1.addAll(al2);

	al1.forEach(x -> System.out.println(x.toUpperCase()));

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

		 ArrayList<String> al3=new ArrayList<>();
		 al3.add("Ravi");
		 al3.add("Rahul");
		 al3.add("Rohit");

		 ArrayList<String> al4=new ArrayList<>();
```

```
                    al4.add("Pallavi");
                    al4.add("Rahul");
                    al4.add("Raj");

                    al3.retainAll(al4);

     al3.forEach(x -> System.out.println(x));
  }
}
```

---

```
//Program to fetch the elements in forward and backward
//direction using ListIterator interface

package com.ravi.arraylist;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.ListIterator;

public class ArrayListDemo3
{
public static void main(String args[])
 {
        List<String> listOfName = Arrays.asList("Rohit","Akshar","Pallavi","Sweta");

        Collections.sort(listOfName);
```

```
        //Fetching the data in both the direction
        ListIterator<String> lst = listOfName.listIterator();

        System.out.println("In Forward Direction..");
        while(lst.hasNext())
        {
                System.out.println(lst.next());
        }
        System.out.println("In Backward Direction..");
        while(lst.hasPrevious())
        {
                System.out.println(lst.previous());
        }

 }
}
```

---

* Serialization and De-Serialization :

---

Serialization :

---

It is a technique through which we can store the object data into the file.

In order to provide serialization we should use a predefined class

available in java.io package called ObjectOutputStream which contains a method writeObject()

De-Serialization :

---

It is a technique through which we can read the object data from the file.

In order to provide de-serialization we should use a class called

ObjectInputStream which contains a predefined method readObject(),

return type of this method is Object.

A class on which we want to perform serialization operation that class must implements java.io.Serializable interface.

---

3 files :

---

Employee.java

---

```
package com.ravi.serialization;

import java.io.Serializable;
import java.util.Scanner;

public class Employee implements Serializable
{
	private Integer employeeId;
	private String employeeName;
	private Double employeeSalary;

	public Employee(Integer 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 salary = sc.nextDouble();

    return new Employee(id, name, salary);

	}
```

```
}
```

StoreEmployeeObject.java

---

```
package com.ravi.serialization;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.Scanner;

public class StoreEmployeeObject
{
	public static void main(String[] args) throws IOException
	{
		//Creating a file to write object data
		var fout = new FileOutputStream("C:\\Batch26\\EmpObj.txt");
		var oos = new ObjectOutputStream(fout);
		var sc = new Scanner(System.in);

		try(fout; oos; sc)
		{
			System.out.println("How many objects u want to store");
			int noOfObj = sc.nextInt();

			for(int i=1; i<=noOfObj; i++)
			{
		    Employee object = Employee.getEmployeeObject();
```

```
                    oos.writeObject(object);
                        }
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }
    System.out.println("Employee Object Stored Successfully");
        }

}
```

RetrieveEmployeeObject.java

---

```
package com.ravi.serialization;

import java.io.EOFException;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class RetrieveEmployeeObject
{
        public static void main(String[] args) throws IOException, Exception
        {
                var fin = new FileInputStream("C:\\Batch26\\EmpObj.txt");
                var ois = new ObjectInputStream(fin);
```

```
                try(fin; ois)
                {
                 Employee e = null;
                 while((e = (Employee)ois.readObject())!=null)
                 {
                          System.out.println(e);
                 }
                }
                catch(EOFException e)
                {
                        System.err.println("End if file reached ");
                        e.printStackTrace();
                }
        }
}
```

---

```
//Serialization and De-serialization on ArrayList Object
package com.ravi.arraylist;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;

public class ArrayListDemo4
```

```
{
  public static void main(String [] args) throws IOException, Exception
  {
    ArrayList<String> listOfCity = new ArrayList<>();
    listOfCity.add("Hyderabad");
    listOfCity.add("Nagpur");
    listOfCity.add("Bangalore");
    listOfCity.add("Chennai");

    //Serialization
    var fout = new FileOutputStream("C:\\Batch26\\City.txt");
    var oos = new ObjectOutputStream(fout);
    try(fout; oos)
    {
          oos.writeObject(listOfCity);
    }
    catch(Exception e)
    {
          e.printStackTrace();
    }

    //De-serialization
    var fin = new FileInputStream("C:\\Batch26\\City.txt");
    var ois = new ObjectInputStream(fin);
    ArrayList<String> list = (ArrayList<String>)ois.readObject();
    list.forEach(System.out::println);

  }
```

}

Note :- ArrayList can hold multiple obejcts and we can perform serialization operation on ArrayList object because ArrayList implements Seraializable interface java.io.Serializable interafce.

---

12-02-2024

---

public void ensureCapacity(int minimumCapacity) :

---

It is a predefined method of ArrayList class, by using this method we can resize the capacity of ArrayList Object.

Here by specifying the parameter it ensures that it can hold at least the number of elements specified by the minimum capacity argument.

ArrayList class does not provide capacity() method support.

---

```
package com.ravi.arraylist;

import java.util.ArrayList;
import java.util.LinkedList;

public class ArrayListDemo5
{
	public static void main(String[] args)
	{
		ArrayList<String> city= new ArrayList<>();
```

```
                city.ensureCapacity(15); //resize the capacity of Arraylist
                city.add("Hyderabad");
                city.add("Mumbai");
                city.add("Delhi");

                city.add("Kolkata");
                System.out.println("ArrayList: " + city);
        }
}
```

---

```
package com.ravi.arraylist;

//Program on ArrayList that contains null values as well as we can pass the element based on the index position
import java.util.*;
public class ArrayListDemo6
{
        public static void main(String[] args)
        {

                ArrayList<Object> al = new ArrayList<>(); //Generic type
                al.add(12);
                al.add("Ravi");
                al.add(12);
                al.add(3,"Hyderabad"); //add(int index, Object o)method of List interface
                al.add(1,"Naresh");
                al.add(null);
```

```
                al.add(11);
                System.out.println(al);  //12 Naresh Ravi  12  Hyderabad
        }
}
```

---

Assignment :

---

Create a Record called Product, create an ArrayList object to store product object (minimum 5 product object). Perform

Serialization and De-serialization operation on ArrayList object.

---

Limitation of ArrayList :

---

The time complexcity of ArrayList to insert and delete an element from the middle would be O(n) because 'n' number of elements will be re-located so it is not a good choice to perform insertion and deletion operation in the middle of the List. [12-FEB-24]

To avoid this we introduced LinkedList.

---

LinkedList :

---

public class LinkedList<E> extends AbstractSequentialList<E> implements List<E>, Deque<E>, Cloneable, Serializable

It is a predefined class available in java.util package under List interface.

It is ordered by index position like ArrayList except the elements (nodes) are doubly linked to one another. This linkage provide us new method for adding and removing the elements from the middle of LinkedList.

*The important thing is, LikedList may iterate more slowely than ArrayList but LinkedList is a good choice when we want to insert or delete the elements frequently in the list.

From jdk 1.6 onwards LinkedList class has been enhanced to support basic queue operation by implementing Deque<E> interface.

ArrayList is using Array data structure but LinkedList class is using LinkedList data structure.

Constructor:

---

It has 2 constructors

1) LinkedList list1 = new LinkedList();

   It will create a LinkedList object with 0 capacity.

2) LinkedList list2 = new LinkedList(Collection c);

   Interconversion between the collection

Methods of LinkedList class:

---

1) void addFirst(Object o)

2) void addLast(Object o)

3) Object getFirst()

4) Object getLast()

5) Object removeFirst()

6) Object removeLast()

Note :- It stores the elements in non-contiguous memory location.

The time complexcity for insertion and deletion is  O(1)

The time complexcity for seraching O(n)

---

```
package com.ravi.linked_list;

import java.util.LinkedList;
import java.util.List;
public class LinkedListDemo
{
 public static void main(String args[])
 {
    List<Object> list=new LinkedList<>();
            list.add("Ravi");
            list.add("Vijay");
            list.add("Ravi");
            list.add(null);
            list.add(42);

            System.out.println("1st Position Element is :"+list.get(1));

           //Iterator interface

             Iterator<Object> itr = list.iterator();
```

```
            itr.forEachRemaining(System.out::println); //JDK 1.8
 }
}
```

---

```
package com.ravi.linked_list;

import java.util.*;
public class LinkedListDemo1
{
    public static void main(String args[])
    {
        LinkedList<String> list= new LinkedList<>(); //generic
        list.add("Item 2");//2
        list.add("Item 3");//3
        list.add("Item 4");//4
        list.add("Item 5");//5
        list.add("Item 6");//6
        list.add("Item 7");//7

        list.add("Item 9"); //10

        list.add(0,"Item 0");//0
        list.add(1,"Item 1"); //1

        list.add(8,"Item 8");//8
                    list.add(9,"Item 10");//9
         System.out.println(list);
```

```
                        list.remove("Item 5");

                        System.out.println(list);

                         list.removeLast();
                         System.out.println(list);

                         list.removeFirst();
                         System.out.println(list);

                        list.set(0,"Ajay"); //set() will replace the existing value
                        list.set(1,"Vijay");
                        list.set(2,"Anand");
                        list.set(3,"Aman");
                        list.set(4,"Suresh");
                        list.set(5,"Ganesh");
                        list.set(6,"Ramesh");
                        list.forEach(x -> System.out.println(x));


    }
}
```

Note :- We can perform frequent insertion and deletion opeartion
on LinkedList

---

```
package com.ravi.linked_list;
```

```
//Methods of LinkedList class
import java.util.LinkedList;
public class LinkedListDemo2
{
   public static void main(String[] argv)
   {
       LinkedList<String> list = new LinkedList<>();

       list.addFirst("Ravi");
       list.add("Rahul");
       list.addLast("Anand");

       System.out.println(list.getFirst());
       System.out.println(list.getLast());

       list.removeFirst();
       list.removeLast();

       System.out.println(list);
   }
}
```

---

```
package com.ravi.linked_list;
//ListIterator methods
import java.util.*;
public class LinkedListDemo3
{
        public static void main(String[] args)
```

```
    {
            LinkedList<String> city = new LinkedList<> ();
 city.add("Kolkata");
             city.add("Bangalore");
             city.add("Hyderabad");
             city.add("Pune");
             System.out.println(city);

            ListIterator<String> lt = city.listIterator();

while(lt.hasNext())
             {
                    String cityName =  lt.next();

                    if(cityName.equals("Kolkata"))
                    {
      lt.remove();
                    }
                    else if(cityName.equals("Hyderabad"))
                    {
      lt.add("Ameerpet");
                    }
                    else if(cityName.equals("Pune"))
                    {
      lt.set("Mumbai");
                    }
            }
            city.forEach(System.out::println);
```

```
        }
}
```

---

```
package com.ravi.linked_list;

//Insertion, deletion, displaying and exit

import java.util.LinkedList;
import java.util.Scanner;

public class LinkedListDemo4
{
 public static void main(String[] args)
        {
    LinkedList<Integer> linkedList = new LinkedList<>();
    Scanner scanner = new Scanner(System.in);

     while (true)
                {
      System.out.println("Linked List: " + linkedList);
      System.out.println("1. Insert Element");
      System.out.println("2. Delete Element");
                System.out.println("3. Display Element");
      System.out.println("4. Exit");
      System.out.print("Enter your choice: ");

      int choice = scanner.nextInt();
      switch (choice)
```

```
                    {
        case 1:
          System.out.print("Enter the element to insert: ");
          int elementToAdd = scanner.nextInt();
          linkedList.add(elementToAdd);
          break;
        case 2:
          if (linkedList.isEmpty())
                                        {
            System.out.println("Linked list is empty. Nothing to delete.");
          }
                                        else
                                        {
            System.out.print("Enter the element to delete: ");
            int elementToDelete = scanner.nextInt();
            //Converting the primitive to Wrapper object
             boolean remove = linkedList.remove(Integer.valueOf(elementToDelete));

             if(remove)
             {
                  System.out.println("Element "+elementToDelete+ " is deleted
Successfully" );
             }
             else
             {
                  System.out.println(elementToDelete+" not available is the LinkedList");
             }
```

```
            }
            break;
                                    case 3:
                                                System.out.println("Elements in the linked list.");
             System.out.println(linkedList);
                                                  break;
          case 4:
            System.out.println("Exiting the program.");
            scanner.close();
            System.exit(0);
          default:
            System.out.println("Invalid choice. Please try again.");
       }
    }
 }
}
```

---

```
package com.ravi.linked_list;

import java.util.Arrays;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;

public class LinkedListDemo5 {

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

```
        {
                List<String> listOfName = Arrays.asList("Ravi","Rahul","Ankit", "Rahul");

                LinkedList<String> list = new LinkedList<String>(listOfName);
                list.forEach(System.out::println);
                System.out.println(".....");

        }

}
```

---

```
import java.util.LinkedList;
import java.util.Iterator;
import java.util.List;

class Dog
{
    public String name;
    Dog(String n)
    {
        name = n;
    }

   public String toString()
        {
                return this.name;
        }
}
```

```
public class LinkedListDemo5
{
    public static void main(String[] args)
    {
        List<Dog> d = new LinkedList<>();
        Dog dog = new Dog("Tiger");
        d.add(dog);
        d.add(new Dog("Tommy"));
        d.add(new Dog("Rocky"));

        Iterator<Dog> i3 = d.iterator();
                i3.forEachRemaining(x -> System.out.println(x.name)); //java 8

        System.out.println("size " + d.size());
        System.out.println("Get 1st Position Object " + d.get(1).name);

    }
}
```

---

13-02-2024

---

Vector :

---

public class Vector<E> extends AbstractList<E>  implements List<E>, Serializable, Clonable, RandomAccess

Vector is a predefined class available in java.util package under List interface.

Vector is always from java means it is available from jdk 1.0 version.

Vector and Hashtable, these two classes are available from jdk 1.0, remaining Collection classes were added from 1.2 version. That is the reason Vector and Hashtable are called legacy(old) classes.

The main difference between Vector and ArrayList is, ArrayList methods are not synchronized so multiple threads can access the method of ArrayList where as on the other hand most the methods are synchronized in Vector so performance wise Vector is slow.

*We should go with ArrayList when Threadsafety is not required on the other hand we should go with Vector when we need ThreadSafety for reterival operation.

It also stores the elements on index basis.It is dynamically growable with initial capacity 10. The next capacity will be 20 i.e double of the first capacity.

new capacity = current capacity * 2;

Just like ArrayList it also implements List, Serializable, Clonable, RandomAccess interfaces.

Constructors in Vector :

---

We have 4 types of Constructor in Vector

1) Vector v1 = new Vector();

It will create the vector object with default capacity is 10

2) Vector v2 = new Vector(int initialCapacity);

Will create the vector object with user specified capacity.

3) Vector v3 = new Vector(int initialCapacity, int incrementalCapacity);

Eg :- Vector v = new Vector(1000,5);

Initially It will create the Vector Object with initial capacity 1000 and then when the capacity will be full then increment by 5 so the next capacity would be 1005, 1010 and so on.

4) Vector v4 = new Vector(Collection c);

Interconversion between the Collection.

---

//Vector Program on capacity

```
package com.ravi.vector;

import java.util.*;
public class VectorDemo1
{
	public static void main(String[] args)
	{
		Vector<Integer> v = new Vector<>(100, 10);  //initial capacity is 100
		System.out.println("Initial capacity is :"+v.capacity());

    for(int i = 0; i<100; i++)
		{
```

```
		v.add(i);
			}

System.out.println("After adding 100 elements  capacity is :"+v.capacity());
v.add(101);
System.out.println("After adding 101th elements  capacity is :"+v.capacity());

			for(Integer i : v)
			{
				System.out.print(i+"\t");
				if(i%5==0)
					System.out.println();
			}
		}
}
-----------------------------------------------------------------
package com.ravi.vector;

import java.util.Vector;

record Product(int pid, String pname, double price)
{
		public Product
		{
			if(pid < 0)
				throw new IllegalArgumentException("Product Id is Invalid");
		}
```

```
}

public class VectorDemo2
{
	public static void main(String[] args)
	{
		Vector<Product> v1 = new Vector<>();
		v1.add(new Product(1, "Camera", 45000));
		v1.add(new Product(2, "Laptop", 65000));
		v1.add(new Product(3, "Mobile", 25000));
		v1.add(new Product(4, "LED", 85000));
		v1.add(new Product(5, "Cooler", 32000));
		v1.add(new Product(6, "AC", 55000));

		v1.stream().filter(p -> p.price()<30000).forEach(System.out::println);


	}
}
```

---

```
package com.ravi.vector;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
```

```
public class StreamTest
{
	public static void main(String[] args)
	{
    List<Integer> numbers = Arrays.asList(1,22,3,44,5,6,7,28,9,10,11,22,2,4);

    //Storing all the even numbers in the list with Stream API

    List<Integer> even = new ArrayList<>();

    for(Integer i : numbers)
    {
         if(i%2==0)
                even.add(i);
    }
    System.out.println(even);

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

   //Storing all the even numbers in the list with Stream API (Including duplicates)

     List<Integer> collect = numbers.stream().filter(num ->
num%2==0).sorted().collect(Collectors.toList());
   System.out.println(collect);
```

```
	}

}
```

---

```
package com.ravi.vector;
//Array To Collection
import java.util.*;
public class VectorDemo3
{
	public static void main(String args[])
	{
		Vector<Integer> v = new Vector<>();

		int x[]={22,20,10,40,15,58};

	//Adding array values to Vector
		for(int i=0; i<x.length; i++)
		{
			v.add(x[i]);
		}
		Collections.sort(v);
		System.out.println("Maximum element is :"+Collections.max(v));
		System.out.println("Minimum element is :"+Collections.min(v));
		System.out.println("Vector Elements :");
		v.forEach(y -> System.out.println(y));
```

```
	}
}
```

---

```
//Program to describe that ArrayList is better than Vector in performance

package com.ravi.vector;

import java.util.ArrayList;
import java.util.Vector;

public class VectorDemo4
{
	public static void main(String[] args)
	{
		long startTime = System.currentTimeMillis();

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

		for(int i=0 ; i<=1000000; i++)
		{
			al.add(i);
		}

		long endTime = System.currentTimeMillis();
		System.out.println("Total time taken by ArrayList :"+(endTime-startTime)+
"ms");

	startTime = System.currentTimeMillis();
```

```
		Vector<Integer> v = new Vector<>();


		for(int i=0 ; i<=1000000; i++)
		{
			v.add(i);
		}


		endTime = System.currentTimeMillis();
		System.out.println("Total time taken by Vector :"+(endTime-startTime)+
"ms");


	}
}
```

---

```
package com.ravi.vector;

import java.util.Arrays;
import java.util.List;

public class Vector5 {

	public static void main(String[] args)
	{
		List<String> names =
Arrays.asList("Scott","Smith","Akshar","Rahul","Sunny");
		names.stream().filter(str ->
str.startsWith("S")).sorted().forEach(System.out::println);
	}
```

}

---

15-02-2024

---

Stack :

---

public class Stack<E> extends Vector<E>

It is a predefined class available in java.util package. It is the sub class of Vector class introduced from JDK 1.0 so, It is also a legacy class.

It is a linear data structure that is used to store the Objects in LIFO (Last In first out) order.

Inserting an element into a Stack is known as push operation  where as extracting an element from the top of the stack is known as pop operation.

It throws an exception called EmptyStackException, if Stack is empty and we want to fetch the element.

It has only one constructor as shown below

Stack s = new Stack();

---

Methods :

---

push(Object o) :- To insert an element.

pop() :- To remove and return the element from the top of the Stack.

peek() :- Will fetch the element from top of the Stack without removing.

empty() :- Verifies whether the stack is empty or not (return type is boolean)

search(Object o) :- It will search a particular element in the Stack and it returns OffSet position (int value). If the element is not present in the Stack it will return -1

---

```
//Program to insert and fetch the elements from stack
import java.util.*;
public class Stack1
{
    public static void main(String args[])
    {
        Stack<Integer> s = new Stack<>();
       try
        {
            s.push(12);
            s.push(15);
                                    s.push(22);
                                    s.push(33);
                                    s.push(49);
                                    System.out.println("After insertion elements are :"+s);

            System.out.println("Fetching the elements using pop method");
            System.out.println(s.pop());
            System.out.println(s.pop());
```

```
            System.out.println(s.pop());
            System.out.println(s.pop());
            System.out.println(s.pop());



                                    System.out.println("After deletion elements are :"+s); //[]
                                    System.out.println("Is the Stack empty ? :"+s.empty());
        }
                            catch(EmptyStackException e)
                            {
                              e.printStackTrace();
                            }
    }
}
```

---

```
//add(Object obj) is the method of Collection
import java.util.*;
public class Stack2
{
    public static void main(String args[])
    {
        Stack<Integer> st1 = new Stack<>();
        st1.add(10);
        st1.add(20);
        st1.forEach(x -> System.out.println(x));

        Stack<String> st2 = new Stack<>();
        st2.add("Java");
```

```
        st2.add("is");
        st2.add("programming");
        st2.add("language");
        st2.forEach(x -> System.out.println(x));

        Stack<Character> st3 = new Stack<>();
        st3.add('A');
        st3.add('B');
        st3.forEach(x -> System.out.println(x));

        Stack<Double> st4 = new Stack<>();
        st4.add(10.5);
        st4.add(20.5);
        st4.forEach(x -> System.out.println(x));
    }
}
```

---

```
import java.util.Stack;
public class Stack3
{
        public static void main(String[] args)
                {
                        Stack<String> stk= new Stack<>();
                        stk.push("Apple");
                        stk.push("Grapes");
                        stk.push("Mango");
                        stk.push("Orange");
                        System.out.println("Stack: " + stk);
```

```
                String fruit = stk.peek();
                System.out.println("Element at top: " + fruit);
                System.out.println("Stack elements are : " + stk);
        }
}
```

---

```
//Searching an element in the Stack
import java.util.Stack;
public class Stack4
{
    public static void main(String[] args)
        {
            Stack<String> stk= new Stack<>();
            stk.push("Apple");
            stk.push("Grapes");
            stk.push("Mango");
            System.out.println("Offset Position is : " + stk.search("Mango"));
//1
            System.out.println("Offset Position is : " + stk.search("Banana"));
//-1
          System.out.println("Is stack empty ? "+stk.empty());     //false

            System.out.println("Index Position is : " + stk.indexOf("Mango"));
//2
        }
}
```

---

Set interface :

---

It is the sub interface of Collection.

Set interface will never accept duplicate elements, Here our best friend is equals(Object obj) method of Object class. If two objects are identical then it will accept only one object.

Set interface does not follow index and we cannot read the data in backward direction (ListIterator will not work)

We cannot perform manual sorting operation using Set interface but the same thing will be available using Comparator interface.

Set interface uses almost all the method of Collection interface.

---

Hierarchy of Set interface [15-FEB-24]

---

HashSet (UNSORTED, UNORDERED , NO DUPLICATES)

---

public class HashSet<E> extends AbstractSet<E> implements Set<E>, Clonabale, Serializable

It is a predefined class available in java.util package under Set interface and introduced from JDK 1.2V.

It is an unsorted and unordered set.

It accepts hetrogeneous kind of data.

*It uses the hashcode of the object being inserted into the Collection. Using this hashcode it finds the bucket location.

It doesn't contain any duplicate elements as well as It does not maintain any order while iterating the elements from the collection.

It can accept null value.

HashSet is used for fast searching operation.

It contains 4 types of constructors

1) HashSet hs1 = new HashSet();

It will create the HashSet Object with default capacity is 16. The default load fator or Fill Ratio is 0.75 (75% of HashSet is filled up then new HashSet Object will be created having double capacity)

2) HashSet hs2 = new HashSet(int initialCapacity);

will create the HashSet object with user specified capacity

3) HashSet hs3 = new HashSet(int initialCapacity, float loadFactor);

we can specify our own initialCapacity and loadFactor(by default load factor is 0.75%)

4) HashSet hs = new HashSet(Collection c);

Interconversion of Collection

---

```
//Unsorted, Unordered and no duplicates
import java.util.*;
public class HashSetDemo
```

```
{
 public static void main(String args[])
 {
            HashSet<Integer> hs = new HashSet<>();
                    hs.add(67);
                    hs.add(89);
                    hs.add(33);
                    hs.add(45);
                    hs.add(12);
                    hs.add(35);
                    hs.forEach(str-> System.out.println(str));
          }
}
```

---

16-02-2024

---

```
import java.util.*;
public class HashSetDemo1
{
    public static void main(String[] argv)
    {
    HashSet<String> hs=new HashSet<>();
            hs.add("Ravi");
            hs.add("Vijay");
            hs.add("Ravi");
            hs.add("Ajay");
            hs.add("Palavi");
            hs.add("Sweta");
```

```
        hs.add(null);
        hs.add(null);
        hs.forEach(str -> System.out.println(str));

    }
}
```

Note :- Duplicates are not allowed and will accept only one null.

---

```
package com.ravi.set_interface;

import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

public class HashSetDemo2 {

        public static void main(String[] args)
        {
                Set<Object> set = new HashSet<>();

                boolean []arr = new boolean[5];

                arr[0] = set.add(10);
                arr[1] = set.add(56.90);
                arr[2] = set.add(new String("NIT"));
                arr[3] = set.add(10);
```

```
                arr[4] = set.add(20);

                System.out.println(Arrays.toString(arr));

                if(set.contains(20))
                {
                        System.out.println("Object 20 is available");
                }
                else
                {
                        System.out.println("Object 20 is not available");
                }

                set.forEach(System.out::println);
        }
}
```

add(Object obj) method return type is boolean so, if set interface is unable to add an element in the set collection then this add() method will return false.

public boolean contains(Object obj) : It is a predefined method

to search an element in the available collection, if available it will return true otherwise it will return false.

---

```
import java.util.HashSet;
import java.util.Scanner;

public class HashSetDemo3
```

```
{
   public static void main(String[] args)
                    {
      HashSet<String> hashSet = new HashSet<>();
      Scanner scanner = new Scanner(System.in);

      while (true)
                    {
         System.out.println("Options:");
         System.out.println("1. Add element");
         System.out.println("2. Delete element");
         System.out.println("3. Display HashSet");
         System.out.println("4. Exit");

         System.out.print("Enter your choice (1/2/3/4): ");
         int choice = scanner.nextInt();

         switch (choice)
                        {
            case 1:
               System.out.print("Enter the element to add: ");
               String elementToAdd = scanner.next();
               if (hashSet.add(elementToAdd))
                                    {
                  System.out.println("Element added successfully.");
               }
                                    else
                                    {
```

```
            System.out.println("Element already exists in the HashSet.");
        }
        break;
        case 2:
        System.out.print("Enter the element to delete: ");
        String elementToDelete = scanner.next();
        if (hashSet.remove(elementToDelete))
                                        {
            System.out.println("Element deleted successfully.");
        }
                                        else
                                        {
            System.out.println("Element not found in the HashSet.");
        }
        break;
        case 3:
        System.out.println("Elements in the HashSet:");
        for (String element : hashSet) {
            System.out.println(element);
        }
        break;
        case 4:
        System.out.println("Exiting the program.");
        scanner.close();
        System.exit(0);
        default:
        System.out.println("Invalid choice. Please try again.");
    }
```

```
        System.out.println();
      }
   }
}
```

---

LinkedHashSet :

---

public class LinkedHashSet extends HashSet implements Set, Clonable, Serializable

It is a predefined class in java.util package under Set interface and introduced from java 1.4v.

It is the sub class of HashSet class.

It is an orderd version of HashSet that maintains a doubly linked list across all the elements.

It internally uses Hashtable and LinkedList data structures.

We should use LinkedHashSet class when we want to maintain an order.

When we iterate the elements through HashSet the order will be unpredictable, while when we iterate the elments through LinkedHashSet then the order will be same as they were inserted in the collection.

It accepts hetrogeneous and null value is allowed.

It has same constructor as HashSet class.

```
---------------------------------------------------------------------
import java.util.*;
public class LinkedHashSetDemo
{
 public static void main(String args[])
        {
                LinkedHashSet<String> lhs=new LinkedHashSet<>();
                lhs.add("Ravi");
                lhs.add("Vijay");
                lhs.add("Ravi");
                lhs.add("Ajay");
                lhs.add("Pawan");
                lhs.add("Shiva");
                lhs.add(null);
                lhs.add("Ganesh");
                lhs.forEach(str -> System.out.println(str));
        }
}
---------------------------------------------------------------------
import java.util.LinkedHashSet;

public class LinkedHashSetDemo1
{
   public static void main(String[] args)
        {
     LinkedHashSet<Integer> linkedHashSet = new LinkedHashSet<>();

      linkedHashSet.add(10);
```

```
        linkedHashSet.add(5);
        linkedHashSet.add(15);
        linkedHashSet.add(20);
        linkedHashSet.add(5);

        System.out.println("LinkedHashSet elements: " + linkedHashSet);

        System.out.println("LinkedHashSet size: " + linkedHashSet.size());

        int elementToCheck = 15;
        if (linkedHashSet.contains(elementToCheck))
                        {
           System.out.println(elementToCheck + " is present in the LinkedHashSet.");
        }
                        else
                        {
           System.out.println(elementToCheck + " is not present in the LinkedHashSet.");
        }

        int elementToRemove = 10;
        linkedHashSet.remove(elementToRemove);
        System.out.println("After removing " + elementToRemove + ", LinkedHashSet
elements: " + linkedHashSet);

            linkedHashSet.clear();
        System.out.println("After clearing, LinkedHashSet elements: " + linkedHashSet); //[]
    }
}
```

---

SortedSet :

---

It is the sub interface of Set interface which is introduced from

JDK 1.2V.

As we know we can't perform sorting opeartion on HashSet and LinkedHashSet class manually so, if we don't want duplicate elements and want to store the element in sorted order then we should use SortedSet interface.

SortedSet interface provides default natural sorting order, default natural sorting order means, if numbers then ascending order, if String then Alphabetical order OR Dictionary order.

In order to sort the elements in natural sorting order as well as user-defined sorting we have two interfaces java.lang.Comparable and java.util.Comparator.

---

*** What is the difference between Comparable and Comparator

Program on Comparable ineterface :

---

2 Files :

---

Employee.java(R)

---

```
package com.ravi.comparable;

public record Employee(Integer eid, String ename, Double salary) implements
Comparable<Employee>
{
```

```
   //Sorting Employee Object based on Employee Name
        @Override
        public int compareTo(Employee e1)
        {
                return this.ename.compareTo(e1.ename);
        }
}
```

EmployeeComparable.java

---

```
package com.ravi.comparable;

import java.util.ArrayList;
import java.util.Collections;

public class EmployeeComparable
{
        public static void main(String[] args)
        {
                ArrayList<Employee> al = new ArrayList<>();
                al.add(new Employee(3,"Aman",23890.90));
                al.add(new Employee(1,"Zuber",25890.90));
                al.add(new Employee(2,"Yuvraj",27890.90));
                al.forEach(System.out::println);
                Collections.sort(al);
                System.out.println("After Sorting based on the Name ");
                al.forEach(System.out::println);
```

```
        }

}
```

---

Limitation of Comparable interface :

---

1) We need to modify the original source code (BLC class), If the source code is not available then it is not possible to perform  sorting operation.

2) We can provide only one sorting logic if we want to provide mutiple sorting logic then it is not possible.

To avoid the above said problems we introduced Comparator interface available in java.util package.

---

//Program on Comparator interface

Product.java(R)

---

```
package com.ravi.comparator;

public record Product(Integer pid, String pname, Double price)
{

}
```

ProductComparator.java

---

```
package com.ravi.comparator;
```

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class ProductComparator
{
	public static void main(String[] args)
	{
		ArrayList<Product> al = new ArrayList<>();
		al.add(new Product(3, "Camera", 12890.90));
		al.add(new Product(1, "Laptop", 85890.90));
		al.add(new Product(2, "Mobile", 43890.90));

		Comparator<Product> cmpId = new Comparator<Product>()
		{
			@Override
			public int compare(Product p1, Product p2)
			{
				return p1.pid() - p2.pid();
			}
		};

		Collections.sort(al, cmpId);
		System.out.println("Sorting based on the ID");
		al.forEach(System.out::println);

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

```
		Comparator<Product> cmpName = (p1, p2)-
>p1.pname().compareTo(p2.pname());
		Collections.sort(al, cmpName);
		System.out.println("Sorting based on the Name");
		al.forEach(System.out::println);

	}

}
```

---

19-02-2024

---

Program that describes the descending order of Integer object.

```
package com.ravi.java8;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class DescendingInteger
{
	public static void main(String[] args)
	{
		ArrayList<Integer> al = new ArrayList<>();
		al.add(78);
		al.add(44);
```

```
                al.add(23);
                al.add(12);

                Comparator<Integer> desc = (i1, i2)-> -(i1 - i2);

                Collections.sort(al, desc);

                al.forEach(System.out::println);
        }
}
```

---

TreeSet :

---

public class TreeSet<E> extends AbstractSet<E> implements NavigableSet<E>, Clonable, Serializable

It is a predefined class available in java.util package under Set interface.

TreeSet, TreeMap and PriorityQueue are the three sorted collection in the entire Collection Framework so these classes never accepting non comparable objects.

It will sort the elements in natural sorting order i.e ascending order in case of number , and alphabetical order or Dictionary order in the case of String. In order to sort the elements according to user choice, It uses Comparator interface.

It does not accept duplicate and null value(java.lang.NullPointerException).

It does not accept non comparable type of data if we try to insert it will throw a runtime exception i.e java.lang.ClassCastException

TreeSet implements NavigableSet.

NavigableSet extends SortedSet.

It contains 4 types of constructors :

---

1) TreeSet t1 = new TreeSet();

create an empty TreeSet object, elements will be inserted in a natural sorting order.

2) TreeSet t2 = new TreeSet(Comparator c);

Customized sorting order

3) TreeSet t3 = new TreeSet(Collection c);

4) TreeSet t4 = new TreeSet(SortedSet s);

---

```
//program that describes TreeSet provides default natural sorting order
import java.util.*;
public class TreeSetDemo
{
	public static void main(String[] args)
	{
		SortedSet<Integer> t1 = new TreeSet<>();
		t1.add(4);
		t1.add(7);
		t1.add(2);
		t1.add(1);
```

```
                t1.add(9);



                System.out.println(t1);


                NavigableSet<String> t2 = new TreeSet<>();
                t2.add("Orange");
                t2.add("Mango");
                t2.add("Banana");
                t2.add("Grapes");
                t2.add("Apple");
                System.out.println(t2);
        }
}
```

Note : Integer and String both classes are implementing java.lang.Comparable so internally compareTo() is invoked the

compare the Integer and String Object.

---

```
import java.util.*;
public class TreeSetDemo1
{
        public static void main(String[] args)
        {
                TreeSet<String> t1 = new TreeSet<>();
                t1.add("Orange");
                t1.add("Mango");
                t1.add("Pear");
```

```
                t1.add("Banana");
                t1.add("Apple");
                System.out.println("In Ascending order");
                t1.forEach(i -> System.out.println(i));

                TreeSet<String> t2 = new TreeSet<>();
                t2.add("Orange");
                t2.add("Mango");
                t2.add("Pear");
                t2.add("Banana");
                t2.add("Apple");

    System.out.println("In Descending order");
                Iterator<String> itr2 = t2.descendingIterator();  //for descending order

     itr2.forEachRemaining(x -> System.out.println(x));
        }
}
```

Note :- descendingIterator() is a predefined method of TreeSet class which will traverse in the descending order and return type of this method is Iterator interface.

---

```
import java.util.*;
public class TreeSetDemo2
{
        public static void main(String[] args)
        {
                Set<String> t = new TreeSet<>();
```

```
                t.add("6");
                t.add("5");
                t.add("4");
                t.add("2");
                t.add("9");
                Iterator<String> iterator = t.iterator();
                iterator.forEachRemaining(x -> System.out.println(x));

                //From 1.8 to replace hasNext() and next() method
        }
}
```

---

```
import java.util.*;

public class TreeSetDemo3
        {
        public static void main(String[] args)
        {
                Set<Character> t = new TreeSet<>();
                t.add('A');
                t.add('C');
                t.add('B');
                t.add('E');
                t.add('D');
                Iterator<Character> iterator = t.iterator();
                iterator.forEachRemaining(x -> System.out.println(x));
        }
}
```

---

In the following program we are comparing Student object based on the name so Student class is implementing Comparable interface.

```
package com.ravi.java8;

import java.util.TreeSet;

class Student implements Comparable<Student>
{

	int id;
	String name;
	public Student(int id, String name) {
		super();
		this.id = id;
		this.name = name;
	}

	@Override
	public String toString() {
		return "Student [id=" + id + ", name=" + name + "]";
	}


	@Override
	public int compareTo(Student s2)
	{
```

```
                return this.name.compareTo(s2.name);
        }




}

public class DescendingInteger
{
        public static void main(String[] args)
        {
                TreeSet<Student> ts = new TreeSet<>();
                ts.add(new Student(1,"Z"));
                ts.add(new Student(3,"A"));
                ts.add(new Student(2,"B"));

                System.out.println(ts);



        }



}
```

---

Program on Comparator passed as Constructor to TreeSet object.

```
package com.ravi.java8;
```

```
import java.util.TreeSet;

record Employee(int id, String name)
{

}

public class TreeSetConstructor {

	public static void main(String[] args)
	{
		TreeSet<Employee> ts1 = new TreeSet<>((e1, e2)-> e1.id()-e2.id());
		ts1.add(new Employee(333, "A"));
		ts1.add(new Employee(111, "Z"));
		ts1.add(new Employee(222, "R"));

		System.out.println("Employee Sorted based on ID :");
		ts1.forEach(System.out::println);


		TreeSet<Employee> ts2 = new TreeSet<>((e1, e2)-> -(e1.id()-e2.id()));
		ts2.add(new Employee(333, "A"));
		ts2.add(new Employee(111, "Z"));
		ts2.add(new Employee(222, "R"));

		System.out.println("Employee Sorted based on ID In Descending Order
:");
		ts2.forEach(System.out::println);
```

```
		TreeSet<Employee> ts3 = new TreeSet<>((e1, e2)->
e1.name().compareTo(e2.name()));

		ts3.add(new Employee(333, "A"));

		ts3.add(new Employee(111, "Z"));

		ts3.add(new Employee(222, "R"));


		System.out.println("Employee Sorted based on Name :");

		ts3.forEach(System.out::println);


		TreeSet<Employee> ts4 = new TreeSet<>((e1, e2)-> -
e1.name().compareTo(e2.name()));

		ts4.add(new Employee(333, "A"));

		ts4.add(new Employee(111, "Z"));

		ts4.add(new Employee(222, "R"));


		System.out.println("Employee Sorted based on Name Descending order
:");

		ts4.forEach(System.out::println);

	}


}
```

---

20-02-2024

---

Methods of SortedSet interface :

---

public E first() :- Will fetch first element

public E last() :- Will fetch last element

public SortedSet headSet(int range) :- Will fetch the values which are less than specified range

public SortedSet tailSet(int range) :- Will fetch the values which are equal or greater than the specified range.

public SortedSet subSet(int startRange, int endRange) :- Will fetch the range of values where startRange is inclusive and endRange is exclusive

Note :- headSet(), tailSet() and subSet(), return type is SortedSet

---

```
import java.util.*;
public class SortedSetMethodDemo
{
    public static void main(String[] args)
    {
        TreeSet<Integer> times = new TreeSet<>();
        times.add(1205);
        times.add(1505);
        times.add(1545);
                        times.add(1600);
        times.add(1830);
        times.add(2010);
        times.add(2100);

        SortedSet<Integer> sub = new TreeSet<>();
```

```
                    sub = times.subSet(1545,2100);
        System.out.println("Using subSet() :-"+sub);//[1545, 1600,1830,2010]
        System.out.println(sub.first());
        System.out.println(sub.last());

                sub = times.headSet(1545);
                    System.out.println("Using headSet() :-"+sub); //[1205, 1505]

                sub =  times.tailSet(1545);
                    System.out.println("Using tailSet() :-"+sub); //[1545 to 2100]
    }
}
```

---

NavigableSet(I) :

---

With the help of SortedSet interface method we can find out the range of values but we can't navigate among those elements.

Now to frequently navigate among those range of elements, Java software people introduced new interface called NavigableSet from 1.6V

NavigableSet extends SortedSet

---

```
import java.util.*;

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

```
NavigableSet<Integer> ns = new TreeSet<>();
ns.add(1);
ns.add(2);
ns.add(3);
ns.add(4);
ns.add(5);
ns.add(6);

System.out.println("lower(3): " + ns.lower(3));//Just below than the specified element or null

System.out.println("floor(3): " + ns.floor(3)); //Equal  less or null

System.out.println("higher(3): " + ns.higher(3));//Just greater than specified element or null

System.out.println("ceiling(3): " + ns.ceiling(3));//Equal or greater or null

  }
}
```

---

Map interface :

---

As we know Collection interface is used to hold single Or individual object but Map interface will hold group of objects in the form key and value pair. {key = value}

Map interface is not the part the Collection.

Before Map interface We had Dictionary(abstract class) class and it is extended by Hashtable class in JDK 1.0V

Map interface works with key and value pair introduced from 1.2V.

Here key and value both are objects.

Here key must be unique and value may be duplicate.

Each key and value pair is creating one Entry.(Entry is nothing but the combination of key and value pair)

```
interface Map
{
    interface Entry
    {
      //key and value
    }
}
```

How to represent this entry interface (Map.Entry in .java) [Map$Entry in .class]

In Map interface whenever we have a duplicate key then the old key value will be replaced by new key(duplicate key) value.

---

Map interface Hierarchy :

---

Available in the diagram [20-FEB-24]

---

Methods of Map interface :

---

1) Object put(Object key, Object value) :- To insert one entry in the Map collection. It will return the old object key's value if the key is already available(Duplicate key), If key is not available then it will return null.

2) putIfAbsent(Object key, Object value) :- It will insert an entry if and only if key is not present , if the key is already available then it will not insert the Entry to the Map Collection

Methods of Map interface :

---

1) Object put(Object key, Object value) :- To insert one entry in the Map collection. It will return the old object key's value if the key is already available(Duplicate key).

2) void putAll(Map m) :- Merging of two Map collection

3) int size() :- To count the number of Entries.

4) void clear() :- Used to clear the Map

5) boolean isEmpty() :- To verify Map is empty or not?

6) boolean containsKey(Object key) :- To Search a particular key

7) boolean containsValue(Object value) :- To Search a particular value

8) Object get(Object key) :- It will return  corresponding value of key, if the key is not present then it will return null.

9) Object getOrDefault(Object key, Object defaultValue) :- To avoid null value this method has been introduced, here we can pass some defaultValue to avoid the null value.

10) remove(Object key) :- One complete entry will be removed.

11) putIfAbsent(Object key, Object value) :- It will insert an entry if and only if key is not present , if the key is already available then it will not insert the Entry to the Map Collection

Collection views Methods :

---

public Set keySet() :- Will return only keys (Set of keys)

Collection values() :- Will return all values.

Set<Map.Entry> entrySet() :- It will return key and value pair in the form of Entry.

a) getKey()  b) getValue()  c) setValue()

---

*** How HashMap Works internally ?

---

a) While working with HashSet or HashMap every object must be compared because duplicate objects are not allowed.

b) Whenever we add any new key to verify whether key is unique or duplicate, HashMap internally uses hashCode(), == operator and equals method.

c) While adding the key object in the HashMap, first of all it will invoke the hashCode() method to retrieve the corresponding key hashcode value.

Example :- hm.put(key,value);

then internally key.hashCode();

d) If the newly added key and existing key hashCode value both are same (Hash collision), then only == operator is used for comparing those keys by using reference or memory address, if both keys references are same then existing key value will be replaced with new key value.

If the reference of both keys are different then equals(Object obj) method is invoked to compare those keys by using state(data). [content comparison]

If the equals(Object obj) method returns true (content wise both keys are same), this new key is duplicate then existing key value will be replaced.

If equals(Object obj) method returns false, this new key is unique, new entry (key-value) will be inserted.

Note :- equals(Object obj) method is invoked only when two keys are having same hashcode as well as their references are different.

e) Actually by calling hashcode method we are not comparing the objects, we are just storing the objects in a group so the currently adding key object will be compared with its HASHCODE GROUP objects, but not with all the keys which are available in the Map.

f) The main purpose of storing objects into the corresponding group to decrease the number of comparison so the efficiency of the program will increase.

g) To insert an entry in the HashMap, HashMap internally uses Hashtable data structure

h) Now, for storing same hashcode object into a single group, hash table data structure internally uses one more data structure called Bucket.

i) The Hash table data structure internally uses Node class array object.

j) The bucket data structure internally uses LinkedList data structure, It is a single linked list again implemented by Node class only.

*k) A bucket is group of entries of same hash code keys.

l) Performance wise LinkedList is not good to serach, so from java 8 onwards LinkedList is changed to Binary tree to decrease the number of comparison within the same bucket hashcode if the number of entries are greater than 8.

* equals() and hashCode() method contract :

---

Both the methods are working together to find out the duplicate objects in the Map.

*If equals() method invoked on two objects and it returns true then hashcode of both the objects must be same.

---

```
package com.ravi.hash_map_demo;

import java.util.HashMap;

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

```
		HashMap<String,Integer> hm1 = new HashMap<>();
		hm1.put("A",1);
		hm1.put(new String("A"), 2);

		System.out.println(hm1.size());

		System.out.println(hm1);



	}

}
```

---

```
package com.ravi.hash_map_demo;

import java.util.HashMap;

public class HashMapDemo
{
	public static void main(String[] args)
	{
		HashMap<Integer,String> hm1 = new HashMap<>();
		hm1.put(128, "A");
		hm1.put(128, "B");

		System.out.println(hm1.size());
```

```
                System.out.println(hm1);

        }

}
```

128 == 128 wil return false so, to verify whether key is unique
or duplicate we are depending upon equals(Object obj) method
of Integer class.

---

```
import java.util.*;
class Employee
{
        int eid;
        String ename;

        Employee(int eid, String ename)
        {
                this.eid = eid;
                this.ename = ename;
        }

   @Override
        public boolean equals(Object obj)  //obj = e2
        {
                if(obj instanceof Employee)
     {
```

```
                    Employee e2 = (Employee) obj; //downcasting


                    if(this.eid == e2.eid && this.ename.equals(e2.ename))
                    {
                        return true;
                    }
                    else
                    {
                        return false;
                    }
          }
              else
              {
                    System.out.println("Comparison is not possible");
                    return false;
              }
        }

        public String toString()
              {
                     return " "+eid+" "+ename;
              }
}
public class HashMapDemo8
{
        public static void main(String[] args)
        {
              Employee e1 = new Employee(101,"Aryan");
```

```
        Employee e2 = new Employee(102,"Pooja");
        Employee e3 = new Employee(101,"Aryan");
        Employee e4 = e2;

        HashMap<Employee,String> hm = new HashMap<>();
        hm.put(e1,"Ameerpet");
        hm.put(e21-02-20242,"S.R Nagar");
        hm.put(e3,"Begumpet");
        hm.put(e4,"Panjagutta");

        hm.forEach((k,v)-> System.out.println(k+" : "+v));
    }
}
```

Here e2 and e4 are pointing to same memory object so hashcode is same hence == opeartor will retun true so duplicate key, S R nagar will be replaced by panjagutta.

---

HashMap :- [Unsorted, Unordered, No Duplicate keys]

---

public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Serializable, Clonable

It is a predefined class available in java.util package under Map interface available from JDK 1.2.

It gives us unsorted and Unordered map. when we need a map and we don't care about the order while iterating the elements through it then we should use HashMap.

It inserts the element based on the hashCode of the Object key using hashing technique [hasing alogorithhm]

It does not accept duplicate keys but value may be duplicate.

It accepts only one null key(because duplicate keys are not allowed) but multiple null values are allowed.

HashMap is not synchronized.

Time complexcity of search, insert and delete will be O(1)

We should use HashMap to perform searching opeartion.

It contains 4 types of constructor

1) HashMap hm1 = new HashMap();

It will create the HashMap Object with default capacity is 16. The default load fator or Fill Ratio is 0.75 (75% of HashMap is filled up then new HashMap Object will be created having double capacity)

2) HashMap hm2 = new HashMap(int initialCapacity);

will create the HashMap object with specified capacity

3) HashMap hm3 = new HashMap(int initialCapacity, float loadFactor);

we can specify our own initialCapacity and loadFactor(by default load factor is 0.75%)

4)HashMap hm4 = new HashMap(Map m);

Interconversion of Map Collection

---

```
//Program that shows HashMap is unordered
import java.util.*;
public class HashMapDemo
{
    public static void main(String[] a)
    {
        Map<String,String> map = new HashMap<>();
                    map.put("Ravi", "12345");
                    map.put("Rahul", "12345");
                    map.put("Aswin", "5678");
                    map.put(null, "6390");
                    map.put("Ravi","1529");

                    System.out.println(map);

        System.out.println(map.get(null));
                    System.out.println(map.get("Virat"));

                    map.forEach((k,v)-> System.out.println("Key is :"+k+" value is "+v));

    }
}
```

---

```
//Program to search a particular key and value in the Map collection
import java.util.*;
public class HashMapDemo1
```

```
{
	public static void main(String args[])
	{
		HashMap<Integer,String> hm = new HashMap<>();
		hm.put(1, "JSE");
		hm.put(2, "JEE");
		hm.put(3, "JME");
		hm.put(4,"JavaFX");
		hm.put(5,null);
		hm.put(6,null);

		System.out.println("Initial map elements: " + hm);
		System.out.println("key 2 is present or not :"+hm.containsKey(2));

		System.out.println("JME is present or not :"+hm.containsValue("JME"));

		System.out.println("Size of Map : " + hm.size());
		hm.clear();
		System.out.println("Map elements after clear: " + hm);
	}
}
```

---

```
//Collection view methods [keySet(),  values(),  entrySet()]
import java.util.*;
public class HashMapDemo2
{
public static void main(String args[])
	{
```

```
                Map<Integer,String> map = new HashMap<>();
                map.put(1, "C");
                map.put(2, "C++");
                map.put(3, "Java");
                map.put(4, ".net");

                map.forEach((k,v)->System.out.println("Key :"+k+" Value :"+v) );

                System.out.println("Return Old Object value
:"+map.put(4,"Python"));

                for(Map.Entry m : map.entrySet())
                {
                        System.out.println(m.getKey()+" : "+m.getValue());
                }
        }
}
```

---

```
package com.ravi.static_method_reference;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

public class Main
{
```

```
	public static void main(String[] args)
	{
		Map<Integer,String> map = new HashMap<>();
		map.put(1, "C");
		map.put(2, "C++");
		map.put(3, "Java");
		map.put(4, ".net");



		Iterator<Entry<Integer, String>> itr = map.entrySet().iterator();


		while(itr.hasNext())
		{
			System.out.println(itr.next());
		}


	}


}
```

iterator() method is not available with Map interface becuse map interface with key and value pair, if we want to use iterator()

then we need to use entrySet() method to convert Map into Set

---

```
import java.util.*;
public class HashMapDemo3
{
public static void main(String args[])
```

```
 {
                HashMap<Integer,String> map = new HashMap<>(10,8.5f);
                map.put(1, "Java");
                map.put(2, "is");
                map.put(3, "best");
                map.remove(3);  //will remove the complete Entry
                String val=(String)map.get(3);
                System.out.println("Value for key 3 is: " + val);
                map.forEach((k,v)->System.out.println(k +" : "+v));
  }
}
```

---

```
//To merge two Map Collection (putAll)
import java.util.*;
public class HashMapDemo4
{
public static void main(String args[])
        {
                HashMap<Integer,String> newmap1 = new HashMap<>();

                HashMap<Integer,String> newmap2 = new HashMap<>();

                newmap1.put(1, "OCPJP");
                newmap1.put(2, "is");
                newmap1.put(3, "best");

                System.out.println("Values in newmap1: "+ newmap1);
```

```
                newmap2.put(4, "Exam");

                newmap2.putAll(newmap1);

                newmap2.forEach((k,v)->System.out.println(k+" : "+v));
  }
}
------------------------------------------------------------
import java.util.*;
public class HashMapDemo5
{
   public static void main(String[] argv)
   {
      Map<String,String> map = new HashMap<>(9, 0.85f);
      map.put("key", "value");
      map.put("key2", "value2");
      map.put("key3", "value3");
                map.put("key7","value7");

                 Set keys = map.keySet();//keySet return type is Set
                 System.out.println(keys ); //[]

      Collection val = map.values(); //values return type is collection
      System.out.println(val);

                map.forEach((k,v)-> System.out.println(k+" : "+v));
```

```
    }
}
```

---

```
//getOrDefault() method
import java.util.*;
public class  HashMapDemo6
{
        public static void main(String[] args)
        {
                Map<String, String> map = new HashMap<>();
                map.put("A", "1");
                map.put("B", "2");
                map.put("C", "3");
     //if the key is not present, it will return default value .It is used to avoid null
                String value = map.getOrDefault("D","Key is not available");
                System.out.println(value);
                System.out.println(map);
        }
}
```

---

```
//interconversion of two HashMap
import java.util.*;
public class HashMapDemo7
        {
        public static void main(String args[])
        {
                HashMap<Integer, String> hm1 = new HashMap<>();
```

```
                hm1.put(1, "Ravi");
                hm1.put(2, "Rahul");
                hm1.put(3, "Rajen");

                HashMap<Integer, String> hm2       = new HashMap<>(hm1);

                System.out.println("Mapping of HashMap hm1 are : "      + hm1);

                System.out.println("Mapping of HashMap hm2 are : " + hm2);
        }
}
```

---

LinkedHashMap :

---

public class LinkedHashMap<K,V> extends HashMap<K,V> implements Map<K,V>

It is a predefined class available in java.util package under Map interface available from 1.4.

It is the sub class of HashMap class.

It maintains insertion order. It contains a doubly linked with the elements or nodes so It will iterate more slowly in comparison to HashMap.

It uses Hashtable and LinkedList data structure.

If We want to fetch the elements in the same order as they were inserted then we should go with LinkedHashMap.

It accepts one null key and multiple null values.

It is not synchronized.

It has also 4 constructors same as HashMap

1) LinkedHashMap hm1 = new LinkedHashMap();

will create a LinkedHashMap with default capacity 16 and load factor 0.75

2) LinkedHashMap hm1 = new LinkedHashMap(iny initialCapacity);

3) LinkedHashMap hm1 = new LinkedHashMap(iny initialCapacity, float loadFactor);

4) LinkedHashMap hm1 = new LinkedHashMap(Map m);

---

```
import java.util.*;
public class LinkedHashMapDemo
{
	public static void main(String[] args)
	{
		LinkedHashMap<Integer,String> l = new LinkedHashMap<>();
		l.put(1,"abc");
		l.put(3,"xyz");
		l.put(2,"pqr");
		l.put(4,"def");
		l.put(null,"ghi");
```

```
                System.out.println(l);
        }
}
```

---

```
import java.util.LinkedHashMap;
import java.util.Map;

public class LinkedHashMapDemo1
{
    public static void main(String[] a)
    {
        Map<String,String> map = new LinkedHashMap<>();
        map.put("Ravi", "1234");
                map.put("Rahul", "1234");
                map.put("Aswin", "1456");
                map.put("Samir", "1239");

                map.forEach((k,v)->System.out.println(k+" : "+v));
    }
}
```

---

Hashtable :

---

public class Hashtable<K,V> extends Dictionary<K,V> implements Map<K,V>, Clonable, Serializable

It is predefined class available in java.util package under Map interface.

Like Vector, Hashtable is also form the birth of java so called legacy class.

It is the sub class of Dictionary class which is an abstract class.

*The major difference between HashMap and Hashtable is, HashMap methods are not synchronized where as Hastable methods are synchronized.

HashMap can accept one null key and multiple null values where as Hashtable does not contain anything as a null(key and value both). if we try to add null then JVM will throw an exception i.e NullPointerException.

The initial default capacity of Hashtable class is 11 where as loadFactor is 0.75.

It has also same constructor as we have in HashMap.(4 constructors)

1) Hashtable hs1 = new Hashtable();

It will create the Hashtable Object with default capacity as 11 as well as load factor is 0.75

2) Hashtable hs2 = new Hashtable(int initialCapacity);

will create the Hashtable object with specified capacity

3) Hashtable hs3 = new Hashtable(int initialCapacity, float loadFactor);

we can specify our own initialCapacity and loadFactor

4) Hashtable hs = new Hashtable(Map c);

Interconversion of Map Collection

```
---------------------------------------------------------------
import java.util.*;
public class HashtableDemo
        {
         public static void main(String args[])
                {
                 Hashtable<Integer,String> map=new Hashtable<>();
                 map.put(1, "Java");
                 map.put(2, "is");
                 map.put(3, "best");
                 map.put(4,"language");

                 //map.put(5,null);

                 System.out.println(map);

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

                  for(Map.Entry m : map.entrySet())
                        {
                                System.out.println(m.getKey()+" = "+m.getValue());
                        }
    }
}
---------------------------------------------------------------
import java.util.*;
public class HashtableDemo1
{
```

```
  public static void main(String args[])
          {
   Hashtable<Integer,String> map=new Hashtable<>();
    map.put(1,"Priyanka");
    map.put(2,"Ruby");
    map.put(3,"Vibha");
    map.put(4,"Kanchan");

          map.putIfAbsent(5,"Bina");
          map.putIfAbsent(24,"Pooja");
          map.putIfAbsent(26,"Ankita");

    map.putIfAbsent(1,"Sneha");
    System.out.println("Updated Map: "+map);
 }
}
```

---

WeakHashMap :-

---

public class WeakHashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>

It is a predefined class in java.util package under Map interface.It was introduced from JDK 1.2v onwards.

While working with HashMap, keys of HashMap are of strong reference type. This means the entry of map will not be deleted by the garbage collector even though the key is set to be null and still it is not eligible for Garbage Collector.

On the other hand while working with WeakHashMap, keys of WeakHashMap are of weak reference type. This means the entry of a map is deleted by the garbage collector if the key value is set to be null because it is of weak type.

So, HashMap dominates over Garbage Collector where as Garbage Collector dominates over WeakHashMap.

It contains 4 types of Constructor :

---

1) WeakHashMap wm1 = new WeakHashMap();

Creates an empty WeakHashMap object with default capacity is 16 and load fator 0.75

2) WeakHashMap wm2 = new WeakHashMap(int initialCapacity);

3) WeakHashMap wm3 = new WeakHashMap(int initialCapacity, float loadFactor);

Eg:- WeakHashMap wm = new WeakHashMap(10,0.9);

capacity - The capacity of this map is 10. Meaning, it can store 10 entries.

loadFactor - The load factor of this map is 0.9. This means whenever our hashtable is filled up by 90%, the entries are moved to a new hashtable of double the size of the original hashtable.

4) WeakHashMap wm4 = new WeakHashMap(Map m);

---

import java.util.*;

```
public class WeakHashMapDemo
{
	public static void main(String args[])  throws Exception
	{
	  WeakHashMap<Test,String> map = new WeakHashMap<>();

		Test  t  = new Test();
		map.put(t," Rahul ");

		System.out.println(map); //{Test Nit = Rahul}

		t = null;

		System.gc();   //Explicitly calling garbage collector

		Thread.sleep(5000);

		System.out.println(map); //{}
	}
}

class Test
{
	@Override
	public String toString()
	{
		return "Test Nit";
	}
```

```
	@Override
	public void finalize() //called automaticaly if an object is eligible 4 GC
	{
		System.out.println("finalize method is called");
	}
}
```

---

23-02-2024

---

IdentityHashMap :

---

public class IdentityHashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Clonable, Serializable.

It was introduced from JDK 1.4 onwards.

The IdentityHashMap uses == operator to compare keys.

As we know HashMap uses equals() and hashCode() method for comparing the keys based on the hashcode of the object it will serach the bucket location and insert the entry their only.

So We should use IdentityHashMap where we need to check the reference or memory address instead of logical equality.

HashMap uses hashCode of the "Object key" to find out the bucket loaction in Hashtable, on the other hand IdentityHashMap does not use hashCode() method actually It uses System.identityHashCode(Object o)

IdentityHashMap is more faster than HashMap in case of Comparison.

It has three constrcutors, It does not contain loadFactor specific constructor.

---

```
import java.util.*;
public class IdentityHashMapDemo
{
	public static void main(String[] args)
	{
		HashMap<String,Integer> hm = new HashMap<>();

		IdentityHashMap<String,Integer> ihm = new IdentityHashMap<>();

		hm.put("Ravi",23);
		hm.put(new String("Ravi"), 24);

		ihm.put("Ravi",23);
		ihm.put(new String("Ravi"), 27); //compares based on == operator

		System.out.println("HashMap size :"+hm.size());
		System.out.println(hm);
		System.out.println(".....................");
		System.out.println("IdentityHashMap size :"+ihm.size());
		System.out.println(ihm);

	}

}
```

---

SortedMap<K,V>

---

It is a predefined interface available in java.util package under Map interface.

We should use SortedMap interface when we want to insert the key element based on some sorting order i.e the default natural sorting order.

---

TreeMap<K,V>

---

public class TreeMap<K,V> extends AbstractMap<K,V> implements NavigableMap<K,V> , Clonable, Serializable

It is a predefined class avaialble in java.util package under Map interface.

It is a sorted map that means it will sort the elements by natural sorting order based on the key or by using Comparator interface as a constructor parameter.

It does not allow non comparable keys.

It does not accept null key but null value allowed.

TreeMap implements NavigableMap and NavigableMap extends SortedMap. SortedMap extends Map interface.

TreeMap contains 4 types of Constructors :

1) TreeMap tm1 = new TreeMap(); //creates an empty TreeMap

2) TreeMap tm2 = new TreeMap(Comparator cmp); //user defined soting logic

3) TreeMap tm3 = new TreeMap(Map m);

4) TreeMap tm4 = new TreeMap(SortedMap m);

---

```
import java.util.*;
public class TreeMapDemo
{
	public static void main(String[] args)
	{
		TreeMap t = new TreeMap();
		t.put(4,"Ravi");
		t.put(7,"Aswin");
		t.put(2,"Ananya");
		t.put(1,"Dinesh");
		t.put(9,"Ravi");
		t.put(3,"Ankita");
		t.put(5,null);

		System.out.println(t);
	}
}
```

---

```
import java.util.*;
public class TreeMapDemo1
{
    public static void main(String args[])
```

```
    {
        TreeMap map = new TreeMap();
        map.put("one","1");
        map.put("two",null);
        map.put("three","3");
                            map.put("four",4);

        displayMap(map);

          map.forEach((k, v) -> System.out.println("Key = " + k + ", Value = " + v));

    }
    static void displayMap(TreeMap map)
    {
        Collection c = map.entrySet();   //Set<Map.Entry>

        Iterator i = c.iterator();
        i.forEachRemaining(x -> System.out.println(x));
    }
}
-------------------------------------------------------------
//firstKey()  lastKey()  headMap()  tailMap()  subMap()     Map
// first()    last()    headSet()  tailSet()  subSet()    Set

import java.util.*;
public class TreeMapDemo2
{
   public static void main(String[] argv)
```

```
  {
     Map map = new TreeMap();
     map.put("key2", "value2");
     map.put("key3", "value3");
     map.put("key1", "value1");

     System.out.println(map); //{}

     SortedMap x = (SortedMap) map;
     System.out.println("First key is :"+x.firstKey());
     System.out.println("Last Key is :"+x.lastKey());
   }
}
```

---

```
package com.ravi.map;

import java.util.TreeMap;

record Employee(int id, String name, int age)
{

}


public class TreeMapComparator {

        public static void main(String[] args)
        {
```

```
		System.out.println("Sorting name -> Ascending Order");

		TreeMap<Employee,String> tm1 = new TreeMap<Employee,String>((e1, e2)-> e1.name().compareTo(e2.name()));

		tm1.put(new Employee(101, "Zaheer", 24),"Hyderabad");
		tm1.put(new Employee(201, "Aryan", 27),"Jamshedpur");
		tm1.put(new Employee(301, "Pooja", 26),"Mumbai");

		tm1.forEach((k,v)-> System.out.println(k+" : "+v));

		System.out.println("Sorting EID -> Ascending Order");
		TreeMap<Employee,String> tm2 = new TreeMap<Employee,String>((e1,e2)-> e1.id()-e2.id());

		tm2.put(new Employee(101, "Zaheer", 24),"Hyderabad");
		tm2.put(new Employee(201, "Aryan", 27),"Jamshedpur");
		tm2.put(new Employee(301, "Pooja", 26),"Mumbai");

		tm2.forEach((k,v)-> System.out.println(k+" : "+v));


	}

}
```

---

Methods of SortedMap interface :

---

Methods of SortedMap interface :

---

1) firstKey()  //first key

2) lastKey()  //last key

3) headMap(int keyRange) //less than the specified range

4) tailMap(int keyRange)  //equal or greater than the specified range

5) subMap(int startKeyRange, int endKeyRange) //the range of key where startKey will be inclusive and endKey will be exclusive.

return type of headMap(), tailMap() and subMap() would be SortedMap(I)

---

```
import java.util.*;
public class SortedMapMethodDemo
        {
 public static void main(String args[])
         {
                SortedMap<Integer,String> map=new TreeMap<>();
                 map.put(100,"Amit");
                 map.put(101,"Ravi");
                 map.put(102,"Vijay");
                 map.put(103,"Rahul");

                 System.out.println("First Key: "+map.firstKey());  //100
                  System.out.println("Last Key "+map.lastKey());   //103
                  System.out.println("headMap: "+map.headMap(102));  //100 101
```

```
                System.out.println("tailMap: "+map.tailMap(102));  //102 103
                System.out.println("subMap: "+map.subMap(100, 102)); //100 101

        }
}
```

---

Assignment

NavigableMap Method

---

Properties class:

---

Properties class is used to maintain the data in the key-value form. It takes both key and value as a string.

Properties class is a subclass of Hashtable. It provides the methods to store properties in a properties file and to get the properties from the properties file, we should load method of Properties class.

System.getProperties() returns the all system properties.

Here we need to create a file with with extension .properties

Example :

---

```
driver = oracle.jdbc.driver.OracleDriver
user = system
password = tiger
```

```
FileReader fr = new FileReader("db.properties");

Properties p = new Properties();
p.load(fr);
```

---

db.properties

---

```
driver = sun.jdbc.odbc.JdbcOdbcDriver
user = scott
password = tiger
```

PropertiesExample1.java

---

```
import java.util.*;
import java.io.*;
public class PropertiesExample1
{
public static void main(String[] args)throws Exception
 {
   //Reading the data of properties file
   FileReader reader=new FileReader("db.properties");

   Properties p=new Properties();
   p.load(reader);  //load the properties file

   System.out.println(p.getProperty("user"));
   System.out.println(p.getProperty("password"));
          System.out.println(p.getProperty("driver"));
```

```
 }
}
```

By using Properties class we can dynamically change the property which we are taking from properties file without re-compilation of .java file.

---

```
import java.util.*;
import java.io.*;
public class PropertiesExample2
{
public static void main(String[] args)throws Exception
{
        Properties p=System.getProperties();
        Set set=p.entrySet();

        Iterator itr=set.iterator();
        while(itr.hasNext())
  {
                Map.Entry entry=(Map.Entry)itr.next();
                System.out.println(entry.getKey()+" = "+entry.getValue());
        }

} }
```

System class has predefined static method getProperties() whose return type is Properties and it will provide all the System property.

---

24-02-2024

---

* Generics :

---

Why generic came into picture :

---

As we know our compiler is known for Strict type checking because java is a statically typed checked language.

The basic problem with collection is It can hold any kind of Object.

```
ArrayList al = new ArrayList();
al.add("Ravi");
al.add("Aswin");
al.add("Rahul");
al.add("Raj");
al.add("Samir");

for(int i =0; i<al.size(); i++)
{
  String s = (String) al.get(i);
  System.out.println(s);
}
```

By looking the above code it is clear that Collection stores everything in the form of Object so here even after adding String type only we need type casting as shown below.

---

```
import java.util.*;
class Test
{
```

```
        public static void main(String[] args)
        {
    ArrayList al = new ArrayList();
                al.add(12);
                al.add(15);
                al.add(18);
                al.add(22);
                al.add(24);

                for (int i=0; i<al.size(); i++)
                {
       Integer x =(Integer) al.get(i);
                        System.out.println(x);
                }


  }

}
```

Even after type casting there is no guarantee that the things which are coming from ArrayList Object is Integer only because we can add anything in the Collection as a result java.lang.ClassCastException as shown in the program below.

```
import java.util.*;
class Test2
{
        public static void main(String[] args)
        {
```

```
                ArrayList t = new ArrayList(); //raw type
                t.add("alpha");
                t.add("beta");
                for (int i = 0; i < t.size(); i++)
                {
                 String str =(String) t.get(i);
                 System.out.println(str);
                }

                 t.add(1234);
                 t.add(1256);
                  for (int i = 0; i < t.size(); ++i)
            {
                        String obj= (String)t.get(i); //we can't perform type casting here
                        System.out.println(obj);
                  }
          }
}
```

---

To avoid all the above said problem Generics came into picture from JDK 1.5 onwards

-> It deals with type safe Object so there is a guarantee of both the end i.e putting inside and getting out.

Example:-

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

Now here we have a guarantee that only String can be inserted as well as only String will come out from the Collection so we can perform String related operation.

Advantages :-

---

a) Type safe Object (No compilation warning)

b) Strict compile time checking (Type erasure)

c) No need of type casting

```
import java.util.*;
public class Test3
{
public static void main(String[] args)
{
                    ArrayList<String> al = new ArrayList<>();  //Generic type
                    al.add("Ravi");
                    al.add("Ajay");
                    al.add("Vijay");

     for(int i=0; i<al.size(); i++)
                    {
                    String name = al.get(i); //no type casting is required
                    System.out.println(name.toUpperCase());
                    }
  }
}
```

---

//Program that describes the return type of any method can be type safe

//[We can apply generics on method return type]

```
import java.util.*;
public class Test4
{
	public static void main(String [] args)
	{
		Dog d1 = new Dog();
		Dog d2 = d1.getDogList().get(0);
		System.out.println(d2);
	}
}
class Dog
{
	public List<Dog> getDogList()
	{
		ArrayList<Dog> d = new ArrayList<>();
      d.add(new Dog());
		d.add(new Dog());
		d.add(new Dog());
		return d;
	}
}
```

Note :- In the above program the compiler will stop us from returning anything which is not compaitable List<Dog> and there is a guarantee that only "type safe list of Dog object" will be returned so we need not to provide type casting as shown below

```
Dog d2 = (Dog) d1.getDogList().get(0);  //before generic.
```

---

```
//Mixing generic with non-generic
import java.util.*;

class Car
{
}
public class Test5
{
	public static void main(String [] args)
	{
	ArrayList<Car> a = new ArrayList<>();
	a.add(new Car());
	a.add(new Car());
   a.add(new Car());

	ArrayList b = a;  //assigning Generic to raw type

   System.out.println(b);
	}
}
```

---

```
//Mixing generic to non-generic
import java.util.*;
public class Test6
{
	public static void main(String[] args)
	{
		List<Integer> myList = new ArrayList<>();
```

```
                myList.add(4);
                myList.add(6);
                myList.add(5);

                UnknownClass u = new UnknownClass();
                int total = u.addValues(myList);
                System.out.println("The sum of Integer Object is :"+total);
        }
}
class UnknownClass
{
        public int addValues(List list)  //generic to raw type OR
        {
        Iterator it = list.iterator();
        int total = 0;
        while (it.hasNext())
        {
                int i = ((Integer)it.next());
                total += i;                     //total = 15
        }
        return total;
        }
}
```

Note :-

In the above program the compiler will not generate any warning message because even though we are assigning type safe Integer Object to unsafe or raw type List Object but this List Object is not inserting anything new in the collection so there is no risk to the caller.

```
---------------------------------------------------------------
//Mixing generic to non-generic
import java.util.*;
public class Test7
{
	public static void main(String[] args)
	{
		List<Integer> myList = new ArrayList<>();

		myList.add(4);
		myList.add(6);
		UnknownClass u = new UnknownClass();
		int total = u.addValues(myList);
		System.out.println(total);
	}
}
class UnknownClass
{
   public int addValues(List list)
	{
		list.add(5);	//adding object to raw type
		Iterator it = list.iterator();
		int total = 0;
		while (it.hasNext())
		{
		int i = ((Integer)it.next());
		total += i;
		}
```

```
            return total;
        }
}
```

Here Compiler will generate warning message because the unsafe object is inserting the value 5 to safe object.

---

*Type Erasure

---

In the above program the compiler will generate warning message because the unsafe List Object is inserting the Integer object 5 so the type safe Integer object is getting value 5 from unsafe type so there is a problem to the caller method.

By writing ArrayList<Integer> actually JVM does not have any idea that our ArrayList was suppose to hold only Integers.

All the type safe generics information does not exist at runtime. All our generic code is Strictly for compiler.

There is a process done by java compiler called "Type erasure" in which the java compiler converts generic version to non-generic type.

List<Integer> myList = new ArrayList<Integer>();

At the compilation time it is fine but at runtime for JVM the code becomes

List myList = new ArrayList();

Note :- GENERIC IS STRICTLY A COMPILE TIME PROTECTION.

---

Behavior of Polymorphism with Array and Generics :

---

//Polymorphism with array

```
import java.util.*;
abstract class Animal
{
	public abstract void checkup();
}

class Dog extends Animal
{
	@Override
	public void checkup()
	{
		System.out.println("Dog checkup");
	}
}

class Cat extends Animal
{
	@Override
	public void checkup()
	{
		System.out.println("Cat checkup");
	}
}
```

```
class Bird extends Animal
{
	@Override
	public void checkup()
	{
		System.out.println("Bird checkup");
	}
}

public class  Test8
{
	public void checkAnimals(Animal animals[])
	{
		for(Animal animal : animals)
		{
			animal.checkup();
		}
	}

	public static void main(String[] args)
	{
		Dog []dogs={new Dog(), new Dog()};

		Cat []cats={new Cat(), new Cat(), new Cat()};

		Bird []birds = {new Bird(), new Bird()};

		Test8 t = new Test8();
```

```
                t.checkAnimals(dogs);
                t.checkAnimals(cats);
                t.checkAnimals(birds);
        }
}
```

Note :-From the above program it is clear that polymorphism(Upcasting) concept works with array.

---

```
import java.util.*;
abstract class Animal
{
        public abstract void checkup();
}

class Dog extends Animal
{
   @Override
        public void checkup()
        {
                System.out.println("Dog checkup");
        }
}

class Cat extends Animal
```

```
{
	@Override
	public void checkup()
	{
		System.out.println("Cat checkup");
	}
}
class Bird extends Animal
{
	@Override
	public void checkup()
	{
		System.out.println("Bird checkup");
	}
}
public class Test9
{
	public void checkAnimals(List<Animal> animals)
	{
		for(Animal animal : animals)
		{
	animal.checkup();
		}
	}
	public static void main(String[] args)
	{
		List<Dog> dogs = new ArrayList<>();
		dogs.add(new Dog());
```

```
        dogs.add(new Dog());

        List<Cat> cats = new ArrayList<>();
        cats.add(new Cat());
        cats.add(new Cat());

        List<Bird> birds = new ArrayList<>();
        birds.add(new Bird());

        Test9 t = new Test9();
        t.checkAnimals(dogs);
        t.checkAnimals(cats);
        t.checkAnimals(birds);

    }
}
```

Note :- The above program will generate the compilation error.

So from the above program it is clear that polymorphism does not work in the same way for generics as it does with arrays.

Eg:-

```
Parent [] arr = new Child[5]; //valid
Object [] arr = new String[5]; //valid
```

But in generics the same type is not valid

```
List<Object> list = new ArrayList<Integer>(); //Invalid

List<Parent> mylist = new ArrayList<Child>(); //Invalid
```

---

```
import java.util.*;

public class Test10

{

public static void main(String [] args)

        {

          //ArrayList<Number> al = new ArrayList<Integer>(); [Compile time]

          //ArrayList al = new ArrayList();  [Runtime]

          //al.add("Ravi");



                Object []obj = new String[3]; //valid with Array

                obj[0] = "Ravi";

                obj[1] = "hyd";

                obj[2] = 12; //java.lang.ArrayStoreException

        }

}
```

Note :- It will generate java.lang.ArrayStoreException because we are trying to insert 12 (integer value) into String array.

In Array we have an Exception called ArrayStoreException but the same Exception or such type of exception, is not available with Generics that is the reason in generics compiler does not allow upcasting concept.

(It is a strict compile time checking)

---

```
import java.util.*;
class Parent
{
}
class Child extends Parent
{
}

public class Test11
{
public static void main(String [] args)
	{
		ArrayList<Parent> lp = new ArrayList<Child>();//error

		ArrayList<Parent> lp1 = new ArrayList<Parent>();

		ArrayList<Child> lp2 = new ArrayList<>();

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

---

---

Wild card character(?) :

---

<?>  -: Many possibilities

<Animal> -: Only <Animal> can assign, but not Dog or sub type of animal

<? super Dog> -: Dog, Animal, Object can assign (Compiler has surity)

<? extends Animal> -: Below of Animal(Child of Animal) means, sub classes of Animal (But the compiler does not have surity because you can have many sub classes of Animal in the future, so chances of wrong collections)

---

```
//program on wild-card chracter
import java.util.*;
class Parent
{

}
class Child extends Parent
{
}
public class Test12
{
public static void main(String [] args)
	{
		List<?> lp = new ArrayList<Parent>();
		System.out.println("Wild card....");
	}
}
```

---

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

```
public class Test13
{
	public static void main(String[] args)
	{
		List<? extends Number> list1 = new ArrayList<Integer>();

		List<? super Integer> list2 = new ArrayList<Object>();

		List<? super Gamma> list3 = new ArrayList<Alpha>();

		List list4 = new ArrayList();

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

class Alpha
{
}
class Beta extends Alpha
{
}
class Gamma extends Alpha
{
}
```

---

```
//By using generic we can accept hetrogeneous elements
import java.util.*;
```

```
public class  Test14
{
	public static void main(String[] args)
	{
		try
		{
		   List<Object> x = new ArrayList<>(); //Array of Object[java 9]
	       x.add(10);
			x.add("Ravi");
			x.add(true);
			x.add(34.89);
			System.out.println(x);
		}
		catch (Exception e)
		{
			System.out.println(e);
		}
	}
}
------------------------------------------------------------
class MyClass<T>
{
	T obj;
	public MyClass(T obj)	//Student obj
	{
		this.obj=obj;
	}
```

```
	T getObj()
	{
		return obj;
	}
}
public class Test15
{
	public static void main(String[] args)
	{
		Integer i=12;
		MyClass<Integer> mi = new MyClass<>(i);
		System.out.println("Integer object stored :"+mi.getObj());

		Float f=12.34f;
		MyClass<Float> mf = new MyClass<>(f);
		System.out.println("Float object stored :"+mf.getObj());

		MyClass<String> ms = new MyClass<>("Rahul");
		System.out.println("String object stored :"+ms.getObj());

		MyClass<Boolean> mb = new MyClass<>(false);
		System.out.println("Boolean object stored :"+mb.getObj());

		Double d=99.34;
		MyClass<Double> md = new MyClass<>(d);
		System.out.println("Double object stored :"+md.getObj());

		MyClass<Student> std = new MyClass<>(new Student(1,"Scott"));
```

```
                System.out.println("Student object stored :"+std.getObj());
        }
}

record Student(int id, String name)
{

}
------------------------------------------------------------
//E stands for Element type
class Fruit
{
}
class Apple extends Fruit   //Fruit is super, Apple is sub class
{
        @Override
        public String toString()
        {
                return "Apple";
        }
}

class Basket<E>   //E type is Fruit type
{
        private E element;
        public void setElement(E element) //Fruit element
        {
                this.element = element;
```

```
	}

	public E getElement() //public Fruit getElement(){}
	{
		return this.element;
	}
}

public class Test16
{
	public static void main(String[] args)
	{
		Basket<Fruit> b = new Basket<>();
		b.setElement(new Apple());

		Apple x = (Apple) b.getElement();
		System.out.println(x);


    Basket<Fruit> b1 = new Basket<>();
		b1.setElement(new Mango());
		Mango y = (Mango)b1.getElement();
		System.out.println(y);

	}
}
class Mango extends Fruit
{
```

```
	@Override
	public String toString()
	{
		return "Mango";
	}
}
```

---

Concurrent collections in java

---

Concurrent Collections are introduced from JDK 1.5 onwards to enhance the performance of multithreaded application.

These are threadsafe collection and available in java.util.concurrent sub package.

Limitation of Traditional Collection :

---

1) In the Collection framework most of the Collection classes are not thread-safe because those are non-synchronized like ArrayList, LinkedList, HashSet, HashMap is non-synchronized in nature, So If multiple threads will perform any operation on the collection object simultaneously then we will get some wrong data this is known as Data race or Race condition.

2) Some Collection classes are synchronized like Vector, Hashtable but performance wise these classes are slow in nature.

Collections class has provided static methods to make our List, Set and Map interface classes as a synchronized.

a) public static List synchronizedList(List list)

b) public static Set synchronizedSet(Set set)

c) public static Map synchronizedMap(Map map)

3) Traditional Collection works with fail fast iterator that means while iterating the element,

if there is a change in structure then we will get ConcurrentModificationException.

On the other hand concurrent collection works with fail safe iterator where even though there is a change in structure but we will not get ConcurrentModificationException.

---

```
import java.util.*;
public class Collection1
{
   public static void main(String args[])
   {
      ArrayList al = new ArrayList();
      al.add(10);
      al.add(20);
      al.add(30);
      al.add(40);
      al.add(50);
                  al.add(50);
      System.out.println("Arraylist Elements : "+al);
      Set s = new HashSet(al);
      System.out.println("Set Elements are: "+s);
   }
}
```

---

```
//Collections.synchronizedList(List list);
import java.util.*;
```

```
public class Collection2
{
	public static void main(String[] args)
	{
		ArrayList<String> arl = new ArrayList<>();
		arl.add("Apple");
		arl.add("Orange");
		arl.add("Grapes");
		arl.add("Mango");
		arl.add("Guava");
		arl.add("Mango");

		List<String> syncCollection = Collections.synchronizedList(arl);

		List<String> upperList = new ArrayList<>(); //New List

		Runnable listOperations = () ->
		{
			synchronized (syncCollection)
			{
	syncCollection.forEach(str -> upperList.add(str.toUpperCase()));
	  }
	};

	  Thread t1 = new Thread(listOperations);
	  t1.start();

    upperList.forEach(x -> System.out.println(x));
```

```
        }
}
```

---

```
//Collections.synchronizedSet(Set set);
import java.util.*;
public class Collection3
{
   public static void main(String[] args)
                    {
      Set<String> set = Collections.synchronizedSet(new HashSet<>());
      set.add("Apple");
                    set.add("Orange");
                    set.add("Grapes");
                    set.add("Mango");
                    set.add("Guava");
                    set.add("Mango");
      System.out.println("Set after Synchronization :");
      synchronized (set)
                    {
        Spliterator<String> itr = set.spliterator();
                      itr.forEachRemaining(str -> System.out.println(str));
      }
   }
}
```

---

```
//Collections.synchronizedMap(Map map);
import java.util.*;
public class Collection4
```

```
{
   public static void main(String[] args)
        {
      Map<String, String> map = new HashMap<String, String>();
      map.put("1", "Ravi");
      map.put("4", "Elina");
      map.put("3", "Aryan");
      Map<String, String> synmap = Collections.synchronizedMap(map);
      System.out.println("Synchronized map is :" + synmap);
   }
}
```

---

```
import java.util.*;
class ConcurrentModification extends Thread
{
        ArrayList<String> al = null;
        public ConcurrentModification(ArrayList<String> al) //al = arl
        {
                this.al = al;
        }

        @Override
        public void run()
        {
                try
                {
                        Thread.sleep(1000);
                }
```

```
                catch (InterruptedException e)
                {
                }
                al.add("KIWI");
        }
}
public class Collection5
{
        public static void main(String[] args) throws InterruptedException
        {
                ArrayList<String> arl = new ArrayList<>();
                arl.add("Apple");
                arl.add("Orange");
                arl.add("Grapes");
                arl.add("Mango");
                arl.add("Guava");

                ConcurrentModification cm = new ConcurrentModification(arl);

                cm.start();

                Iterator<String> itr = arl.iterator();
    while(itr.hasNext())
                {
                        String str = itr.next();
                        System.out.println(str);
                        Thread.sleep(1500);
                }
```

```
        }
}
```

Note :- The above program will throw an exception i.e ConcurrentModificationException because ArrayList class is non-synchronized so, it cannot handle multiple threads

---

CopyOnWriteArrayList in java :

---

public class CopyOnWriteArrayList implements List, Cloneable, Serializable, RanomAccess

A CopyOnWriteArrayList is similar to an ArrayList but it has some additional features like thread-safe. This class is existing in java.util.concurrent sub package.

ArrayList is not thread-safe. We can't use ArrayList in the multi-threaded environment because it creates a problem in ArrayList values (Data inconsistency).

*The CopyOnWriteArrayList is an enhanced version of ArrayList. If we are making any modifications(add, remove, etc.) in CopyOnWriteArrayList then JVM creates a new copy by use of Cloning.

The CopyOnWriteArrayList is costly, if we want to perform update operations, because whenever we make any changes the JVM creates a cloned copy of the array and add/update element to it.

It is a thread-safe version of ArrayList as well as here Iterator is fail safe iterator. Multiple threads can read the data but only one thread can write the data at one time.

*CopyOnWriteArrayList is the best choice if we want to perform read operation frequently in multithreaded environment.

The CopyOnWriteArrayList is a replacement of a synchronized List, because it offers better concurrency.

Constructors of CopyOnWriteArrayList in java :

---

We have 3 constructors :

1) CopyOnWriteArrayList c = new CopyOnWriteArrayList();

It creates an empty list in memory. This constructor is useful when we want to create a list without any value.

2) CopyOnWriteArrayList c = new CopyOnWriteArrayList(Collection c);

Interconversion of collections.

3) CopyOnWriteArrayList c = new CopyOnWriteArrayList(Object[] obj) ;

It Creates a list that containing all the elements that is specified Array. This constructor is useful when we want to create a CopyOnWriteArrayList from Array.

---

```
import java.util.Arrays;

import java.util.Iterator;

import java.util.List;

import java.util.concurrent.CopyOnWriteArrayList;


public class CopyOnWriteArrayListExample1

{

   public static void main(String[] args)

   {

      List<String> list = Arrays.asList("Apple", "Orange", "Mango","Kiwi", "Grapes");
```

```
    CopyOnWriteArrayList<String> copyOnWriteList = new
CopyOnWriteArrayList<String>(list);

    System.out.println("Without modification = "+copyOnWriteList);

                //Iterator1
    Iterator<String> iterator1 = copyOnWriteList.iterator();

    //Add one element and verify list is updated
    copyOnWriteList.add("Guava");

    System.out.println("After modification = "+copyOnWriteList);

    //Iterator2
    Iterator<String> iterator2 = copyOnWriteList.iterator();

    System.out.println("Element from first Iterator:");
    iterator1.forEachRemaining(System.out :: println);

    System.out.println("Element from Second Iterator:");
    iterator2.forEachRemaining(System.out :: println);
  }
}
```

---

```
import java.util.*;
import java.util.concurrent.*;
class ConcurrentModification extends Thread
```

```
{
        CopyOnWriteArrayList<String> al = null;
        public ConcurrentModification(CopyOnWriteArrayList<String> al)
        {
                this.al = al;
        }
        @Override
        public void run()
        {
                try
                {
                        Thread.sleep(1000);
                }
                catch (InterruptedException e)
                {
                }
           al.add("KIWI");
        }
}
public class CopyOnWriteArrayListExample2
{
        public static void main(String[] args) throws InterruptedException
        {
                CopyOnWriteArrayList<String> arl = new CopyOnWriteArrayList<>();
                arl.add("Apple");
                arl.add("Orange");
                arl.add("Grapes");
                arl.add("Mango");
```

```
                arl.add("Guava");
                ConcurrentModification cm = new ConcurrentModification(arl);
                cm.start();

                Iterator<String> itr = arl.iterator();
    while(itr.hasNext())
                {
                        String str = itr.next();
                        System.out.println(str);
                  Thread.sleep(1500);
                }

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

                Spliterator<String> spl =  arl.spliterator();
                spl.forEachRemaining(x -> System.out.println(x));
        }
}
```

---

CopyOnWriteArraySet :

---

public class CopyOnWriteArraySet extends AbstractSet implements Serializable

A CopyOnWriteArraySet is a thread-safe version of HashSet in Java and it works like CopyOnWriteArrayList in java.

The CopyOnWriteArraySet internally used CopyOnWriteArrayList to perform all type of operation.It means the CopyOnWriteArraySet internally creates an object of CopyOnWriteArrayList and perform operation on it.

Whenever we perform add, set, and remove operation on CopyOnWriteArraySet, it internally creates a new object of CopyOnWriteArrayList and copies all the data to the new object. So, when it is used in by multiple threads, it doesn't create a problem, but it is well suited if we have small size collection and want to perform only read operation by multiple threads.

The CopyOnWriteArraySet is the replacement of synchronizedSet and offers better concurrency.

It creates a new copy of the array every time iterator is created, so performance is slower than HashSet.

Constructors :

---

It has two constructors

1) CopyOnWriteArraySet set1 = new CopyOnWriteArraySet();

It will create an empty Set

2) CopyOnWriteArraySet set1 = new CopyOnWriteArraySet(Collection c);
Interconversion of collection

```
import java.util.ArrayList;
import java.util.concurrent.CopyOnWriteArraySet;

public class CopyOnWriteArraySetExample1
```

```
{
   public static void main(String[] args)
   {
      ArrayList<String> list = new ArrayList<String>();
      list.add("Apple");
      list.add("Orange");
      list.add("Grapes");
                  list.add("Grapes");

      CopyOnWriteArraySet<String> set = new CopyOnWriteArraySet<String>(list);
      System.out.println("Element from Set: "+ set);
   }
}
```

---

```
import java.util.concurrent.CopyOnWriteArraySet;

public class CopyOnWriteArraySetExample2
{
   public static void main(String[] args)
   {
      CopyOnWriteArraySet<Integer> set = new CopyOnWriteArraySet<Integer>();
      set.add(1);
      set.add(2);
      set.add(3);
      set.add(4);
      set.add(5);

      System.out.println("Is element contains: "+set.contains(1));
```

```
        System.out.println("Is set empty: "+set.isEmpty());

        System.out.println("remove element from set: "+set.remove(3));

        System.out.println("Element from Set: "+ set);
    }
}
```

---

*** ConcurrentHashMap :

---

public class ConcurrentHashMap<K,V> extends AbstractMap<K,V> implements ConcurrentMap<K,V>, Serializable

Like HashMap, ConcurrentHashMap provides similar functionality except that it has internally maintained concurrency.

It is the concurrent version of the HashMap. It internally maintains a Hashtable that is divided into segments(Buckets).

The number of segments depends upon the level of concurrency required the Concurrent HashMap. By default, it divides into 16 segments and each Segment behaves independently. It doesn't lock the whole HashMap as done in Hashtables/synchronizedMap, it only locks the particular segment(Bucket) of HashMap. [Bucket level locking]

ConcurrentHashMap allows multiple threads can perform read operation without locking the ConcurrentHashMap object.

It does not allow null as a key or evan null as a value.

[Note :- TreeSet, TreeMap, Hashtable, PriroityQueue, ConcurrentHashMap]

It contains 5 types of constructor :

---

1) ConcurrentHashMap chm1 = new ConcurrentHashMap();

2) ConcurrentHashMap chm2 = new ConcurrentHashMap(int initialCapacity);

3) ConcurrentHashMap chm3 = new ConcurrentHashMap(int initialCapacity, float loadFactor);

4) ConcurrentHashMap chm4 = new ConcurrentHashMap(int initialCapacity, float loadFactor, int concurrencyLevel);

5) ConcurrentHashMap chm5 = new ConcurrentHashMap(ConcurrentMap m);

Internal Working of ConcurrentHashMap :

---

Like HashMap and Hashtable, the ConcurrentHashMap is also used Hashtable data structure. But it is using the segment locking strategy to handle the multiple threads.

A segment(bucket) is a portion of ConcurrentHashMap and ConcurrentHashMap uses a separate lock for each thread. Unlike Hashtable or synchronized HashMap, it doesn't synchronize the whole HashMap or Hashtable for one thread.

As we have seen in the internal implementation of the HashMap, the default size of HashMap is 16 and it means there are 16 buckets. The ConcurrentHashMap uses the same concept is used in ConcurrentHashMap. It uses the 16 separate locks for 16 buckets by default because the default concurrency level is 16. It means a

ConcurrentHashMap can be used by 16 threads at same time. If one thread is reading from one bucket(Segment), then the second bucket doesn't affect it.

Why we need ConcurrentHashMap in java?

---

As we know Hashtable and HashMap works based on key-value pairs. But why we are introducing another Map? As we know HashMap is not thread safe, but we can make it thread-safe by using Collections.synchronizedMap() method and Hashtable is thread-safe by default.

But a synchronized HashMap or Hashtable is accessible only by one thread at a time because the object get the lock for the whole HashMap or Hashtable. Even multiple threads can't perform read operations at the same time. It is the main disadvantage of Synchronized HashMap or Hashtable, which creates performance issues. So ConcurrentHashMap provides better performance than Synchronized HashMap or Hashtable.

---

```
import java.util.HashMap;
import java.util.concurrent.ConcurrentHashMap;

public class ConcurrentHashMapExample1
{
  public static void main(String args[])
  {

    HashMap<Integer, String> hashMap = new HashMap<Integer, String>();
    hashMap.put(1, "Ravi");
    hashMap.put(2, "Ankit");
    hashMap.put(3, "Prashant");
        hashMap.put(4, "Pallavi");
```

```
    ConcurrentHashMap<Integer, String> concurrentHashMap = new
ConcurrentHashMap<Integer, String>(hashMap);
    System.out.println("Object from ConcurrentHashMap: "+ concurrentHashMap);


  }


}
```

---

```
import java.util.Iterator;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;


public class ConcurrentHashMapExample2
{
  public static void main(String args[])
  {
    // Creating ConcurrentHashMap
    Map<String, String> cityTemperatureMap = new ConcurrentHashMap<String,
String>();


     cityTemperatureMap.put("Delhi", "30");
     cityTemperatureMap.put("Mumbai", "32");
     cityTemperatureMap.put("Chennai", "35");
     cityTemperatureMap.put("Bangalore", "22" );


     Iterator<String> iterator = cityTemperatureMap.keySet().iterator();


     while (iterator.hasNext())
     {
```

```
        System.out.println(cityTemperatureMap.get(iterator.next()));
        // adding new value, it won't throw error
        cityTemperatureMap.put("Hyderabad", "28");
      }
  }
}
```

---

Topics we need to cover :

---

1) Stream API (Intermediate Opeartion & Terminal Opeartion)

2) Queue interface (PriorityQueue)

3) Sealed class

4) New Date and Time API

5) Optional class

6) Java IO and File Handling

7) Inner classes in java

8) Array and String (Online)

Streams in java :

---

It is introduced from Java 8 onwards, the Stream API is used to process the collection objects.

It contains classes for processing sequence of elements over Collection object and array.

Stream is a predefined interface available in java.util.stream sub package.

Package Information :

---

java.util -> Base package

java.util.function -> Functional interfaces

java.util.concurrent -> Multithreaded support

java.util.stream -> Processing of Collection Object

forEach() method in java :

---

The Java forEach() method is a technique to iterate over a collection such as (list, set or map) and stream. It is used to perform a given action on each of the element of the collection.

The forEach() method has been added in following places:

Iterable interface – This makes Iterable.forEach() method available to all collection classes. Iterable interface is the super interface of Collection interface

Map interface – This makes forEach() operation available to all map classes.

Stream interface – This makes forEach() operations available to all types of stream.

---

Creation of Streams to process the data :

---

We can create Stream from collection or array with the help of stream() and of() methods:

A stream() method is added to the Collection interface and allows creating a Stream<T> using any collection object as a source

public java.util.Stream<E> stream();

The return type of this method is Stream interafce available in java.util.stream sub package.

Eg:-

```
List<String> items = new ArrayList<String>();
                items.add("Apple");
                items.add("Orange");
                items.add("Mango");
                Stream<String> stream = items.stream();
```

---

```
package com.ravi.basic;
import java.util.*;  //Base package
import java.util.stream.*; //Sub package
public class StreamDemo1
{
        public static void main(String[] args)
        {
                List<String> items = new ArrayList<>();

                items.add("Apple");
                items.add("Orange");
                items.add("Mango");

    //Collections Object to Stream
                Stream<String> strm = items.stream();
                strm.forEach(p -> System.out.println(p));
```

```
	}
}
```

---

Stream.of()

---

public static java.util.stream.Stream  of(<T>)

---

It is a static method of Stream interface through which we can create Stream of arrays and Collection. The return type of this method is Stream interface

---

```
//Stream.of()
package com.ravi.basic;
import java.util.stream.*;
public class StreamDemo2
{
	public static void main(String[] args)
	{
		Stream<Integer> stream = Stream.of(1,2,3,4,5,6,7,8,9);
		stream.forEach(p -> System.out.println(p));

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

	//Anonymous Array Object
		Stream<Integer> strm = Stream.of( new Integer[]{15,29,45,8,16} );
		strm.forEach(p -> System.out.println(p));
	}
}
```

---

Operation in Stream API :

---

In Java, the Stream API provides a wide range of operations that can be performed on a stream of elements.

These operations can be categorized into two main types: intermediate operations and terminal operations.

Intermediate Operations:

---

filter(Predicate<T> predicate): Returns a new stream which contains filtered elements based on the boolean expression using Predicate.

map(Function<T, R> mapper): Transforms elements in the stream using the provided mapping function.

flatMap(Function<T, Stream<R>> mapper): Flattens a stream of streams into a single stream.

distinct(): Returns a stream with distinct elements (based on their equals method).

sorted(): Returns a stream with elements sorted in their natural order.

sorted(Comparator<T> comparator): Returns a stream with elements sorted using the specified comparator.

peek(Consumer<T> action): Allows us to perform an action on each element in the stream without modifying the stream.

limit(long maxSize): Limits the number of elements in the stream to a specified maximum size.

skip(long n): Skips the first n elements in the stream.

takeWhile(Predicate<T> predicate): Returns a stream of elements from the beginning until the first element that does not satisfy the predicate.

dropWhile(Predicate<T> predicate): Returns a stream of elements after skipping elements at the beginning that satisfy the predicate.

---

public abstract Stream<T> filter(Predicate<T> p) :

---

It is a predfined method of Stream interface. It is used to select/filter elements as per the Predicate passed as an argument. It is basically used to filter the elements based on boolean condition.

public abstract <T>  collect(java.util.stream.Collectors c)

---

It is a predfined method of Stream interface. It is used to return the result of the intermediate operations performed on the stream. It is used to collect the data after filteration and convert the data to the Collection.

Collectors is a predfined final class available in java.util.stream sub package which conatins a static method toList() and toSet() to convert the data as a List/Set i.e Collection object. The return type of this method is List/Set interface.

---

package com.ravi.stream_demo;

import java.util.ArrayList;

import java.util.Arrays;

```
import java.util.List;

public class FilterDemo {

	public static void main(String[] args)
	{
		List<Integer> list = Arrays.asList(1,2,3,4,5,6,7,8,9,10);

	//Print all the even numbers for this list without Stream

		List<Integer> even = new ArrayList<>();

		for(Integer i : list)
		{
			if(i % 2==0)
				even.add(i);
		}
		System.out.println("All the even numbers without Stream");
		even.forEach(System.out::println);

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

		//Print all the even numbers for this list with Stream
		System.out.println("All the even numbers with Stream");
		list.stream().filter(num -> num%2==0).forEach(System.out::println);


	}
```

```
}
```

---

```
package com.ravi.stream_demo;


import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;


public class FilterDemo2
{
	public static void main(String[] args)
	{
		List<String> listOfName = 
Arrays.asList("Ravi","Rahul","Ankit","Samir","Raj","Ravi");


		//fetching the name starts with 'R'
		 List<String> collect = listOfName.stream().filter(str -> 
str.startsWith("R")).collect(Collectors.toList());
		 System.out.println(collect);



	}


}
```

---

```
package com.ravi.stream_demo;


import java.util.ArrayList;
```

```
record Employee(int id, String name, double salary)
{

}



public class FilterDemo3
{
	public static void main(String[] args)
	{
		ArrayList<Employee> al = new ArrayList<>();
		al.add(new Employee(1, "Scott", 30000));
		al.add(new Employee(2, "Smith", 40000));
		al.add(new Employee(3, "Virat", 20000));
		al.add(new Employee(4, "Rohit", 50000));

		al.stream().filter(emp -> emp.salary() > 20000).forEach(e->
System.out.println(e.name()));

	}

}
```

---

public Stream map(Function<T,R> mapper) :

---

It is a predefined method of Stream interface.

It takes Function (Predefined functional interafce ) as a parameter.

It performs intermediate operation and consumes single element from input Stream and produces single element to output Stream.

Here mapper function is functional interface which takes one input and provides one output.

```
//Programs
package com.ravi.basic;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class StreamDemo7
{
   public static void main(String[] args)
   {
      List<Integer> listOfNumbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
      listOfNumbers.stream().map(num -> num + 10).forEach(System.out::println);

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

      List<Integer> numbers = Arrays.asList(1,2,3,4,5,6,7,8,9,10);

      List<Integer> cubeOfNum = numbers.stream().filter(n -> n%2==0).map(x -> x*x*x).collect(Collectors.toList());
      System.out.println(cubeOfNum);
```

```
        System.out.println(".....");
        List<String> listOfString = Arrays.asList("Raj","Rahul","Chandrashekhar",
"ArrayIndexOutOfBoundsException");

        listOfString.stream().map(str -> str.length()).forEach(System.out::println);



    }
}
```

---

```
//Program on map(Function<T,R> mapped)
package com.ravi.basic;
import java.util.*;
import java.util.stream.*;
public class StreamDemo8
{
        public static void main(String args[])
        {
                List<Player> listOfPlayers = createMyPlayerList();

                //List<Player> to Set<String>
                Set<String> setOfNames = listOfPlayers.stream().map(p ->
p.name()).collect(Collectors.toSet());
                System.out.println(setOfNames);
        }

        public static List<Player> createMyPlayerList()
        {
                List<Player> listOfPlayers = new ArrayList<>();
```

```
                listOfPlayers.add(new Player("Virat",32));
                listOfPlayers.add(new Player("Rohit",33));
                listOfPlayers.add(new Player("Shami",34));
                listOfPlayers.add(new Player("Siraj",28));
                listOfPlayers.add(new Player("Sarfaraj",26));
                listOfPlayers.add(new Player("Virat",32));

                return listOfPlayers;
        }
}

record Player(String name, int age)
{

}
```

---

public Stream flatMap(Function<? super T,? extends Stream<? extends R>> mapper)

---

It is a predefined method of Stream interface.

The map() method produces one output value for each input value in the stream. So if there are n elements in the stream, map() operation will produce a stream of n output elements.

flatMap() is two step process i.e. map() + Flattening. It helps in converting Collection<Collection<T>> to Collection<T> [to make flat i.e converting Collections of collection into single collection or merging of all the collection]

---

```
//flatMap()
```

```
//map + Flattening [Converting Collections of collection into single collection]
package com.ravi.basic;
import java.util.*;
import java.util.stream.*;
public class StreamDemo9
{
	public static void main(String[] args)
	{
		List<String> list1 = Arrays.asList("A","B","C");
		List<String> list2 = Arrays.asList("D","E","F");
		List<String> list3 = Arrays.asList("G","H","I");

		List<List<String>> listOfLists = Arrays.asList(list1, list2, list3);

			List<String> collect = listOfLists.stream().flatMap(list->
list.stream()).collect(Collectors.toList());
			System.out.println(collect);

	}
}
```

---

```
//Flattening of prime, even and odd number
package com.ravi.basic.flat_map;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;
```

```
public class FlatMapDemo1
{
	public static void main(String[] args)
	{
    List<Integer> primeNumbers = Arrays.asList(5,7,11);
    List<Integer> evenNumbers = Arrays.asList(2,4,6);
    List<Integer> oddNumbers = Arrays.asList(1,3,5);

    List<List<Integer>> listOfCollection = Arrays.asList(primeNumbers, evenNumbers, oddNumbers);

    List<Integer> collect = listOfCollection.stream().flatMap(num -> num.stream()).collect(Collectors.toList());

  System.out.println(collect);

	}
}
```

---

//Fetching first character using flatMap()

```
package com.ravi.basic.flat_map;

import java.util.Arrays;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;
import java.util.stream.Stream;
```

```
public class FlatMapDemo2
{
	public static void main(String[] args)
	{
		List<String> asList = Arrays.asList("Jyoti","Ankit","Vaibhab","Aman");

		List<Character> collect = asList.stream().flatMap(str ->
Stream.of(str.charAt(0))).collect(Collectors.toList());
		System.out.println(collect);



	}

}
```

---

```
package com.ravi.basic.flat_map;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

class Product
{
	private Integer productId;
	private List<String> listOfProducts;

	public Product(Integer productId, List<String> listOfProducts) {
```

```
		super();
		this.productId = productId;
		this.listOfProducts = listOfProducts;
	}

	public Integer getProductId() {
		return productId;
	}

	public List<String> getListOfProducts() {
		return listOfProducts;
	}
}

public class FlatMapDemo3
{
	public static void main(String[] args)
	{

		List<Product> listOfProduct = Arrays.asList(
	   new Product(1, Arrays.asList("Camera", "Mobile","Laptop")),
		new Product(2, Arrays.asList("Bat", "Ball","Wicket")),
		new Product(3, Arrays.asList("Chair", "Table","Lamp")),
		new Product(4, Arrays.asList("Cycle", "Bike","Car"))

					);
```

```
		List<String> collect = listOfProduct.stream().flatMap(p ->
p.getListOfProducts().stream()).collect(Collectors.toList());

		System.out.println(collect);


	}


}
```

Difference between map() and flatMap()

---

map()  method transforms each element into another single element.

flatMap() transforms each element into a stream of elements and then flattens those streams into a single stream.

We should use map() when you want a one-to-one transformation, and  we should use flatMap() when dealing with nested structures or when you need to produce multiple output elements for each input element.

---

public Stream sorted() :

---

It is a predefined method of Stream interface.

It provides default natural sorting order.

The return type of this methof is Stream.

//Sorting the data

```
package com.ravi.basic;

import java.util.*;
```

```
import java.util.stream.*;
public class StreamSortedDemo
{
	public static void main(String[] args)
	{
	List<String> names = Arrays.asList("Zaheer","Rahul","Aryan","Sailesh");

		List<String> sortedName =
	names.stream().sorted().collect(Collectors.toList());

   System.out.println(sortedName);
	}
}
```

---

01-03-2024

---

public Stream distinct() :

---

It is a predefined method of Stream interface.

If we want to return stream from another stream by removing all the duplicates then we should use distinct() method.

---

```
package com.ravi.basic;
import java.util.stream.Stream;
public class StreamDemo10
{
	public static void main(String[] args)
```

```
		{
		Stream<String> s = Stream.of("Virat", "Rohit", "Dhoni", "Virat",
"Rohit","Aswin","Bumrah");
		s.distinct().sorted().forEach(System.out::println);

	}

}
```

---

public Stream<T> limit(long maxSize) :

---

public Stream<T> limit(long maxSize) :

---

It is a predefined method of Stream interface to work with sequence of elements.

The limit() method is used to limit the number of elements in a stream by providing maximum size.

It creates a new Stream by taking the data from original Stream.

Elements which are not in the range or beyond the range of specified limit will be ignored.

---

```
package com.ravi.basic;
import java.util.stream.Stream;
public class StreamDemo11
{
	public static void main(String[] args)
	{
```

```
        Stream<Integer> numbers = Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        Stream<Integer> limitedStream = numbers.limit(4);

        limitedStream.forEach(System.out::println);
    }
}
```

---

public Stream<T> skip(long n) :

---

It is a predefined method of Stream interface which is used to skip the elements from begning of the Stream.

It returns a new stream that contains the remaining elements after skipping the specified number of elements which is passed as a parameter.

---

```
package com.ravi.basic;

import java.util.stream.Stream;

public class StreamDemo12
{
    public static void main(String[] args)
    {
        Stream<String> s = Stream.of("Virat", "Rohit", "Dhoni", "Zaheer", "Raina");

        s.skip(2).limit(3).forEach(System.out::println);
    }
}
```

---

public Stream<T> peek(Consumer<? super T> action) :

---

It is a predefined method of Stream interface which is used to perform a side-effect operation on each element in the stream while the stream remains unchanged.

It is an intermediate operation that allows us to perform operation on each element of Stream without modifying original.

The peek() method takes a Consumer as an argument, and this function is applied to each element in the stream. The method returns a new stream with the same elements as the original stream.

```
package com.ravi.basic;


import java.util.List;

import java.util.stream.Collectors;

import java.util.stream.Stream;


public class StreamDemo13

{

	public static void main(String[] args)

	{

		Stream<String> numbers =
Stream.of("Apple","Mango","Grapes","Kiwi","pomogranate");


		List<Integer> doubledNumbers = numbers

		  .peek(str -> System.out.println("Peeking from Original: " +
str.toUpperCase()))

		  .map(num -> num.length())

		  .collect(Collectors.toList());

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

		System.out.println(doubledNumbers);
```

```
        }

}
```

---

public Stream<T> takeWhile(Predicate<T> predicate) :

---

It is a predefined method of Stream interface introduced from java 9 which is used to perform a side-effect operation on each element in the stream while the stream remains unchanged.

*It is used to create a new stream that includes elements from the original stream only as long as they satisfy a given predicate.

StreamDemo14.java

---

```
package com.ravi.basic;

import java.util.stream.Stream;

public class StreamDemo14
{
 public static void main(String[] args)
 {
        Stream<Integer> numbers = Stream.of(10,11,9,13,2,1,100);

    numbers.takeWhile(n -> n > 9).forEach(System.out::println);

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

```
    numbers = Stream.of(12,2,3,4,5,6,7,8,9);

    numbers.takeWhile(n -> n%2==0).forEach(System.out::println);

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

    numbers = Stream.of(1,2,3,4,5,6,7,8,9);

    numbers.takeWhile(n -> n < 9).forEach(System.out::println);

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

    numbers = Stream.of(11,2,3,4,5,6,7,8,9);

    numbers.takeWhile(n -> n > 9).forEach(System.out::println);

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

    Stream<String> stream = Stream.of("Ravi", "Ankit", "Rohan", "Aman", "Ravish");

    stream.takeWhile(str -> str.charAt(0)=='R').forEach(System.out::println);



 }
}
```

---

public Stream<T> dropWhile(Predicate<T> predicate) :

---

It is a predefined method of Stream interface introduced from java 9 which is used to create a new stream by excluding elements from the original stream as long as they satisfy a given predicate.

```
package com.ravi.basic;

import java.util.stream.Stream;

public class StreamDemo15 {

	public static void main(String[] args)
	{
		Stream<Integer> numbers = Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

		numbers.dropWhile(num -> num < 5).forEach(System.out::println);

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

		numbers = Stream.of(15, 8, 7, 9, 5, 6, 7, 8, 9, 10);

		numbers.dropWhile(num -> num > 5).forEach(System.out::println);

	}
}
```

---

*Optional<T> class in java :

---

Optional<T> class in Java :

---

It is a predefined final and immutable class available in java.util package from java 1.8v.

It is a container object which is used to represent an object (Optional object) that may or may not contain a non-null value.

If the value is available in the container, isPresent() method will return true and get() method will return the actual value.

It is very useful in industry to avoid NullPointerException.

Methods of Optional<T> class :

---

1) public static Optional<T> ofNullable(T x) :

---

It will return the object of Optional class with specified value. If the specified value is null then this method will return an empty object of the optional class.

2) public boolean isPresent() :

---

It will return true, if the value is available in the container otherwise it will return false.

3) public T get() :

---

It will get/fetch the value from the container, if the value is not available then it will throw java.util.NoSuchElementException.

4) public T orElse(T defaultValue) :

---

It will return the value, if available  otherwise it will return the specified default value.

5) public static Optional<T> of (T value) :

---

It will return the optional object with the specified value that is non- null value.

6) public static Optional<T> empty() :

---

It will return an empty Optional Object.

---

//Program to verify whether the container has value or not

```
package com.ravi.optional_class_demo;

import java.util.Optional;

public class OptionalDemo1
{
	public static void main(String[] args)
	{
		String str = null;

		Optional<String> optional = Optional.ofNullable(str);

		String orElse = optional.orElse("No value in container");
		System.out.println("Value by orElse :"+orElse);

		//Optional is containing value or not?
```

```
		if(optional.isPresent())
		{
			System.out.println("Value by get :"+optional.get());
		}
		else
		{
			System.err.println("No value is available in the container");
		}


	}

}
```

---

```
//Writing different style of getter
package com.ravi.optional_class_demo;
import java.util.Optional;
class Employee
{
	private Integer empId;
	private String empName;

	public Employee() {}

	public Employee(Integer empId, String empName)
	{
		super();
		this.empId = empId;
```

```
		this.empName = empName;
	}

	//Changing the style of writing getter method
	public Optional<Integer> getEmpId()
	{
		return Optional.ofNullable(empId);
	}

	public Optional<String> getEmpName()
	{
		return Optional.ofNullable(empName);
	}
}

public class OptionalDemo2
{
	public static void main(String[] args)
	{
		Employee emp = new Employee(111,"Ravi");
		//Employee emp = new Employee();

		Optional<Integer> empId = emp.getEmpId();
		if(empId.isPresent())
		{
			System.out.println(empId.get());
		}
		else
```

```
		{
			System.err.println("No id value ");
		}

		Optional<String> empName = emp.getEmpName();
		if(empName.isPresent())
		{
			System.out.println(empName.get());
		}
		else
		{
			System.err.println("No name value ");
		}

	}
}
```

Now the return type of getter must be Optional to create a container as well as to avoid NullPointerException.

---

```
//Program to verify value is available or not

package com.ravi.optional_class_demo;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

public class OptionalDemo3
```

```
{
   public static void main(String[] args)
   {
      List<Optional<String>> optionalList = new ArrayList<>();

      optionalList.add(Optional.of("Ameerpet"));
      optionalList.add(Optional.of("S.R Nager"));
      optionalList.add(Optional.of("Begumpet"));
      optionalList.add(Optional.of("Koti"));
      optionalList.add(Optional.empty());

      for (Optional<String> optional : optionalList)
      {
         if (optional.isPresent())
         {
            System.out.println(optional.get());
         }
         else
         {
            System.out.println("No data is available");
         }
      }
   }
}
```

---

```
//Immutability of Optional class
```

```
package com.ravi.optional_class_demo;
import java.util.Optional;
public class OptionalDemo4
{
   public static void main(String[] args)
   {
      Optional<String> optl = Optional.of("India");
      System.out.println(optl.hashCode());

      Optional<String> newOptnl = modifyOptional(optl);
      System.out.println(newOptnl.hashCode());

      // Check if the original Optional is still the same
      System.out.println("Address is :" + (optl == newOptnl));

   }

   public static Optional<String> modifyOptional(Optional<String> optional)
   {

      if (optional.isPresent())
      {
         return Optional.of("Modified: " + optional.get());
      }
      else
      {
         return Optional.empty();
```

```
    }
  }
}
```

---

02-03-2024

---

New Date and Time API :

---

LocalDate :

---

It is a predefined final class which represents only Date. The java.util.Date class is providing Date and Time both so, only to get the Date we need to use LocalDate class available in java.time package.

```
LocalDate d = LocalDate.now();
```

Here now is a static method of LocalDate class and its return type is LocalDate class. (Factory Method)

LocalTime :

---

It is also a final class which will provide only time.

```
LocalTime d = LocalTime.now();
```

Here now is a static method of LocalTime class and its return type is LocalTime (Factory Method).

LocalDateTime :

---

It is also a final class which will provide Date and Time both without a time zone. It is a combination of LocalDate and LocalTime class.

```
LocalDateTime d = LocalDateTime.now();
```

```
package com.ravi.new_date_time;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;

public class Demo1
{
    public static void main(String[] args)
    {
      LocalDate d = LocalDate.now();
      System.out.println(d);

      LocalTime t = LocalTime.now();
      System.out.println(t);

       LocalDateTime dt = LocalDateTime.now();
       System.out.println(dt);
    }
}
```

---

ZonedDateTime :

---

It is a final class available in java.time package.

It is also provides date and time along with time zone so, by using this class we can work with different time zone in a global way.

```
ZonedDateTime x = ZonedDateTime.now();
ZoneId zone = x.getZone();
```

getZone() is a predefined non static method of ZonedDateTime class which returns ZoneId class, this ZoneId class provides the different zones, by using getAvailableZoneIds() static method we can find out the total zone available using this ZoneId class.

```
package com.ravi.new_date_time;

import java.time.ZoneId;
import java.time.ZonedDateTime;

public class Demo2
{

	public static void main(String[] args)
	{
		ZonedDateTime z = ZonedDateTime.now();
		System.out.println(z);

	  ZoneId zone = z.getZone();
	  System.out.println(zone);

	  System.out.println(ZoneId.getAvailableZoneIds());
```

```
        }

}
```

---

Differenr of() static method :

---

List.of();

Stream.of();

Optional.of();

ZoneId.of();

---

```
package com.ravi.new_date_time;

import java.time.ZoneId;
import java.time.ZonedDateTime;

public class Demo3
{
   public static void main(String[] args)
   {
      ZoneId AusTimeZone = ZoneId.of("Australia/Hobart");
      ZonedDateTime aus = ZonedDateTime.now(AusTimeZone);
      System.out.println("Current Date and Time in Australia Time Zone: " + aus);
   }
}
```

---

DateTimeFormatter :

---

It is a predefined final class available in java.time.format sub package.

It is mainly used to formatting and parsing date and time objects according to Java Date and Time API.

Method :

---

public static DateTimeFormatter ofPattern(String pattern) :

It is a static method of DateTimeFormatter class, It creates a

formatter using user specified String pattern ("dd-MM-yyyy HH:mm:ss")

and LocalDateTime class contains format method which takes

DateTimeFormatter as a parameter and returns the String value.

```
package com.ravi.new_date_time;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public  class Demo4
{
  public static void main(String[] args)
 {
        LocalDateTime now = LocalDateTime.now();
        System.out.println(now);
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM-dd-YYYY");
```

```
        String formattedDateTime = now.format(formatter);
        System.out.println("Formatted DateTime: " + formattedDateTime);
 }
}
```

---

Sealed class in Java :

---

It is a new feature introduced from java 15.

It is an improvement over final keyword.

By using sealed keyword we can declare classes and interfaces as sealed.

It is one kind of restriction that describes which classes and interfaces can extends or implement from Sealed class Or interface

It is similar to final keyword with less restriction because here we can permit the classes to extend from the original Sealed class.

The class which is inheriting from the sealed class must be final, sealed or non-sealed modifiers.

The sealed class must have at-least one sub class.

We can also create object for Sealed class.

It provides the following modifier :

1) sealed : Can be extended only through permitted class.

2) non-sealed : Can be extended by any sub class, if a user wants to give permission to its sub classes

3) permits : We can provide permission to the sub classes, which are inheriting through Sealed class.

4) final : we can declare permitted sub class as final so, it cannot be extended further.

```
package com.ravi.sealed;

sealed class Bird permits Parrot,Sparrow,Peacock
{
	public void fly()
	{
		System.out.println("Bird can fly");
	}
}
 non-sealed class Parrot extends Bird
{
	public void fly()
		{
			System.out.println("Parrot can fly");
		}
}
 final class Sparrow extends Bird
{
	public void fly()
```

```
                {
                        System.out.println("Sparrow can fly");
                }
}
non-sealed class Peacock extends Bird
{
        public void fly()
                {
                        System.out.println("Peacock can fly");
                }
}

public class SealedDemo
{
        public static void main(String[] args)
        {
                Bird b = null;

                b = new Parrot(); b.fly();

                b = new Sparrow(); b.fly();

                b = new Peacock(); b.fly();


        }

}
```

```
-------------------------------------------------------------
package com.ravi.sealed;

sealed class OnlineJavaClass permits Mobile,Laptop
{
	public void attendClass()
	{
		System.out.println("Attend online class");
	}
}

non-sealed class Mobile extends OnlineJavaClass
{
	public void attendClass()
	{
		System.out.println("Attending Java class through Mobile");
	}
}

final class Laptop extends OnlineJavaClass
{
	public void attendClass()
	{
		System.out.println("Attending Java class through Laptop");
	}
}

public class OnlineClass {
```

```
	public static void main(String[] args)
	{
		OnlineJavaClass cls = null;
		cls = new Mobile(); cls.attendClass();
		cls = new Laptop(); cls.attendClass();



	}

}
```

---

Queue interface :-

---

1) It is sub interface of Collection(I)

2) It works in FIFO(First In first out)

3) It is an ordered collection.

4) In a queue, insertion is possible from last is called REAR where as deletion is possible from the starting is called FRONT of the queue.

5) From jdk 1.5 onwards LinkedList class implments Queue interface to handle the basic queue operations.

PriorityQueue :

---

public class PriorityQueue extends AbstractQueue implements Serializable

It is a predefined class in java.util package, available from Jdk 1.5 onwards.

It inserts the elements based on the priority HEAP (Using Binary tree).

The elements of the priority queue are ordered according to their natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.

A priority queue does not permit null elements as well as It uses Binary tree to insert the elements.

It provides natural sorting order so we can't take non-comparable objects(hetrogeneous types of Object)

The initial capacity of PriorityQueue is 11.

Constructor :

---

1) PriorityQueue pq1 = new PriorityQueue();

2) PriorityQueue pq2 = new PriorityQueue(int initialCapacity);

3) PriorityQueue pq3 = new PriorityQueue(int initialCapacity, Comparator cmp);

4-) PriorityQueue pq3 = new PriorityQueue(Comparator cmp);

5) PriorityQueue pq4 = new PriorityQueue(Collection c);

Methods :-

---

add() / offer() :- Used to add an element in the Queue

poll() :- It is used to fetch the elements from top of the queue, after fetching it will delete the element.

peek() :- It is also used to fetch the elements from top of the queue, Unlike poll it will only fetch but not delete the element.

boolean remove(Object element) :- It is used to remove an element. The return type is boolean.

---

```
import java.util.PriorityQueue;

public class PriorityQueueDemo
{
    public static void main(String[] argv)
    {
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("Orange");
                        pq.add("Apple");
                        pq.add("Mango");
                        pq.add("Guava");
                        pq.add("Grapes");

                        System.out.println(pq);
```

```
    }
}
```

---

```
import java.util.PriorityQueue;
public class PriorityQueueDemo1
{
    public static void main(String[] argv)
    {
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("9");
        pq.add("8");
                        pq.add("7");
        System.out.print(pq.peek() + " ");
        pq.offer("6");
                        pq.offer("5");
        pq.add("3");

        pq.remove("1");
        System.out.print(pq.poll() + " ");
        if (pq.remove("2"))
          System.out.print(pq.poll() + " ");
        System.out.println(pq.poll() + " " + pq.peek());
                        System.out.println(pq);
    }
}
```

---

```
import java.util.PriorityQueue;
public class PriorityQueueDemo2
{
    public static void main(String[] argv)
    {
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("2");
        pq.add("4");
                        pq.add("6");  //    6  9
        System.out.print(pq.peek() + " "); //2  2  3  4  4
        pq.offer("1");
                        pq.offer("9");
        pq.add("3");

        pq.remove("1");
        System.out.print(pq.poll() + " ");
        if (pq.remove("2"))
          System.out.print(pq.poll() + " ");
        System.out.println(pq.poll() + " " + pq.peek()+"  "+pq.poll());
           }
}
```

---

```
import java.util.PriorityQueue;
public class PriorityQueueDemo3
{
    public static void main(String[] argv)
    {
        PriorityQueue<Integer> pq = new PriorityQueue<>();
```

```
                        pq.add(11);
        pq.add(2);
        pq.add(4);
                        pq.add(6);
       System.out.println(pq);
          }
}
```

---

Input Output in java :

---

In order to work with input and output concept, java software people has provided a separate package called java.io package.

By using this java.io package we can read the data from the user, creating file, reading/writing the data from the file and so on.

How to take the input from the user using java.io package :

---

Scanner class is available from java 1.5 onwards but before 1.5, In order to read the data we were using the following two classes which are available in java.io package.

1) DataInputStream (Deprecated)

2) BufferedReader

How to create the object :

---

DataInputStream :

---

DataInputStream d = new DataInputStream(System.in);

BufferedRedaer :

---

It provides more faster technique because it internally stores the data in a buffer and it is always recomended to read the data from the buffer.

InputStreamReader isr = new InputStreamReader(System.in);

BufferedReader br = new BufferedReader(isr);

OR

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

Working with Methods :

---

1) public int read() :- It is used to read a single character from the source and return the UNICODE value of the character.

If the data is not available from the source then it will return -1.

2) public String readLine() :- It is used to read multiple characters or complete line from the source. The

return type of this method is String.

Program to read name from the keyboard :

---

```
import java.io.*;
public class ReadName
{
    public static void main(String[] args) throws IOException
    {
```

```
                DataInputStream d = new DataInputStream(System.in);
                System.out.print("Enter Your Name :");
                String name = d.readLine();
                System.out.println("Your Name is :"+name);
        }
}
```

---

```
import java.io.*;
public class ReadAge
{
        public static void main(String[] args) throws IOException
        {
                BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
                System.out.print("Enter Your Age:");
                int age = Integer.parseInt(br.readLine());
                System.out.println("Your Age is :"+age);

                if(age > 18)
                {
                        System.out.println("Go for a Movie");
                }
                else
                {
                        System.out.println("Try after some year");
                }
        }
}
```

```
------------------------------------------------------------
//WAP in java to read a float value(salary) from the keyboard
import java.io.*;
public class ReadSalary
{
	public static void main(String[] args) throws IOException
	{
		BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

		System.out.print("Please enter your Salary :");
		String sal = br.readLine();

	//Converting String to float value
		float salary = Float.parseFloat(sal);

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

	}
}
------------------------------------------------------------
//WAP in java to read a character i.e gender from the keyboard
import java.io.*;
class ReadCharacter
{
	public static void main(String[] args) throws IOException
	{
```

```
                var br = new BufferedReader(new InputStreamReader(System.in));
                System.out.print("Enter your Gender :");
                char gen = (char) br.read();
                System.out.println("Your Are :"+gen);
        }
}
```

---

```
//WAP in java to read employee data
import java.io.*;
public class EmpDataDemo
{
public static void main(String [] args) throws IOException
{
        BufferedReader br = new BufferedReader
        (new InputStreamReader(System.in));

        System.out.print("Enter id: ");
        int id = Integer.parseInt(br.readLine());

        System.out.print("Enter sex (M/F): ");
        //char gen = (char) br.readLine(); //Buffer Problem
        char gen = br.readLine().charAt(0);  //Male \n


        System.out.print("Enter name: ");
        String name = br.readLine();

        System.out.println("Id = "+id);
```

```
        System.out.println("Sex = "+gen);
        System.out.println("Name = "+name);
        }
}
```

```
/*
Enter Id : 111 \n      readLine();

Enter M/F : Male \n    read() -> It will read only single character i.e M

Enter Name : ale

*/
```

---

File Handling :

---

File Handling :

---

What is the need of File Handling ?

---

As we know variables are used to store some meaningful value in our program but once the execution of the program is over, now we can't get those values so to hold those values permanently in our memory we use files.

Files are stored in the secondary storage devices so, we can use/read the data stored in the file anytime according to our requirement.

In order to work with File system java software people has provided number of predefined classes like File, FileInputStream, FileOutputStream and so on. All these classes are available in java.io package. We can read and write the data in the form of Stream.

---

Streams in java :

---

A Stream is nothing but flow of data or flow of characters to both the end.

Stream is divided into two categories

1) byte oriented Stream :-

---

It used to handle characters, images, audio and video file in binary format.

2) character oriented Stream :-

---

It is used to handle the data in the form of characters or text.

Now byte oriented or binary Stream can be categorized as "input stream" and "output stream". input streams are used to read or receive the data where as output streams are used to write or send the data.

Again Character oriented Stream is divided into Reader and Writer. Reader is used to read() the data from the file where as Writer is used to write the data to the file.

All Streams are represented by classes in java.io package.

InputStream is the super class for all kind of input operation where as OutputStream is the super class for all kind of output Operation for byte oriented stream.

Where as Reader is the super class for all kind reading operation where as Writer is the super class for all kind of writing operation in character oriented Stream.

---

File :-

---

It is a predefined class in java.io package through which we can create file and directory. By using this class we can verify whether the file is existing or not.

File f = new File("abc.txt");

The above statement will not create any file, It actually create the file object and perform one of the following two task.

a) If abc.txt does not exist, It will not create it

b) if abc.txt does exist, the new file object will be refer to the referenec variable f

Now if the file does not exist then to create the file and verify whether file is existing or not, we should use the following two methods :

1) public boolean exists() : Will verify whether file is existing or not?

2) public boolean createNewFile() : Will Create a new file , if

file is already available then return false.

File class has also a predefined method called getName(), to get the name of the file.

---

import java.io.*;

```
public class File0
	{
		public static void main(String[] args)
		{
			try
				{
					File f = new File("C:\\new\\India.txt");

	  if(f.exists())
					{
						System.out.println("File is existing");
					}
					else
					{
						System.out.println("File is not existing");
					}

					if (f.createNewFile())
					 {
						 System.out.println("File created: " +
f.getName());
	  }
					  else
						{
	   System.out.println("File is already existing....");
	   }
	  }
				catch (IOException e)
```

```
        {
                                        System.err.println(e);
                                    }
                }
}
```

H.W :- With the help of File class create directory.

---

FileOutputStream : (Create New File + Writing the data)

---

It is a predefined class available in java.io package. The main purpose of this class to create a new file and write the data to the file.

Whenever we want to write the data into the file using FileOutputStream class then data must be available into the form of byte because It is byte-oriented class.

Note :- String class has provided a method called getBytes() through which we can read the String data in byte format and the return type of this method is byte[].(byte array)

Eg:-

```
String x = "India is Great";
byte b [] = x.getBytes();
```

Note :- try with resources are used to automatically close our resources.

---

In binary Stream, whenever we want to write the data, the data must be available in byte format, If we want to print the data

on the console(Monitor) the data must be converted into char.

FileInputStream :-

---

It is a predefined class available in java.io package. It is used to read the file data/content. If we want to print the file data in console then data must be available in char format.

```
//Reading tha data from the file
import java.io.*;
public class File2
{
    public static void main(String s[]) throws IOException
    {
                    var fin = new FileInputStream("C:\\new\\Hyderabad.txt");

                    try(fin)
                    {
                            int i = 0;
          while(true)
                            {
                                    i = fin.read();
                                    if(i==-1)
                                            break;
                                    System.out.print((char)i);
                            }

                    }
                    catch(Exception e)
```

```
                {
                        System.out.println(e);
                }
                System.out.println();
    }
}
```

---

17-03-2024

---

Inner classes in java :

---

In java it is possible to define a class (inner class) inside another class (outer class) called nested class.

In the same way it is also possible to define an interface OR enum OR Annotation inside another interface OR enum OR Annotation is called inner class or nested class.

Inner classes in Java create a strong encapsulation and HAS-A relationship between the inner class and its outer class.

An inner class, .class file will be represented by $ symbol.

Advantages of inner class :

---

1) It is a way of logically grouping classes that are only used in one place.

2) It is used to achieve encapsulation.

3) It enhance the readability and maintainability of the code.

Types of Inner classes in java :

---

There are 4 types of inner classes in java :

1) Member Inner class OR Nested Inner class OR Regular class

2) Local inner class OR Method local inner class

3) Nested inner class

4) Anonymous inner class.

1) Member Inner class OR Nested Inner class OR Regular class :

---

A non-static class that is created inside a class but outside of a method is called Member Inner class OR Nested Inner class OR Regular class.

It can be declared with access modifiers like private, default, protected, public, abstract and final.

It is also called as Regular Inner class.

An inner class can also access the private member of outer class.

Note :- The .class file of an inner class will be represented by $ symbol at the time of compilation.

An outer class can be declared as public, abstract and final only.

---

```
package com.ravi.inner_class;
```

```
class Outer
{
	private int x = 100;

	public class Inner
	{
		private int y = 200;

		public void show()
		{
			System.out.println(x);
			System.out.println(y);
		}

	}
}
public class NestedDemo1
{
	public static void main(String[] args)
	{
		Outer outer = new Outer();
		Outer.Inner inner = outer.new Inner();
		inner.show();

		//OR

		Outer.Inner in = new Outer().new Inner();
		in.show();
```

```
        }

}
------------------------------------------------------------
class MyOuter
{
    private int x = 7;
    public void makeInner()
    {
        MyInner in = new MyInner();
                        System.out.println("Inner y is "+in.y);
        in.seeOuter();
    }

    class MyInner
    {
                    private int y = 15;
        public void seeOuter()
        {
            System.out.println("Outer x is "+x);
        }
    }
}
public class Test2
{
    public static void main(String args[])
    {
        MyOuter m = new MyOuter();
```

```
        m.makeInner();
    }
}
```

Note : From the above program it is clear that an inner can access the private data of outer class as well as an outer class can also access the private data of inner class.

---

```
class MyOuter
{
    private int x = 15;
    class MyInner
    {
        public void seeOuter()
        {
            System.out.println("Outer x is "+x);
        }
    }
}
public class Test3
{
    public static void main(String args[])
    {
                    //Creating inner class object in a single line
        MyOuter.MyInner m = new MyOuter().new MyInner();
                            m.seeOuter();
    }
}
```

---

```
class MyOuter
{
    static int x = 7;
          class MyInner
    {
        public static void seeOuter()  //MyInner.seeOuter();
        {
            System.out.println("Outer x is "+x);
        }
    }
}

public class Test4
{
    public static void main(String args[])
    {
       MyOuter.MyInner.seeOuter();
    }
}
```

---

24-03-2024

---

```
class Car
{
   private String make;
   private String model;
   private Engine engine;
```

```
public Car(String make, String model, int horsePower)
{
    this.make = make;
    this.model = model;
    this.engine = new Engine(horsePower);
 }

//Inner class
private class Engine
{
  private int horsePower;

   public Engine(int horsePower)
          {
      this.horsePower = horsePower;
    }

    public void start()
                  {
      System.out.println("Engine started! Horsepower: " + horsePower);
    }

    public void stop()
          {
      System.out.println("Engine stopped.");
    }
 }
```

```
    public void startCar()
            {
        System.out.println("Starting " + make + " " + model);
        this.engine.start();
    }

    public void stopCar()
            {
        System.out.println("Stopping " + make + " " + model);
        this.engine.stop();
    }
}
public class Test5
{

    public static void main(String[] args) {

        Car myCar = new Car("Swift", "Desire", 1200);

        myCar.startCar();

        myCar.stopCar();
    }
}
```

Note :- In the above example Car object is interacting with Engine object.

---

```
class Laptop
```

```
{
    private String brand;
    private String model;
    private Motherboard motherboard;

    public Laptop(String brand, String model, String motherboardModel, String chipset)
            {
        this.brand = brand;
        this.model = model;
        this.motherboard = new Motherboard(motherboardModel, chipset);
    }


    public void switchOn()
            {
        System.out.println("Turning on " + brand + " " + model);
        this.motherboard.boot();
    }

   //Motherboard inner class
   public class Motherboard
   {
        private String model;
        private String chipset;


        public Motherboard(String model, String chipset)
                        {
```

```
        this.model = model;
        this.chipset = chipset;
    }


    public void boot()
                {
        System.out.println("Booting " + brand + " " + model + " with " + chipset + "
chipset");
    }
  }
}
public class Test6
{
  public static void main(String[] args)
        {

    Laptop laptop = new Laptop("HP", "ENVY", "IRIS", "Intel");

    laptop.switchOn();
  }
}
---------------------------------------------------------------
class Person
{
  private String name;
  private int age;
  private Heart heart;
```

```
public Person(String name, int age)
        {
   this.name = name;
   this.age = age;
   this.heart = new Heart();
}

public void describe()
        {
   System.out.println("Name: " + name);
   System.out.println("Age: " + age);
   System.out.println("Heart beats per minute: " + heart.getBeatsPerMinute());
}

// Inner class representing the Heart
private class Heart
        {
   private int beatsPerMinute;

   public Heart() {
      this.beatsPerMinute = 72;
   }

   public int getBeatsPerMinute()
                {
      return this.beatsPerMinute;
   }
```

```
    public void setBeatsPerMinute(int beatsPerMinute)
                {
      this.beatsPerMinute = beatsPerMinute;
    }
  }
}
public class Test7
{
  public static void main(String[] args)
        {
    Person person = new Person("Virat", 30);
    person.describe();
  }
}
---------------------------------------------------------------
class University
{
  private String name;

  public University(String name)
        {
    this.name = name;
  }

  public void displayUniversityName()
        {
    System.out.println("University Name: " + name);
```

```
  }

  public class Department
        {
    private String name;
    private String headOfDepartment;

    public Department(String name, String headOfDepartment)
                {
      this.name = name;
      this.headOfDepartment = headOfDepartment;
    }

    // Method to display department details
    public void displayDepartmentDetails()
                {
      System.out.println("Department Name: " + name);
      System.out.println("Head of Department: " + headOfDepartment);
      displayUniversityName();
    }
  }
}
public class Test8
{
  public static void main(String[] args)
        {

    University university = new University("JNTU");
```

```
    University.Department cs = university.new Department("Computer Science", "Dr. John");

    University.Department ee = university.new Department("Electrical Engineering", "Dr. Scott");

    cs.displayDepartmentDetails();
    ee.displayDepartmentDetails();
  }
}
---------------------------------------------------------------
class OuterClass
{
        private int x = 200;
        class InnerClass
        {
                public void display()  //Inner class display method
                {
                System.out.println("Inner class display method");
                }

                public void getValue()
                {
                        display();
                        System.out.println("Can access outer private var :"+x);
                }
        }
```

```
		public void display()  //Outer class display method
		{
			System.out.println("Outer class display");
		}
}
public class Test9
{
	public static void main(String [] args)
	{
		OuterClass.InnerClass inobj = new OuterClass().new InnerClass();
		inobj.getValue();

		new OuterClass().display();
	}
}
-----------------------------------------------------------------
class OuterClass
{
	int x;
	final class InnerClass  //private, def, prot, public, abstract & final
	{
		int x;
	}
}
public class Test10
{
}
```

---

2) Method local inner class :

---

If a class is declared inside the method then it is called method local inner class.

We cann't apply any access modifier on method local inner class but they can be marked as abstract and final.

A local inner class we can't access outside of the method that means the scope of method local inner class within the same method only.

```
package com.ravi.has_a_reln;


class Outer
{
	private String str = "Outer Data";

	public void access()
	{
		class Local
		{
			String str = "Local class Data";

			public void accessOuter()
			{
				System.out.println(Outer.this.str);
				System.out.println(str);
			}
		}
```

```
                new Local().accessOuter();
        }
}

public class Test11 {

        public static void main(String[] args)
        {
                new Outer().access();
        }

}
```

---

```
//local inner class we can't access outside of the method
class MyOuter3
{
    private String x = "Outer class Data";

    public void doSttuff()
    {
        String z = "local variable";
         class MyInner
        {
                                        String z = "CLASS variable";
            public void seeOuter()
            {
                System.out.println("Outer x is "+x);
                System.out.println("Local variable z is : "+z);
```

```
        }
    }
        }
                MyInner mi = new MyInner();
            mi.seeOuter();




}
public class Test12
{
    public static void main(String args[])
    {
        MyOuter3 m = new MyOuter3();
        m.doSttuff();
    }
}
```

---

3) Static Nested Inner class :

---

A static inner class which is declared with static keyword inside an outer class is called static Nested inner class.

It cann't access non-static variables and methods  i.e (instance members) of outer class.

For static nested inner class, Outer class object is not required.

If a static nested inner class contains static method then object is not required for inner class. On the other hand if the static inner class contains instance method then we need to create an object for static nested inner class.

```
//static nested inner class
class BigOuter
{
   static class Nest   //static nested inner class
   {
      void go()  //Instance method of static inner class
      {
         System.out.println("Hello welcome to static nested class");
      }
   }
}
class Test13
{
    public static void main(String args[])
    {
      BigOuter.Nest n = new BigOuter.Nest();
      n.go();



    }
}
```

---

```
class Outer
{
            static int x=15;
```

```
        static class Inner
        {
                        void msg()
                                {
                                        System.out.println("x value is  "+x);
                                }
        }
}
class Test14
{
        public static void main(String args[])
        {
                Outer.Inner obj=new Outer.Inner();
                obj.msg();
        }
}
---------------------------------------------------------------
class Outer
{
        static int x = 25;
        static class Inner
        {
                        static void msg()
                                {
                                        System.out.println("x value is  "+x);

                                }
```

```
    }
}
class Test15
{
    public static void main(String args[])
    {
        Outer.Inner.msg();
    }
}
-----------------------------------------------------------------
class Outer
{
    int x=15;  //error (not possible because try to access instance variable)
    static class Inner
    {
            void msg()
                {
                    System.out.println("x value is  "+x);
                }
    }
}
class Test16
{
    public static void main(String args[])
    {
        Outer.Inner obj=new Outer.Inner();
        obj.msg();
    }
```

}

---

4) Anonymous inner class :

---

It is an inner class which is defined inside a method without any name and for this kind of inner class only single Object is created. (Singleton class)

*An anonymous inner class is mainly used for extending a class or implementing an interface that means creating sub type of a class or interface.

*A normal class can implement any number of interfaces but an anonymous inner class can implement only one interface at a time.

A normal class can extend one class and implement any number of interfaces at the same time but an anonymous inner class can extend one class or can implement one interface at a time.

Inside an anonymous inner class we can write static , non static variable, static block and non-static block. Here we can't write abstract method and constructor.

```
//Anonymous inner class
class Tech
{
    public void tech()
    {
            System.out.println("Tech");
    }
}
public class Test17
{
```

```
    public static void main(String... args)
    {
            Tech a = new Tech()  //Anonymous inner class
                    {
                            @Override
                                public void tech()
                                {
                                        System.out.println("anonymous tech");
                                }
                    };
                    a.tech();
    }
}
-----------------------------------------------------------------
@FunctionalInterface
interface Vehicle
{
        void move();  //SAM(Single Abstract Method)
}

class Test18
{
        public static void main(String[] args)
        {
                Vehicle car = new Vehicle()
                {
                        @Override
                        public void move()
```

```
                {
                        System.out.println("Moving with Car...");
                }
        };
        car.move();

    Vehicle bike = new Vehicle()
        {
                @Override
                public void move()
                {
                        System.out.println("Moving with Bike...");
                }
        };
        bike.move();
    }
}
------------------------------------------------------------
class Test19
{
    public static void main(String[] args)
    {
     //Anonymous class with Runnable
        Runnable r1 = new Runnable()
        {

                @Override
                public void run()
```

```
                {
    System.out.println("Run method implementation inside Runnable");
                }
            };
   Thread obj = new Thread(r1);
                 obj.start();



//Anonymous Thread class with reference
            Thread t1 = new Thread()
            {
                @Override
                public void run()
                {
    System.out.println("Anonymous Thread class with reference...");
                }
            };
            t1.start();

//Anonymous Thread class without reference
new Thread()
            {
                @Override
                public void run()
                {
    System.out.println("Anonymous Thread class without reference...");
                }
```

```
                }.start();
        }
}
```

---

Method Reference :

---

It is a new feature introduced from java 1.8 onwards.

It is mainly used to write concise coding.

By using method reference we can refer an existing method which is available at API level or Project level.

We can use this technique in the body of Lambda expression just to call method defination.

The enitire method body will be automatically placed into Lambda Expression.

It is used to enhance the code reusability.

It uses :: (Double Colon Operator)

Method Reference Examples :

---

MethodRef.java

---

```
package com.ravi.method_ref;
```

```
interface Worker
{
	void work();
}

class Employee
{
	public void work()
	{
		System.out.println("Employee is working");
	}
}



public class MethodRef {

	public static void main(String[] args)
	{
		//By using Lambda
		Worker w1 = ()-> System.out.println("Worker is working");
		w1.work();

		//By using method reference
		Worker w2 = new Employee()::work;
		w2.work();

	}
```

```
}
---------------------------------------------------------------
package com.ravi.method_ref;

interface Worker
{
	void work();
}

class Employee
{
	public static void salary()
	{
		System.out.println("Employee Salary");
	}
}



public class MethodRef {

	public static void main(String[] args)
	{
		Worker w1 = Employee::salary;
		w1.work();

	}

}
```

```
-----------------------------------------------------------------
package com.ravi.method_ref;

interface Worker
{
	void work(double salary);
}

class Employee
{

	public static void salary()
	{
		System.out.println("Employee Salary");
	}

	public static void salary(double salary)
	{
		System.out.println("Scott Salary :"+salary);
	}
}


public class MethodRef {

	public static void main(String[] args)
	{
		Worker w1 = Employee::salary;
```

```
                w1.work(23000);

        }

}
```

---

There are 4 types of method reference

1) Static Method Reference(ClassName::methodName)

2) Instance Method Reference(objectReference::methodName)

3) Constructor Reference (ClassName::new)

4) Arbitrary Referenec (ClassName::instanceMethodName)

Programs on above references :

---

Program on static method Reference :

---

```
package com.ravi.static_method_reference;

import java.util.Vector;

class EvenOrOdd
{
        public static void isEven(int number)
  {
    if (number % 2 == 0)
    {
```

```
        System.out.println(number + " is even");
      }
      else
      {
        System.out.println(number + " is odd");
      }
   }
}
public class StaticMethodReferenceDemo1
{
        public static void main(String[] args)
   {
     Vector<Integer> numbers = new Vector<>();
     numbers.add(5);
     numbers.add(2);
     numbers.add(9);
     numbers.add(12);
     System.out.println("By using Lambda....");
     numbers.forEach(x -> EvenOrOdd.isEven(x));

     System.out.println("_____");

     System.out.println("By using static method reference");
     numbers.forEach(EvenOrOdd::isEven);
   }
}
```

---

2)Instance Method Reference

```
----------------------------

package com.ravi.instance_method_reference;


interface Trainer

{

 void getTraining(String name, int experience);

}


class InstanceMethod

{

  public void getTraining(String name, int experience)

  {

            System.out.println("Trainer name is :"+name+" having "+experience+" years of experience.");

  }

}


public class InstanceMethodReferenceDemo

{

   public static void main(String[] args)

   {

        //Using Lambda Expression

        Trainer t1 = (name,  exp)-> System.out.println("Trainer name is :"+name+" and total experience is :"+exp);

        t1.getTraining("Smith", 5);



        //By using Method reference

        Trainer t2 = new InstanceMethod()::getTraining;
```

```
            t2.getTraining("Scott", 10);




    }
}
```

---

3) Constructor Reference (ClassName::new)

---

```
package com.ravi.constructor_reference;

@FunctionalInterface
interface A
{
   Test createObject();
}

class Test
{
   public Test()
   {
      System.out.println("Test class Constructor invoked");
   }
}
public class ConstructorReferenceDemo1
{
```

```
   public static void main(String[] args)
   {
      //By using Lambda
      A a1 = ()-> new Test();

      a1.createObject();

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

     //By Using Method Reference
      A a2 = Test::new;   //calling Test class constructor
      a2.createObject();
   }
}
```

---

```
package com.ravi.constructor_reference;

import java.util.function.Function;

class MyClass
{
   private int value;

   public MyClass(int value)
   {
      this.value = value;
   }
```

```
    public int getValue()
    {
        return this.value;
    }
}

public class ConstructorReferenceDemo2
{
    public static void main(String[] args)
    {
        Function<Integer, MyClass> constructorRef = MyClass::new;

        MyClass obj = constructorRef.apply(10);

        System.out.println("Value: " + obj.getValue());
    }
}
```

---

//Array Constructor Reference

```
package com.ravi.constructor_reference;

import java.util.Scanner;
import java.util.function.Function;

class Person
{
```

```
    private String name;

    public Person(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}

public class ConstructorReferenceDemo3
{
    public static void main(String[] args) {

        Function<Integer, Person[]> arrayConstructorRef = Person[]::new;

        Person[] people = arrayConstructorRef.apply(5); //5 is size of Array

        Scanner sc = new Scanner(System.in);

        for (int i = 0; i < people.length; i++)
        {
            System.out.println("Enter person name at "+i+" position :");
            String name = sc.nextLine();
           people[i] = new Person(name);
```

```
    }


    System.out.println("Names of people:");
    for (Person person : people)
    {
       System.out.println(person.getName());
    }
  }
}
```

---

Arbitrary Reference :(ClassName::instanceMethodName)

---

Reference to an instance method of an arbitrary object of a particular type.

| Lambda Expression | Method Reference |
| --- | --- |
| s -> s.length(); | String::length; |
| s -> s.toUpperCase(); | String::toUpperCase; |
| (i1,i2)-> i1.compareTo(i2); | Integer::compareTo |
| (s1,s2)-> s1.compareTo(s2); | String::compareTo |

---

```
package com.ravi.arbitary_reference;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;
```

```
public class ArbitaryRefDemo1
{
  public static void main(String[] args)
  {
        List<Integer> list = Arrays.asList(9,5,3,7,2);

        //By using Lambda
        Collections.sort(list,(i1,i2)->i1.compareTo(i2));
        list.forEach(System.out::println);

        //By Arbitrary Reference
        Collections.sort(list,Integer::compareTo);
        list.forEach(System.out::println);

        String[] stringArray = { "Virat","Rohit","Ajinkya","Dhoni","Aswin"};
    Arrays.sort(stringArray, String::compareTo);
    System.out.println(Arrays.toString(stringArray));

  }
}
```

---

```
package com.ravi.arbitary_reference;

import java.util.Arrays;

class Person
{
```

```
   String name;

   public Person(String name)
   {
      this.name = name;
   }

   public int personInstanceMethod1(Person person)
   {
      return this.name.compareTo(person.name);
   }

         @Override
         public String toString() {
                  return "Person [name=" + name + "]";
         }


}

public class ArbitraryRefDemo2
{
   public static void main (String[] args) throws Exception
   {
      Person[] personArray = {new Person("C"),new Person("B"), new Person("A")};
      Arrays.sort(personArray, Person::personInstanceMethod1);

      System.out.println(Arrays.toString(personArray));
```

```
    }

}
```

Note : Any instance method we can refere by using the class name

which known as Arbitrary method reference.

---