## Data Types and Operators Program

package com.datatype.operaters;  
  
public class DataTypeExample {  
 public static void main(String[] args) {  
// DataTypeExample dataTypeExample=new DataTypeExample();  
// dataTypeExample.dataType();  
// arithmeticOperator();  
// comparisonOperator();  
// assignmentOperator();  
// bitwiseOperator();  
// conditionalOperators();  
// preIncrementOperator();  
 *postIncrementOperator*();  
 }  
 public void dataType(){  
 int a=10;  
 long l=10000000;  
 double d=10.45678;  
 float f=10.45f;  
 char c='s';  
 String name="shalini";  
 System.*out*.println("Integer value is:"+a);  
 System.*out*.println("Long value is:"+l);  
 System.*out*.println("Double value is:"+d);  
 System.*out*.println("Float value is:"+f);  
 System.*out*.println("Charecter value is:"+c);  
 System.*out*.println("String value is:"+name);  
  
 }  
 public static void arithmeticOperator(){  
 int numberOne=4;  
 int numberTwo=3;  
 System.*out*.println("Add of two numbers:"+(numberOne+numberTwo));  
 System.*out*.println("Sub of two numbers:"+(numberOne-numberTwo));  
 System.*out*.println("Mul of two numbers:"+(numberOne\*numberTwo));  
 System.*out*.println("Div of two numbers:"+(numberOne/numberTwo));  
 System.*out*.println("Modulus of two numbers:"+(numberOne%numberTwo));  
 System.*out*.println("Floor Division is:"+Math.*floorDiv*(numberOne,numberTwo));  
 }  
 public static void comparisonOperator(){  
 int numberOne=4;  
 int numberTwo=3;  
 System.*out*.println("greater:"+(numberOne>numberTwo));  
 System.*out*.println("lessor:"+(numberOne<numberTwo));  
 System.*out*.println("Equal:"+(numberOne==numberTwo));  
 System.*out*.println(" Not Equal:"+(numberOne!=numberTwo));  
 System.*out*.println("greater or equal:"+(numberOne>=numberTwo));  
 System.*out*.println("lessor or equal:"+(numberOne<=numberTwo));  
 }  
 public static void assignmentOperator(){  
 int numberOne=5,numberTwo=4;  
 int numberThree=numberOne+numberTwo;  
 System.*out*.println(numberThree);//9  
 numberThree+=numberOne;//c=c+a,c=9+5=14  
 System.*out*.println(numberThree);  
 numberTwo\*=numberThree;//b=b\*c,4\*14  
 System.*out*.println(numberTwo);  
 }  
 public static void bitwiseOperator(){  
 int a=10,b=4;  
 System.*out*.println(a&b);//&-AND  
 System.*out*.println(a|b);//|-OR  
 System.*out*.println(a^b);//^-XOR  
 System.*out*.println(~a);//~-NOT  
 System.*out*.println(a>>2);//>>-RightShift  
 System.*out*.println(a<<2);//<<-leftShift  
 }  
 public static void logicalOperator(){  
 int a=10,b=4;  
 System.*out*.println((a>b)&&(a<b));//t f,f  
 System.*out*.println((a>b)||(a<b));  
 System.*out*.println(!(a>b)&&(a<b));  
  
  
 }  
 public static void conditionalOperators(){  
 int a=10,b=4,c;  
 c=((a>b)?a:b);  
 System.*out*.println("c value is:"+c);  
 }  
 public static void preIncrementOperator(){  
 System.*out*.println("++preIncrement");  
 int a=5;  
 System.*out*.println(a);  
 System.*out*.println(++a);  
 System.*out*.println(a);  
 }  
 public static void postIncrementOperator(){  
 System.*out*.println("postIncrement++");  
 int a=5;  
 System.*out*.println(a);  
 System.*out*.println(a++);  
 System.*out*.println(a);  
 }  
}

**Programing Concept Program**

**Control Statement or Control Structure:**

**Def:A control stmt is a statement that determine whether the other stmt will be exceuted or not.It control the flow of the program**

**There are three type of control statement in java**

**1.Decision making**

**2.Looping Statement**

**3.Jump Statement or Branch Statement**

**Decision Making:**

**Execute a piece of code based on some condition**

**Decision making Statements are:**

* **If**
* **If-else**
* **Else-if**
* **Switch**
* **Nested if -else**

**If Statement:**

**The if stmt is used to decide whether a particular block of code executed or not based on certain condition.If the condition is true,then the block of code is executed otherwise not executed**

**Syntax:**

**If(condition){**

**# block of code executed if the condition is true**

**}**

**If-else Statement:**

**If condition is true ,if block will be executed otherwise elseblock is executed**

**Syntax:**

**If(condition){**

**#condition true block of code executed**

**}**

**Else{**

**#otherwise that block executed**

**}**

**else if:**

**1.if stmt is followed by multiple else-if block**

**2.In all if,else-if condition are not true last else block will be executed**

**Syntax:**

**If(condition){**

**#block of stmt**

**}**

**Else-if(condition){**

**#block of stmt**

**}**

**Else-if(condition){**

**#block of stmt**

**}**

**Else{**

**#block of stmt**

**}**

**Switch Statement:**

In Java to execute a particular code block when a certain condition is met.

**Syntax:**

**switch(expression) {**

**case 1:**

**// code block**

**break;**

**case 2:**

**// code block**

**break;**

**case 3:**

**// code block**

**break;**

**default:**

**// code block**

**}**

**Looping Statement:**

**Looping stmt that executed a block of code repeatedly while some condition evaluvate true**

**Types:**

**1.while loop**

**2.do while loop**

**3.for loop**

**While loop:**

**The block of statement until specified Boolean condition is false,the loop automatically stop.It is also called entry controlled loop**

**Syntax:**

**while(boolean condition)**

**{**

**//statements;**

**}**

**Do while:**

**Java do while loop executes the statement first and then checks for the condition. Other than that it is similar to the while loop. The difference lies in the fact that if the condition is true at the starting of the loop the statements would still be executed, however in case of while loop it would not be executed at all.**

**Syntax:**

**Do{**

**#exceuted block of cod**

**}while(condition)**

**For Loop:-**

**For loop statement consists of initialization of variable,a condition,and an increment/decrement value,all in one line.**

**It execute the body of the loop until the condition is false**

**Syntax:**

**for(initialization; condition ; increment/decrement)**

**{**

**statement(s);**

**}**

**Conditonal operators:-**

**In Java, conditional operators check the condition and decides the desired result on the basis of both conditions.**

**(OR)**

**Conditional operators are used to evaluate a condition that's applied to one or two boolean expressions. The result of the evaluation is either true or false.**

**Types of conditional operators:**

**&& the logical AND operator.**

**|| the logical OR operator.**

**?: the ternary operator.**

**Syntax:**

**variable = (condition) ? expression1 : expression2**

**Jump OR branching Statement:**

**To move program execution from one location to another location**

**Break Statement:-**

**The break statement is frequently used to terminate the processing of a particular case within a switch statement.**

**Continue Statement:**

**The Java continue statement is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.**

package com.programingconcept;  
  
public class ProgramingConceptExample {  
 public static void main(String[] args) {  
// ifExample();  
// ifElseExample();  
// elseIfExample();  
// switchExample();  
// whileLoopExample();  
// doWhileExample();  
// forLoopExample();  
 *breakStatementExample*();  
 *continueStatement*();  
  
 }  
  
 public static void ifExample() {  
 int aValue = 12;  
 int bValue = 25;  
 int cValue = 26;  
 if (aValue > bValue) {  
 System.*out*.println("aValue is greater");  
 }  
 if (bValue > cValue) {  
 System.*out*.println("bValue is greater");  
 }  
 if (cValue > aValue) {  
 System.*out*.println("cValue is greater");  
 }  
 }  
  
 public static void ifElseExample() {  
 int number = 5;  
 if (number % 2 == 0) {  
 System.*out*.println("the given number is even");  
 } else {  
 System.*out*.println("The given number is odd");  
 }  
  
 }  
  
 public static void elseIfExample() {  
 String browser = "chrome";  
 if (browser == "safri") {  
 System.*out*.println("browser name is safri");  
 } else if (browser == "Edge") {  
 System.*out*.println("browser name is Edge");  
  
 } else if (browser == "chrome") {  
 System.*out*.println("browser name is chrome");  
  
 } else {  
 System.*out*.println("nothing");  
 }  
  
 }  
  
 public static void switchExample() {  
 String name = "shalini";  
 switch (name) {  
 case "shalu":  
 System.*out*.println("my name is shalu");  
 break;  
 case "lini":  
 System.*out*.println("my name is lini");  
 break;  
 case "sobith":  
 System.*out*.println("my name is sobith");  
 break;  
 default:  
 System.*out*.println("my name is shalini");  
 }  
 }  
  
 public static void whileLoopExample() {  
 int num = 10;  
 while (num > 0) {  
 num--;  
 System.*out*.println("The value of number is:" + num);  
// num--;  
 }  
 }  
  
 public static void doWhileExample() {  
 int num = 10;  
 do {  
 // num--;  
 System.*out*.println(num);  
  
 // Update Section  
 num--;  
 } while (num > 0);  
  
 }  
  
 public static void forLoopExample() {  
// for(int num=0;num<10;num++){//increment  
// System.out.println(num);  
// System.out.print(num);  
 for (int num = 10; num > 0; num--) {//decrement  
 System.*out*.println(num);  
 }  
  
 }  
 public static void breakStatementExample(){  
 for(int i=1;i<10;i++){  
 System.*out*.print(i);  
 if(i==5){  
 break;  
 }  
 }  
  
 }  
 public static void continueStatement(){  
 for(int i=0;i<10;i++){  
  
 if(i==5){  
 continue;  
 }  
 System.*out*.println(i);  
 }  
 }  
  
}

**Variables and Methods Creation Program**

**Variable:**

Variable is a data container that can be stored the value

Ex:int a=10;

Where, a is variable,10 is value,int is data type

**Types of Variable:**

1.Local variable

2.Instance variable

3.Static variable

**Local Variable program**

**LOCAL VARIABLE:**

**A variable defined within a block or method or constructor is called local variable (OR) A variable that are defined inside the body of method**

package com.variableExample;  
  
public class LocalVariable {  
 public static void main(String[] args){  
 // getAdd();  
// addition(3,4);  
 new LocalVariable(5,6);  
 }  
 static int *a*=10;  
 static int *b*=5;  
 static void getAdd(){  
  
 int result =*a*+*b*;  
 System.*out*.println("Result is:"+result);  
 }  
  
  
 //method  
 private static void addition(int firstNumber, int secondNumber){  
 int result=firstNumber+secondNumber;  
 System.*out*.println(result);  
  
 }  
  
  
 //constructor  
 LocalVariable(int firstNumber,int secondNumber){  
 int result=firstNumber+secondNumber;  
 System.*out*.println(result);  
  
 }  
  
}

**Instance variable in java**

**INSTANCE VARIABLE OR GLOBAL VARIABLE:**

**Java variable is defined without the static keyword,but an outside of the method declarations**

**(OR)**

**Instance variable are non-static variables and are declare in a class outside of the methoed,constructor or block(ie different type of class created in same variable name,these variables are confused in java so at the time create in instance variable)**

**Instance variable are declare in inside the body of class**

package com.variableExample;  
  
public class InstanceVariable {  
 public static void main(String[] args){  
 new InstanceVariable(120,68);  
 InstanceVariable instanceVariable=new InstanceVariable(120,34);  
 instanceVariable.nowPrintWeight();  
 *run*();  
  
 }  
 int height;  
 int weight;  
 InstanceVariable(int height,int weight){  
 this.height=height;  
 this.weight=weight;  
 System.*out*.println(height+","+weight);  
  
 }  
 public static void run(){  
  
 System.*out*.println("huff,,,puff");  
 }  
 public void nowPrintWeight(){  
  
 System.*out*.println("Now print weight is:"+weight);  
 }  
  
}

**Static variables program**

**STATIC VARIABLE:**

**Static variable can be used to refer to the common property of all objects(whitch is not unique for each object)**

**It saves memory.**

package com.variableExample;  
//class emp  
  
public class StaticVariable {  
 public static void main(String[] args) {  
 StaticVariable shalinidetail=new StaticVariable(1,"Shalini",3000);  
 StaticVariable linidetail=new StaticVariable(2,"Lini",4000);  
// StaticVariable shaludetail=new StaticVariable(3,"Shalu",5000,"sss");  
 shalinidetail.displayEmpDetail();  
 linidetail.displayEmpDetail();  
// shaludetail.displayEmpDetail();  
 }  
 int id;  
 String name;  
 int salary;  
 static String *ceo*="shalini";  
 StaticVariable(int id,String name,int salary){  
 this.id=id;  
 this.name=name;  
 this.salary=salary;  
// this.ceo=ceo;  
  
 }  
 public void displayEmpDetail(){  
 System.*out*.println("Emp ID is:"+this.id+";"+"Emp NAME is:"+this.name+";"+"Emp SALARY is:"+this.salary+";"+"CEO NAME is:"+this.*ceo*);  
 }  
}

**Arrays:**

* **Arrays is a non-primitive data type,that contains similar type of data can be stored in single variable,that variable are store both primitive and non primitive type of data in it**
* **The index of array start from 0,ie means the first element will be stored at the 0th index,second element store in 1st index**
* **The size of array cannot be increased at the run time.Therfore we can store only a fixed size of elements in an array**
* **Array can be declared in two type**

**Eg:1.int variablename[]={1,2,3,4,5}**

**2.int day []=new int [7]**

**day[0]=1**

**day[1]=2**

**.**

**.**

**.**

**Day[6]=6**

**Types of Array:**

**1.Single dimentional Array**

**2.Two dimentional Arrays**

**3.multidimentional Arrays**

**Single dimentional Array:**

**Group of values decleared in single index**

**(or)**

**One dimentional array is a collection of similar type of elements stored at adjacent(near by) memory location**

**Two dimentional Arrays:**

**It has stored in the form of rows and columns.The index start from 0,0**

**Arrays Program**

package com.Arrays;  
  
import java.util.Scanner;  
  
public class SingleDimentionalArrays {  
 public static void main(String[] args) {  
 SingleDimentionalArrays singleDimentionalArrays = new SingleDimentionalArrays();  
// singleDimentionalArrays.singleDimentionalArrays();  
 singleDimentionalArrays.twoDimentionalArrays();  
 // singleDimentionalArrays.MultiplicationTable();  
 // singleDimentionalArrays.TriangularFormMultiplicationTable();  
 // singleDimentionalArrays.starPattern();  
 // singleDimentionalArrays.rightStarPattern();  
 // singleDimentionalArrays.reverseString();  
 //singleDimentionalArrays.pyramid();  
  
  
 }  
  
 private void singleDimentionalArrays(){  
 String variableName[]={"Hello","Hi","How","Are","You"};  
 variableName[1]="sir";  
 for(int i=1;i<variableName.length;i++){  
 System.*out*.print(variableName[i]+" ");  
 }  
 }  
  
  
  
  
 private void twoDimentionalArrays() {  
 int variableName[][] = {{1, 2, 3, 4}, {5, 6, 7}};  
 System.*out*.println(variableName[0][3]);//6  
 variableName[1][2] = 10;//7 instead 10  
 System.*out*.println(variableName[1][2]);  
 for (int i = 0; i < variableName.length; i++) {//i=0;i<4;++i  
 for (int j = 0; j < variableName[i].length; j++)  
 System.*out*.print(variableName[i][j] + " ");  
 System.*out*.println();  
 }  
  
  
 }  
  
 //To print a multiplication table for any number  
 /\* private void MultiplicationTable(){  
 System.out.println("Enter any Number:");  
 Scanner scanner=new Scanner(System.in);  
 int number=scanner.nextInt();  
 for (int i=1;i<=10;i++){  
 System.out.println(number+"X"+i+"="+number\*i);  
 }  
 }  
  
 \*/  
 //To Print the Multiplication Table in Triangular Form  
// private void TriangularFormMultiplicationTable(){  
// System.out.println("Enter any number:");  
// Scanner scanner=new Scanner(System.in);  
// int number=scanner.nextInt();  
// for(int i=1;i<=number;i++){  
// for (int j=1;j<=i;j++){  
// System.out.print(i\*j+" ");  
// }  
//  
// }  
 //}  
 //left stare pattern programe  
 /\* private void starPattern(){  
 Scanner scanner=new Scanner(System.in);  
 System.out.println("u have enter row");  
 int row=scanner.nextInt();  
 for (int i = 1; i <= row; i++)//1,2,3,4,5  
 {  
 for (int j = 1; i <= j; j++)//1  
 {  
 System.out.print("\*");  
 }  
 System.out.println();  
 }  
 }  
  
 \*/  
 //right triangle pattern  
 /\* private void rightStarPattern(){  
 Scanner scanner=new Scanner(System.in);  
 System.out.println("u have enter row");  
 int row=scanner.nextInt();  
 for (int i = 1; i <= row; i++)//1,2,3,4,5  
 {  
 for (int j = row; j > i; j--)//1  
 {  
 System.out.print("\*");  
 }  
 System.out.println();  
 }  
 }  
  
 \*/  
 //reverse a string  
 /\* private void reverseString() {  
 Scanner scanner = new Scanner(System.in);  
 System.out.println("u have enter word");  
 String word = scanner.next();  
  
  
 StringBuffer stringBuffer = new StringBuffer(word);  
 stringBuffer.reverse();  
 System.out.println(stringBuffer);  
  
  
 }  
  
 \*/  
 private void pyramid(){  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.println("u have enter row");  
 int row=scanner.nextInt();  
 for(int i=0;i<row;i++){//0-5  
 for(int j=row-i;j>0;--j){//5-0=5;5-1=4  
 System.*out*.print(" ");  
 }  
 for (int j=0; j<=i; j++ ){  
 System.*out*.print("\* ");  
 }  
 System.*out*.println();  
 }  
  
  
  
 }  
}

**Two dimensional Arrays**

package com.Arrays;  
  
public class TwoDim {  
 public static void main(String[] args) {  
// String variableName[]={"Hello","Hi","How","Are","You"};  
// variableName[1]="sir";  
// for(int i=1;i<variableName.length;i++){  
// System.out.print(variableName[i]+" ");  
 int variableName[][] = {{1, 2, 3, 4}, {5, 6, 7}};  
 System.*out*.println(variableName[1][1]);//6  
 variableName[1][2] = 10;//7 instead 10  
 System.*out*.println(variableName[1][2]);  
 for (int i = 0; i < variableName.length; i++) {//i=0;i<4;++i  
 for (int j = 0; j < variableName[i].length; j++)  
 System.*out*.print(variableName[i][j] + " ");  
 System.*out*.println();  
 }  
 }  
}

**String and String Method:**

A string is a non- primitive data type used in programming.A string is a sequence(collection) of characters and can contain letters, numbers, symbols and even spaces. It must be enclosed in quotation marks for it to be recognized as a string.

Eg:String name=”Shalini”;

String age=”30”

String designation=”Software Trainer”

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

There are two ways to create String object:

1. By string literals{literals means constant or fixed value ,that value does not change for execution time}
2. By new keyword

Using String literals:

String name=”shalini”;

String nameOne=”shalini”;

These value can be stored in string Constant Pool

Using new keyword:

String name=new String(“Shalini”)

These value stored in heap memory

**STRING CONSTANT POOL(SCP)**

**String pool is nothing but a storage area in Java heap where string literals stores. It is also known as String Intern Pool or String Constant Pool.**

String Methods:

1.concat():

* The Java String concat() method concatenates one string to the end of another string.
* The concat() method is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

2.equal():

* String’s equals method compares String with specified object. It checks if both have same character sequence. It is generally used to compare two Strings.

3.split()

* The split() in Java is a method that is used to break a string around the matches of the provided regular expression. The split() in Java is also used to break a string based on the provided string delimiter.

4.subString():

* The Java String class substring() method returns a part of the string.
* We pass beginIndex and endIndex number position in the In other words, the beginIndex starts from 0, whereas the endIndex starts from -1.

4.intern():

* The method intern() creates an exact copy of a String object in the heap memory and stores it in the String constant pool.

5.contains():

* String’s contains method checks if sequence of characters can be found in String. It returns true if character sequence is present in String else return false.

6.charAt():

* It is used to retrieve a character at a specific index within a string.
* The indexing starts from 0, so the first character of the string is at index 0, the second character is at index 1
* Syntax: string.charAt(index)
* Parameters:
* string: The string from which you want to extract the character.
* index: The position of the character you want to retrieve.

7.trim():

* It is used to remove leading and trailing whitespace characters (spaces, tabs, newlines, etc.) from a string.
* It does not modify the original string but returns a new string with the leading and trailing whitespace removed.

**Scanner Class in Java:**

The scanner class is the one of the predefined classes that is used to read the data from the keyboard i.e. at run time

To use the Scanner class, you need a predefined package java.util

You need to create an instance(object) of the Scanner class,which is passed to the System.in(an input stream which is used to read data from the keyboard) in the constructor of Scanner class

The new keyword is used to create an instance (object) of the Scanner class

Scanner scanner= new Scanner(System.in)

**Advantages:**

There is no need to mention the type of data

There is no need for exception handling

**Methods of the Scanner class:**

|  |  |
| --- | --- |
| Data type | Methods |
| Byte | nextByte() |
| Short | nextShort() |
| Int | nextInt() |
| Long | nextLong |
| Float | nextFloat |
| Double | nextDouble |
| String(word) | next() |
| String(sentence) | nextLine() |

Example:

package com.scannerclass;  
  
import java.util.Scanner;  
  
public class ScannerExample {  
 public static void main(String[] args) {  
 *scannerMethods*();  
 }  
 static void scannerMethods(){  
 Scanner scanner=new Scanner(System.*in*);  
  
 System.*out*.println("Enter user name:");  
 String name=scanner.nextLine();  
  
 System.*out*.println("Enter user Age:");  
 int age=scanner.nextInt();  
  
 scanner.nextLine();  
  
 System.*out*.println("Enter user Email:");  
 String email=scanner.nextLine();  
  
 System.*out*.println("user name is:"+name);  
 System.*out*.println("User Age is:"+age);  
 System.*out*.println("User Email is:"+email);  
 }  
 /\*  
 ex.1.write a program to input the principal,rate,time.calculate  
 simpleInterest and final amount  
 formula: simpleinterest:(p\*r\*t)/100  
 amount=simpleinterest+principal  
 \*/  
 /\*  
 2.write a program to input the student name and obtained by five marks  
 calculate average of mark  
 \*/  
}

**Recursion:**

In java, a method that calls itself is known as recursive method and this process is know as recursion

Syntax:

Returntype methodName(){

#code to be executed

methodName();#calling same method

}

**Object oriented program**

**Class And Object Program**

**Class:**

* **Class is a template, It define the data member and data member method**
* **Class is a group of object that share common properties and behaviour(variable and method)**
* **In a class we create multiple object**

**Object:**

**Object is a instance of class**

**Object is real-world entity**

**Object is the combination of state and behaviour**

**State means: variable or attributes**

**Behaviour means: methods or functions**

**For example, we can say ‘Orange’ is an object. Its characteristics are: spherical in shape and color is orange. Its behavior is: juicy and tastes sweet-sour.**

**Student.java**

package com.classobject.example.anotherexample;  
  
public class Student {  
 String studentName;  
 String studentRollNo;  
 String studentDepartment;  
  
 public void studentDetails(String studentName,String studentRollNo,String studentDepartment) {  
 this.studentName = studentName;  
 this.studentRollNo = studentRollNo;  
 this.studentDepartment = studentDepartment;  
 System.*out*.println("Student name is:" + studentName);  
 System.*out*.println("Student Rollno is:" + studentRollNo);  
 System.*out*.println("Student department is:" + studentDepartment);  
 }  
  
 public String studentdetail1(String studentName,String studentRollNo,String studentDepartment) {  
 this.studentName = studentName;  
 this.studentRollNo = studentRollNo;  
 this.studentDepartment = studentDepartment;  
 System.*out*.println("Student name is:" + studentName + "\n" + "Student Rollno is:" + studentRollNo + "\n" + "Student department is:" + studentDepartment);  
 return studentName+studentRollNo+studentDepartment;  
  
 }  
}

**StudentMain.java**

package com.classobject.example.anotherexample;  
  
public class StudentMain {  
 public static void main(String[] args){  
 Student student=new Student();  
 student.studentDetails("shalini","34","ECE");  
 student.studentdetail1("Anchu","23","CSE");  
 System.*out*.println("Another Student details");  
 Student student1=new Student();  
 student1.studentDetails("shalini","34","ECE");  
 student1.studentdetail1("Anchu","23","CSE");  
  
 }  
}

**Encapsulation example program**

* Encapsulation in java is the process of binding related data(variables) and functionality(methods) into a single unit called class.
* Encapsulation can be achieved by using access modifier such as public, private, protected or default, so your class will be safe from unauthorized access by others and will be simple to maintain**.**
* We can create fully encapsulated class by
* Making variables private
* Providing getters and setters methods for the accessing the variables.
* Encapsulation is also termed as data hiding because you are making variables private and variables can be only accessed through public getters and setters.

**Student.java**

package com.encapsulationExample;  
   
public class Student {  
 private String studentName;  
 private int studentRoll\_No;  
 private int studentId;  
  
  
 public String getStudentName() {  
  
 return studentName;  
 }  
  
 public void setStudentName(String studentName) {  
  
 this.studentName = studentName;  
 }  
  
  
 public int getStudentRoll\_No() {  
 return studentRoll\_No;  
 }  
  
 public void setStudentRoll\_No(int studentRoll\_No) {  
 this.studentRoll\_No = studentRoll\_No;  
 }  
  
  
 public int getStudentId() {  
 return studentId;  
 }  
  
 public void setStudentId(int studentId) {  
 this.studentId = studentId;  
 }  
  
  
  
  
}

**StudentMainClass.java**

package com.encapsulationExample;  
  
public class StudentMainClass {  
 public static void main(String[] args){  
 Student student=new Student();  
 //call setter method & write the value of variable  
 student.setStudentName("shalini");  
 student.setStudentRoll\_No(34);  
 student.setStudentId(1234);  
 //call the getter method &read the value of variable print  
 System.*out*.println("Student Name is:"+student.getStudentName()+","+  
 "Student Roll\_No is:"+student.getStudentRoll\_No()+","+"Student Id is:"+student.getStudentId());  
//  
 }  
}

**Abstractraction program**

Abstraction is a concept of exposing only essential details and hiding implementation details.

Abstraction can be achieved with abstract class and interfaces.

Abstract Keyword:is a non -access modifier,used for classes and method

Abstract classes: cannot be used to create object{to access it ,it must be inherited from another class

Abstract method:It can only used in abstract class , and it does not have a body{ie.not define,only declare}

Abstract method is the method which do not have implementation i.e. it does not have any body.

**Abstractmethod and class:**

**Addition.java**

package com.abstractMethodandClass;  
  
public abstract class Addition {  
 abstract int addOfTwoNumber(int number1,int number2);  
 abstract int addOfThreeNumber(int number1,int number2,int number3);  
 public void regularMethod(){  
  
 System.*out*.println("I am regular method");  
 }  
}

**Calculation.java**

package com.abstractMethodandClass;  
  
public class Calculation extends Addition {  
 @Override  
 int addOfTwoNumber(int number1, int number2) {  
  
 return number1+number2;  
 }  
  
  
  
 @Override  
 int addOfThreeNumber(int number1, int number2, int number3) {  
  
 return number1+number2+number3;  
 }  
  
}

**MainClass.java**

package com.abstractMethodandClass;  
  
public class MainClass extends Calculation{  
 public static void main(String[] args){  
 MainClass mainClass=new MainClass();  
 System.*out*.println("Addition of two number is:"+mainClass.addOfTwoNumber(2,3)+  
 "\n"+ "Addition of three number is:"+mainClass.addOfThreeNumber(2,3,4));  
 mainClass.regularMethod();  
 }  
}

**INTERFACE:**

Interface is a 100% of abstract classes.So no need mention for public or abstract specifier.

Eg:void eat();

Void walk();

Variable are declared in final and static keyword,but method are not declared in final and static keyword,whybecause interface method are abstract class .so no need to mention final and static keyword

In interface child class are implemented the two or more interface classes.so multiple inheritance are using interface

Eg:public class child class implements interface class1,interface class 2

**Interface (Multiple inheritance)**

**VehicleOne.java(interface class)**

package com.multipleInterfaceExample;  
  
public interface VehicleOne {  
 int *speed*=80;  
  
 void calculateDistance();  
}

**VehicleTwo.java(interface class)**

package com.multipleInterfaceExample;  
  
public interface VehicleTwo {  
 int *distance*=120;  
 void calculateSpeed();  
}

**Vehicle.java(java class)**

package com.multipleInterfaceExample;  
  
public class Vehicle implements VehicleOne,VehicleTwo{  
 @Override  
 public void calculateDistance() {  
 int calculatedistance=*speed*\*100;  
 System.*out*.println("Vehicle travel distance is:"+calculatedistance);  
  
 }  
  
 @Override  
 public void calculateSpeed() {  
 int calculatespeed=*distance*/12;  
 System.*out*.println("Vehicle travel speed is:"+calculatespeed);  
  
 }  
}

**MultipleInheritanceExample.java**

package com.multipleInterfaceExample;  
  
public class MultipleInheritanceExample {  
 public static void main(String[] args){  
 Vehicle vehicle=new Vehicle();  
 vehicle.calculateDistance();  
 vehicle.calculateSpeed();  
 }  
}

**Inheritance Example program**

**Def: inheritance is one of the core concepts of object-oriented programming language. Copy the data member(field) and data method from existing class to another class**

**Types of Inheritance:**

* **Single Inheritance**
* **Multilevel Inheritance**
* **Hirarchical Inheritance**
* **Hybrid Inheritance**
* **Multiple Inheritance(Java not supported for multiple inheritance it can be applicable for interface concept only)**

**SingleInheritance**

**It can be only one base class and derived class**

**Base class is also known as parent class/super class**

**Derived class is also known as child class/sub class**

package com.singleinheritance.shibinsingleinheritance;  
  
public class Programmer {  
 String id;  
 String name;  
 String age;  
 Programmer(String id,String name,String age){  
 this.id=id;  
 this.name=name;  
 this.age=age;  
 System.*out*.println("Id is:"+id);  
 System.*out*.println("Name is:"+name);  
 System.*out*.println("Age is:"+age);  
  
 }  
}  
class Employee extends Programmer{  
 String salary;  
 Employee(String id,String name,String age,String salary){  
 super(id,name,age);  
 this.salary=salary;  
 System.*out*.println("Salary is"+salary);  
  
 }  
  
}  
class MainClass{  
 public static void main(String[] args) {  
//  
 System.*out*.println("Employee Details");  
 new Employee("12","Shalu","31","3000");  
 }  
}

**Another example(calculate salary):**

package com.singleinheritance.calculatesalary;  
  
public class Employee {  
 float salary;  
 Employee(float salary){  
 this.salary=salary;  
 }  
}  
class Programmer extends Employee{  
 float bonus;  
 Programmer(float salary,float bonus){  
 super(salary);  
 this.bonus=bonus;  
 }  
 float getTotalSalary(){  
 return salary+bonus;  
 }  
}  
class MainClass{  
 public static void main(String[] args) {  
  
 Programmer programmer=new Programmer(2000,1000);  
 System.*out*.println("Total salary of Employee is:"+programmer.getTotalSalary());  
 }  
}

**AnotherExample using private Keyword:**

**Animal.java**

package com.singleinheritance.usingprivatevariable;  
  
//Code reuse is the most important benefit of inheritance because subclasses  
// inherits the variables and methods of superclass.  
//Private members of superclass are not directly accessible to subclass.  
//but it can be indirectly accessible via getter and setter methods.  
  
public class Animal {  
  
 private String food;  
 private int noOfLegs;  
 private boolean vegitarien;  
 public Animal(String food,int noOfLegs,boolean vegitarien){  
 this.food=food;  
 this.noOfLegs=noOfLegs;  
 this.vegitarien=vegitarien;  
 }  
 public String getFood() {  
 return food;  
 }  
  
 public void setFood(String food) {  
 this.food = food;  
 }  
  
 public int getNoOfLegs() {  
 return noOfLegs;  
 }  
  
 public void setNoOfLegs(int noOfLegs) {  
 this.noOfLegs = noOfLegs;  
 }  
  
 public boolean isVegitarien() {  
 return vegitarien;  
 }  
  
 public void setVegitarien(boolean vegitarien) {  
 this.vegitarien = vegitarien;  
 }  
  
}

**Cats.java**

package com.singleinheritance.usingprivatevariable;  
  
public class Cats extends Animal{  
  
  
 private String color;  
 Cats(String food, int noOfLegs, boolean vegitarian, String color){  
 super(food,noOfLegs,vegitarian);  
 this.color=color;  
  
 }  
 public String getColor() {  
 return color;  
 }  
  
 public void setColor(String color) {  
 this.color = color;  
 }  
}

**UsingPrivateVariable.java**

package com.singleinheritance.usingprivatevariable;  
  
public class UsingPrivateVariable {  
 public static void main(String[] args){  
 Cats cat=new Cats("Milk",4,false,"white");  
 System.*out*.println("Cat is vegitarien?:"+cat.isVegitarien());  
 System.*out*.println("Cat eats Food is:"+cat.getFood());  
 System.*out*.println("Number of legs in cat:"+cat.getNoOfLegs());  
 System.*out*.println("Cat Color is:"+cat.getColor());  
 }  
}

**MultilevelInheritance:**

Only one parent class and two or more child class

That means class B extends class A,class c extends class B

**WorkerBaseClass.java**

package com.multilevelInheritance.anotherexample;  
  
public class WorkerBaseClass {  
 String code;  
 String name;  
 long salary;  
 WorkerBaseClass(String code,String name,long salary){  
 this.code=code;  
 this.name=name;  
 this.salary=salary;  
 }  
 void workerDetail(){  
 System.*out*.println("Code is:"+code);  
 System.*out*.println("Name is:"+name);  
 System.*out*.println("Salary is:"+salary);  
 }  
}

**OfficerDerivedClassOne.java**

package com.multilevelInheritance.anotherexample;  
  
public class OfficerDerivedClassOne extends WorkerBaseClass {  
 long hra;  
 OfficerDerivedClassOne(String code,String name,long salary){  
 super(code,name,salary);  
 this.hra=salary\*60/100;  
 }  
 void officerDetail(){  
 super.workerDetail();  
  
 System.*out*.println("HRA is:"+hra);  
 }  
  
}

**ManagerDerivedClassTwo.java**

package com.multilevelInheritance.anotherexample;  
  
public class ManagerDerivedClassTwo extends OfficerDerivedClassOne{  
 long da;  
 ManagerDerivedClassTwo(String code,String name,long salary){  
 super(code,name,salary);  
 this.da=salary\*40/100;  
  
 }  
 void managerDetail(){  
 super.officerDetail();  
 System.*out*.println("Da is:"+da);  
 }  
  
}

**MultilevelInheritance.java**

package com.multilevelInheritance.anotherexample;  
  
public class MultilevelInheritance {  
 public static void main(String[] args){  
 /\* System.out.println("Information of Worker:");  
 WorkerBaseClass workerBaseClass=new WorkerBaseClass("101","shalu",6000);  
 workerBaseClass.workerDetail();  
 System.out.println("..............................");  
  
 System.out.println("Information of Officer:");  
 OfficerDerivedClassOne officerDerivedClassOne=new OfficerDerivedClassOne("102","shalini",4000);  
 officerDerivedClassOne.officerDetail();  
 System.out.println("..............................");  
  
 System.out.println("Information of Manager:");  
 ManagerDerivedClassTwo managerDerivedClassTwo=new ManagerDerivedClassTwo("103","Lini",6000);  
 managerDerivedClassTwo.managerDetail();  
  
 \*/  
  
  
 ManagerDerivedClassTwo managerDerivedClassTwo=new ManagerDerivedClassTwo("101","shalini",6000);  
 managerDerivedClassTwo.workerDetail();  
 managerDerivedClassTwo.officerDetail();  
 managerDerivedClassTwo.managerDetail();  
  
  
  
  
  
  
  
 }  
}

**HirarchicalInheritance:**

Hirarchical inheritance in a java is a type of inheritance in which the same class is inherited by more than one class.

That means, One parent class can be inherits by more than child class

**StudentBaseClass.java**

package com.hierarchicalinheritanceExample.anotherexample;  
  
public class StudentBaseClass {  
 String rollNumber;  
 String name;  
 StudentBaseClass(String rollNumber,String name){  
 this.rollNumber=rollNumber;  
 this.name=name;  
  
 }  
 void studentDetails(){  
 System.*out*.println(" Student Roll Number is:"+rollNumber);  
 System.*out*.println("Student Name is:"+name);  
 }  
}

**ScienceDerivedClassOne.java**

package com.hierarchicalinheritanceExample.anotherexample;  
  
public class ScienceDerivedClassOne extends StudentBaseClass {  
 String physics;  
 String chemistry;  
 ScienceDerivedClassOne(String rollNumber, String name, String physics, String chemistry){  
 super(rollNumber,name);  
 this.physics=physics;  
 this.chemistry=chemistry;  
  
 }  
 void scienceDetail(){  
 super.studentDetails();  
 System.*out*.println("Physics mark is:"+physics);  
 System.*out*.println("Chemistry mark is:"+chemistry);  
 }  
}

**ArtsDerivedClassTwo.java**

package com.hierarchicalinheritanceExample.anotherexample;  
  
public class ArtsDerivedClassTwo extends StudentBaseClass{  
 String history;  
 String geography;  
 ArtsDerivedClassTwo(String rollNumber,String name,String history,String geography){  
 super(rollNumber,name);  
 this.history=history;  
 this.geography=geography;  
  
 }  
 void artsDetails(){  
 super.studentDetails();  
 System.*out*.println("History marks:"+history);  
 System.*out*.println("Geography marks:"+geography);  
 }  
  
}

**HirarchicalInheritance.java**

package com.hierarchicalinheritanceExample.anotherexample;  
  
public class HierarchicalInheritance {  
 public static void main(String[] args){  
 System.*out*.println("Science Student Details:");  
 ScienceDerivedClassOne scienceDerivedClassOne=new ScienceDerivedClassOne("1","shalini","200","198");  
 scienceDerivedClassOne.scienceDetail();  
 System.*out*.println(".......................");  
 System.*out*.println("Arts Student Details:");  
 ArtsDerivedClassTwo artsDerivedClassTwo=new ArtsDerivedClassTwo("2","Shalu","178","200");  
 artsDerivedClassTwo.artsDetails();  
  
  
  
 }  
}

**HybridInheritance ExampleProgram**

In a java ,The hybrid inheritance is the combination of two or more type of inheritance

The main purpose of using hybrid inheritance is modularize the code into well defined class

It also provide the code resusability

The implementation of hybrid inheritance in java using

1. Single and multiple inheritance
2. Multilevel and hierarchical inheritance

**1.SingleMultipleInheritance**

**BaseClassA.java**

package com.hybridinheritance.singlemultipleinheritance;  
  
public class BaseClassA {  
 int numberOne;  
 int numberTwo;  
 void addition(int numberOne,int numberTwo){  
 int resultAdd=numberOne+numberTwo;  
 System.*out*.println("Addition is:"+resultAdd);  
 }  
}

**DerivedClassB.java**

package com.hybridinheritance.singlemultipleinheritance;  
  
import com.hybridinheritance.singlemultipleinheritance.BaseClassA;  
  
public class DerivedClassB extends BaseClassA {  
 void subtraction(int numberOne,int numberTwo){  
 int resultSub=numberOne-numberTwo;  
 System.*out*.println("Subtraction of two number is:"+resultSub);  
  
 }  
  
}

**interfaceclassC(interface class)**

package com.hybridinheritance.singlemultipleinheritance;  
  
public interface InterfaceClassC {  
 public int *numberThree*=70;  
 void multiplication();  
}

**DerivedClassD.java**

package com.hybridinheritance.singlemultipleinheritance;  
  
public class DerivedClassD extends DerivedClassB implements InterfaceClassC {  
  
  
 @Override  
 public void multiplication() {  
 int multiResult=*numberThree*\*100;  
 System.*out*.println("Multiplication of two number is:"+multiResult);  
 }  
  
  
}

**HybridSingleMultipleInheritance.java**

package com.hybridinheritance.singlemultipleinheritance;  
  
import com.hybridinheritance.singlemultipleinheritance.DerivedClassD;  
  
public class HybridSingleMultipleInheritance {  
 public static void main(String[] args){  
 DerivedClassD derivedClassD=new DerivedClassD();  
 derivedClassD.addition(3,6);  
 derivedClassD.subtraction(4,8);  
 derivedClassD.multiplication();  
  
 }  
  
  
}

**MultilevelHirarchicalInheritance**

**MultilevelBaseClass.java**

package com.hybridinheritance.multilevelhierarchical;  
  
public class MultilevelBaseClass {  
 int numberOne;  
 int numberTwo;  
 int addition(int numberOne,int numberTwo){  
  
 return numberOne+numberTwo;  
 }  
}

**MultilevelDerivedClassOne.java**

package com.hybridinheritance.multilevelhierarchical;  
  
public class MultilevelDerivedClassOne extends MultilevelBaseClass{  
 int subtraction(int numberOne,int numberTwo){  
  
 return numberOne-numberTwo;  
 }  
}

**MultilevelDerivedClassTwo.java**

package com.hybridinheritance.multilevelhierarchical;  
  
public class MultilevelDerivedClassTwo extends MultilevelDerivedClassOne {  
 String multiplication(int numberOne,int numberTwo){  
  
 return "Multiplication is"+(numberOne\*numberTwo);  
 }  
}

**HirarchicalClass.java**

package com.hybridinheritance.multilevelhierarchical;  
  
public class HirarchicalClass extends MultilevelDerivedClassTwo {  
 int division(int numberOne,int numberTwo){  
  
 return (numberOne\*numberTwo)/3;  
 }  
}

**MultileveHirarchicalInheritance.java**

package com.hybridinheritance.multilevelhierarchical;  
  
public class MultilevelHierarchicalInheritance {  
 public static void main(String[] args){  
 HirarchicalClass hirarchicalClass=new HirarchicalClass();  
 System.*out*.println(hirarchicalClass.multiplication(4,6));  
 System.*out*.println(hirarchicalClass.division(3,2));  
 System.*out*.println(hirarchicalClass.addition(2,3));  
 System.*out*.println(hirarchicalClass.subtraction(8,4));  
  
 }  
}

**Constructor in java**

* Constructors is used to initialize the object of the class
* Constructor name is same as the class name
* Constructor has no return type
* Whenever the object of class is created using new keyword,it invoke(call) the constructor of that class
* The constructor is automatically gets call ,when an object create for the class

Types of constructors:

* Default constructor
* Non-Arguments Constructor
* Parametrized constructor or Argument constructor
* Copy constructor

Default constructors:

When you do not provide the constructor for your class, JVM will create [default constructor](https://java2blog.com/java-default-constructor/" \t "_blank).It will not be visible to you, JVM will create it automatically while initializing object of the class

**Example Program**

package com.constructorExample.shibinnewconstructor;  
  
  
class EmployeeDefault{  
 String empName;  
 String empAge;  
 String empSalary;  
  
 public String getEmpName() {  
 return empName;  
 }  
  
 public void setEmpName(String empName) {  
 this.empName = empName;  
 }  
  
 public String getEmpAge() {  
 return empAge;  
 }  
  
 public void setEmpAge(String empAge) {  
 this.empAge = empAge;  
 }  
  
 public String getEmpSalary() {  
 return empSalary;  
 }  
  
 public void setEmpSalary(String empSalary) {  
 this.empSalary = empSalary;  
 }  
  
 static void normalMethod(){  
 System.*out*.println("This is normal method");  
 }  
  
 public static void main(String[] args) {  
 EmployeeDefault employeeDefault=new EmployeeDefault();  
 employeeDefault.setEmpName("Sibin Raj");  
 employeeDefault.setEmpAge("19");  
 employeeDefault.setEmpSalary("2000000");  
 System.*out*.println("Employee Name is:"+employeeDefault.getEmpName());  
 System.*out*.println("Employee Age is:"+employeeDefault.getEmpAge());  
 System.*out*.println("Employee Salary is:"+employeeDefault.getEmpSalary());  
 *normalMethod*();  
  
 }  
}

Non-Argument constructors:

Non argument constructor is constructor which you provide explicitly in the class and it does not have any argument. Or Arguments are not passing to the constructor is called Non argument constructors

Example Program:

package com.constructorExample.shibinnewconstructor;  
  
  
class EmployeeDefault{  
 String empName;  
 String empAge;  
 String empSalary;  
  
 public String getEmpName() {  
 return empName;  
 }  
  
 public void setEmpName(String empName) {  
 this.empName = empName;  
 }  
  
 public String getEmpAge() {  
 return empAge;  
 }  
  
 public void setEmpAge(String empAge) {  
 this.empAge = empAge;  
 }  
  
 public String getEmpSalary() {  
 return empSalary;  
 }  
  
 public void setEmpSalary(String empSalary) {  
 this.empSalary = empSalary;  
 }  
 EmployeeDefault(){  
 System.*out*.println("This is a non argument constructor");  
 }  
 static void normalMethod(){  
 System.*out*.println("This is normal method");  
 }  
  
 public static void main(String[] args) {  
 EmployeeDefault employeeDefault=new EmployeeDefault();  
 employeeDefault.setEmpName("Sibin Raj");  
 employeeDefault.setEmpAge("19");  
 employeeDefault.setEmpSalary("2000000");  
 System.*out*.println("Employee Name is:"+employeeDefault.getEmpName());  
 System.*out*.println("Employee Age is:"+employeeDefault.getEmpAge());  
 System.*out*.println("Employee Salary is:"+employeeDefault.getEmpSalary());  
 *normalMethod*();  
  
 }  
}

Parametrized or Argument Constructors:

When you pass arguments to the constructor, this type of constructor is called Parameterized constructor.

package com.constructorExample.shibinnewconstructor;  
  
  
class EmployeeDefault {  
 String empName;  
 String empAge;  
 String empSalary;  
 EmployeeDefault(String empName,String empAge,String empSalary){  
 this.empName=empName;  
 this.empAge=empAge;  
 this.empSalary=empSalary;  
 System.*out*.println("Employee Name is:"+empName);  
 System.*out*.println("Employee Age is:"+empAge);  
 System.*out*.println("Employee Salary is:"+empSalary);  
  
 }  
 public static void main(String[] args) {  
 new EmployeeDefault("shibin","19","2000000");  
 }  
}

Constructor overloading:

* More than one constructor with different parameter in a class is called constructor overloading
* Two constructor in the same class have same parameter ,in this case java complier will generate error
* An overloading constructor must have different parameter.In java compiler deside which constructor has to be called on the number of argument passing with an object

Example Program

package com.constructorExample;  
  
public class Person {  
 private String personName;  
 private int adtherNumber;  
 private String voteIdNumber;  
 private boolean isEligible;  
 Person(String personName,String voteIdNumber){  
 this.personName=personName;  
 this.voteIdNumber=voteIdNumber;  
 System.*out*.println("Person Name is:"+personName+","+"Person VoteIdNumber is:"+voteIdNumber);  
  
 }  
 Person(String personName,int adtherNumber,String voteIdNumber,boolean isEligible){  
 this.personName=personName;  
 this.adtherNumber=adtherNumber;  
 this.voteIdNumber=voteIdNumber;  
 this.isEligible=isEligible;  
 System.*out*.println("Person Name is:"+personName+","  
 +"Person AdtherNumber is:"+adtherNumber+","  
 +"Person VoteIdNumber is:"+voteIdNumber+","  
 + "Person Eligible for Vote:"+isEligible);  
  
  
 }  
  
 public static void main(String[] args) {  
 new Person("sibin Raji","As34567D");  
 new Person("shibin",566,"4567Ind",true);  
 }  
}

**POLYMORPHISM:**

Polymorphism means one name many forms. In Java, polymorphism can be achieved by method overloading and method overriding.

Or Single action in different ways

There are two types of polymorphism in java.

* Compile time polymorphism.
* Run time polymorphism.

Compile time Polymorphism is nothing but method overloading in java. You can define various methods with same name but different method arguments.

Calling or compile the method is depend upon the parameter

Calculation.java

package com.polymorphismExample;  
  
public class Calculation {  
 int numberOne;  
 int numberTwo;  
 int numberThree;  
  
 String name;  
 public void addition(int numberOne,int numberTwo){  
 this.numberOne=numberOne;  
 numberOne=5;  
 // numberOne=n;  
 this.numberTwo=numberTwo;  
 System.*out*.println("Addition of two number is:"+(numberOne+numberTwo));  
  
 }  
 public void addition(double numberOne,double numberTwo,int numberThree){  
 this.numberOne= (int)numberOne;//Type casting that means->assign a  
 // one primitive datatype to another primitive data type  
 this.numberTwo=(int) numberTwo;  
 this.numberThree=numberThree;  
 System.*out*.println("Addition of three number is:"+(numberOne+numberTwo+numberThree));  
  
 }  
 //void ->no return type,String->return type  
 public String addition(int numberOne,String name){  
 this.numberOne=numberOne;  
 this.name=name;  
 System.*out*.println("I am a number:"+(numberOne+name));  
 return numberOne+name;  
  
 }  
//come in error  
// public void addition(int number1,String name){  
// this.number1=number1;  
// this.name=name;  
// System.out.println("I am a number:"+(number1+name));  
// //return number1+name;  
//  
// }  
}

CompileTimePolymorphism.java

package com.polymorphismExample;  
  
public class CompiletimePolymorphismExample {  
 public static void main(String[] args){  
 Calculation calculation=new Calculation();  
  
 // wrong way:System.out.println("Compile time Polymorphism Example:"+(calculation.addition(2,4))+"\n"+(calculation.addition(2,4,5))+"\n"+(calculation.addition(1,"elements")));  
// System.out.println("Compile");  
 //calculation.addition(1,"shal");  
//  
 System.*out*.println("Compile time Polymorphism Example:"+  
 calculation.addition(1,"elements"));  
 calculation.addition(2,3);//2 is not consider,assign the numberOne value is 5;  
 calculation.addition(2,3,4);  
  
 }  
}

AnotherExample:

EmployeeClass.java

package com.polymorphismExample.anotherexampleofcompiletimepoly;  
class EmployeeClass{  
 void printSalary(long salary){  
 System.*out*.println("Salary without bonus:"+salary);  
 }  
 void printSalary(long salary,long bonus){  
 System.*out*.println("Salary with bonus:"+(salary+bonus));  
 }  
}  
public class MethodOverloadingExample {  
 public static void main(String[] args) {  
 EmployeeClass employeeClass=new EmployeeClass();  
 employeeClass.printSalary(2000);  
 employeeClass.printSalary(2000,1000);  
 }  
}

MethodOverloading.java:

package com.polymorphismExample.anotherexampleofcompiletimepoly;  
class EmployeeClass{  
 void printSalary(long salary){  
 System.*out*.println("Salary without bonus:"+salary);  
 }  
 void printSalary(long salary,long bonus){  
 System.*out*.println("Salary with bonus:"+(salary+bonus));  
 }  
}  
public class MethodOverloadingExample {  
 public static void main(String[] args) {  
 EmployeeClass employeeClass=new EmployeeClass();  
 employeeClass.printSalary(2000);  
 employeeClass.printSalary(2000,1000);  
 }  
}

**Runtime Polymorphism**

Runtime Polymorphism is nothing but method overriding in java.If deriverdclass is having same method as base class then it is known as method overriding Or in another word, If subclass provides specific implementation to any method which is present in its one of parents classes then it is known as method overriding.

SamsungMobile.java

package com.polymorphismExample.RunTimePolymorphism;  
  
public class SamsungMobile {//base class  
 String devicename;  
 String version;  
 int devicestorage;  
 double ram;  
 public void mobileDetails(String devicename,String version,int devicestorage,double ram){  
 this.devicename=devicename;  
 this.version=version;  
 this.devicestorage=devicestorage;  
 this.ram=ram;  
 //optional used  
// System.out.println("Mobile Name:"+devicename+  
// "Mobile Version:"+version+  
// "MobileStorage"+devicestorage+  
// "Mobile Ram"+ram);  
  
  
 }}

RedmiMObile.java:

package com.polymorphismExample.RunTimePolymorphism;  
  
public class RedmiMobile extends SamsungMobile {//child class  
 public void mobileDetails(String devicename, String version, int devicestorage, double ram) {  
 this.devicename = devicename;  
 this.version = version;  
 this.devicestorage = devicestorage;  
 this.ram = ram;  
  
 System.*out*.println("Mobile Name:" + devicename +  
 "Mobile Version:" + version +  
 "MobileStorage" + devicestorage +  
 "Mobile Ram" + ram);  
  
 }  
}

RuntimePolymorphism.java:

package com.polymorphismExample.RunTimePolymorphism;  
  
public class RunTimePolymorphismExample {  
 public static void main(String[] args){  
 RedmiMobile redmiMobile=new RedmiMobile();  
 System.*out*.println("Sumsung Mobile Details:");  
 redmiMobile.mobileDetails("Sumsung","Android",42,4.00);  
 System.*out*.println("RedMi Mobile Details:");  
 redmiMobile.mobileDetails("Redmi","Android",50,3.5);  
 System.*out*.println("Apple Mobile Detail:");  
 redmiMobile.mobileDetails("Apple","IOS",128,8);  
  
 }  
  
  
}

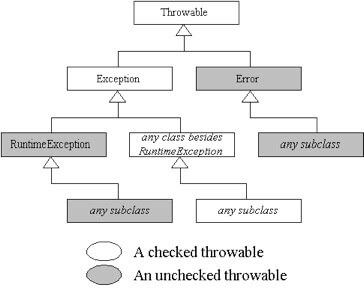
**Limitation Of Runtime PolyMorphism**

* Cannot Override the private method of Parent Class
* Cannot Ovveride Final Method
* Cannot Ovveride static method

**Exception Handling:**

* Java doc says “ An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program’s instructions.”
* The term “exception” means “exceptional condition” and is an occurrence that changes the normal program flow.
* When an exception event occurs in Java , an exception is said to be “thrown”.
* Every Exception will be thrown at runtime.

**Exceptions hierarchy**

* 

**Throwable**

Throwable class is present in java.lang package.

Throwable is the super class for all exception classes in Java.

It has two direct sub classes such as Exception, Error(Of Course every exception class is directly or indirectly sub class of Throwable ).

It has some methods which are to print exceptions details as per application requirement.

**Exception**

Exception is the class present in java.lang package.

This class doesn’t have it’s own methods , it inherited methods from Throwable class.

All sub classes of this class are considered as checked Exceptions(except RuntimeException and it’s sub classes).

**RuntimeException**

RuntimeException is the class present in java.lang package.

It is sub class of Exception class.

This class is also doesn’t have it’s own methods , it inherited methods from Throwable class. Because Throwable is indirectly super class of RuntimeException.

All sub classes of this class are considered as unchecked Exceptions.

**Error**

* Error is the class present in java.lang package.
* It is sub class of Throwable class.
* This class is also doesn’t have it’s own methods , it inherited methods from Throwable class. Because Throwable is indirectly super class of RuntimeException.
* All sub classes of this class are considered as unchecked errors.

**Checked Exceptions**

* If a class is a subclass of Exception class directly or indirectly and should not be subclass of RuntimeException class, it is a checked exception.
* Why the name checked, because these exceptions can be detected at compile time.
* Of course checked exceptions thrown at runtime,It is mandatory that Java code requires to declare or handle them at compile time otherwise the code doesn’t compile .
* These are caused by unexpected conditions outside control of code (e.g. database down, file I/O error, wrong input, etc).
* These are thrown programmatically.
* These are recoverable errors.
* For Example IOException is a checked exception, it occurs when you are try to open a file which is not existed in that location.
* You can recover from this exception by putting the file in the same location.

**Unchecked Exceptions**

* If a class is a subclass of RuntimeException class directly or indirectly, it is a unchecked exception.
* Why the name unchecked , because these are not detected at compile time. So, It is not mandatory that Java code requires to declare or handle them.
* These are also thrown at run time.
* These are thrown by JVM.
* These are also recoverable errors.
* **For Example :** when you call method on reference variablewhich is null causes NullPointerException.
* You can recover from this exception , by checking reference is null or not before method on that reference

**Example Program in Exception(Uunchecked)**

package com.exception.anotherexample;  
  
import java.util.Arrays;  
import java.util.Scanner;  
  
public class ExampleOfUncheckedException {  
 public static void main(String[] args) {  
// arithmeticException();  
// arithmeticExcTryCatchExample();  
// arrayIndexOutOfRangeException();  
// arrayIndexOutOfRangeExceptionTryCatch();  
// classCastException();  
// illegalArgumentException();  
 *numberFormatException*();  
  
 }  
 static void arithmeticException(){  
 Scanner scanner=new Scanner(System.*in*);  
 System.*out*.println("Enter the two numbers");  
 int numberOne=scanner.nextInt();  
 int numberTwo=scanner.nextInt();  
 int result=numberOne/numberTwo;  
 System.*out*.println(result);  
 }  
 //how to handle  
 static void arithmeticExcTryCatchExample(){  
 Scanner scanner=new Scanner(System.*in*);  
 System.*out*.println("Enter the two numbers");  
 int numberOne=scanner.nextInt();  
 int numberTwo=scanner.nextInt();  
 try {  
 int result=numberOne/numberTwo;  
 System.*out*.println(result);  
 }  
 catch (Exception ex){  
 ex.printStackTrace();  
 System.*out*.println("cannot divide by zero");  
  
 }  
  
 }  
 static void arrayIndexOutOfRangeException(){  
 int arrayName []={1,2,3,4,5};  
 System.*out*.println(Arrays.*toString*(arrayName));  
 Scanner scanner=new Scanner(System.*in*);  
 System.*out*.println("Enter the index value");  
 int enterArray=scanner.nextInt();  
 System.*out*.println(arrayName[enterArray]);  
  
 }  
static void arrayIndexOutOfRangeExceptionTryCatch(){  
 try {  
 int arrayName []={1,2,3,4,5};  
 Scanner scanner=new Scanner(System.*in*);  
 System.*out*.println("Enter the index value");  
 int enterArray=scanner.nextInt();  
 System.*out*.println(arrayName[enterArray]);  
 }catch (Exception ex){  
 System.*out*.println("This is a index out of range Exception:"+ex);  
 }  
}  
 static void classCastException(){  
 try {  
 Object obj = "Hello";  
 Integer number = (Integer) obj; // Attempting an incompatible cast  
 System.*out*.println(number);  
 } catch (Exception e) {  
 System.*out*.println("ClassCastException occurred: " + e);  
 }  
 }  
 static void illegalArgumentException(){  
 Scanner scanner = new Scanner(System.*in*);  
  
 try {  
 System.*out*.print("Enter a positive number: ");  
 int number = scanner.nextInt();  
  
 if (number <= 0) {  
 throw new IllegalArgumentException("Number must be positive");  
 }  
  
 System.*out*.println("You entered: " + number);  
 } catch (Exception e) {  
 System.*out*.println("IllegalArgumentException occurred: " + e);  
 }  
 }  
 static void numberFormatException(){  
 String numberString = "123abc"; // Invalid number string  
  
 try {  
 int number = Integer.*parseInt*(numberString);  
 System.*out*.println("Number: " + number);  
 } catch (Exception e) {  
 System.*out*.println("NumberFormatException occurred: " + e);  
 }  
 }  
}  
  
  
  
  
  
  
  
  
  
class Person{  
 String personName;  
  
// public String getPersonName() {  
// return personName;  
// }  
//  
// public void setPersonName(String personName) {  
// this.personName = personName;  
// }  
}  
class NullPointerException{  
 public static void main(String[] args) {  
  
 try {  
 Person person=null;  
 String name= person.personName;  
 person.personName="shalini";  
 System.*out*.println(name);  
 }  
 catch (Exception ex){  
 System.*out*.println("person name is null so null pointer exception occur"+ex);  
 }  
 }  
}

**Java throws and thrown keyword:**

In Java, exceptions can be categorized into two types:

Unchecked Exceptions: They are not checked at compile-time but at run-time.For example: ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, exceptions under Error class, etc.

Checked Exceptions: They are checked at compile-time. For example, IOException,InterruptedException, SqlException etc.

**This Keyword in Java Using Example**

* this keyword in java is used to refer to current object or instance of class.
* The this keyword refers to the current object in a method or constructor. The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).
* It can be used in the constructor to call any other overloaded constructor but this keyword should be the first statement in the constructor.

**Super Keyword in Java Using Example**

super Keyword in java is used to refer the object of the immediate superclass, and it is related to inheritance in java.

Let’s say you have an instance variable or method of same name in subclass and superclass.

**How will JVM know which one are you referring to; superclass or subclass?**

That’s where you can use super keyword to refer superclass’s variables, methods or constructor.

**super keyword in java can be used at three-levels.**

**Usage of super at variable level**

We can refer to variable of super class using: super.variableName

Let’s see this with the help of example.Person class has name variable.Employee class also has same name variable.

You can use super.name to refer to Person’s class name variable, there is no other way to access name variable in subclass.

public **class** Person {

**String** name;

    public Person() {

        System.out.println("Calling Person constructor");

        name = "Default";

    }

}

**class** Employee **extends** Person {

**String** name;

**int** age;

    public Employee() {

        System.out.println("Calling Employee class constructor");

**this**.name = "Martin";

    }

    public **void** workOnAssignment() {

*// Working on assignment*

    }

    public **String** getName() {

**return** name;

    }

    public **void** setName(**String** name) {

**this**.name = name;

    }

    public **int** getAge() {

**return** age;

    }

    public **void** setAge(**int** age) {

**this**.age = age;

    }

    public **void** printName() {

        System.out.println("Printing default name from person class : " + **super**.name);

    }

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

        Employee e1 = **new** Employee();

        e1.printName();

    }

}

OUTPUT:

Calling Person constructor

Calling Employee class constructor

Printing default name from person class : Default

## Usage of super at Constructor level

super keyword can be used to call the [constructor](https://java2blog.com/constructor-java/" \t "_blank) of immediate parent class.

public **class** Person

{

**String** name;

    public Person(**String** name)

    {

**this**.name=name;

        System.out.println("Calling Person Parameterized constructor");

    }

}

**class** Employee **extends** Person{

**int** age;

    public Employee(**String** name)

    {

**super**(name);

        System.out.println("Calling Employee class constructor");

    }

    public **void** workOnAssignment()

    {

*// Working on assignment*

    }

    public **String** getName() {

**return** name;

    }

    public **void** setName(**String** name) {

**this**.name = name;

    }

    public **int** getAge() {

**return** age;

    }

    public **void** setAge(**int** age) {

**this**.age = age;

    }

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

    {

        Employee e1=**new** Employee("John");

        System.out.println("Employee's name:"+e1.getName());

    }

}

OUTPUT:

Calling Person Parameterized constructor  
Calling Employee class constructor  
Employee’s name:John

Usage of super keyword at method level

super keyword can be used to call method of Parent class. It can be used to specifically call method of parent class in case of method overriding.

Let’s see with the help of example:

We have printName() method in Person and Employee class and we will super.printName() to call Person’s printName() method.

public **class** Person

{

**String** name;

    public Person()

    {

        System.out.println("Calling Person constructor");

        name="Default";

    }

    public **void** printName()

    {

        System.out.println("Printing default name from person class : "+**this**.name);

    }

}

**class** Employee **extends** Person{

**String** name;

**int** age;

    public Employee()

    {

        System.out.println("Calling Employee class constructor");

**this**.name="Martin";

    }

    public **void** workOnAssignment()

    {

*// Working on assignment*

    }

    public **String** getName() {

**return** name;

    }

    public **void** setName(**String** name) {

**this**.name = name;

    }

    public **int** getAge() { //OUTPUT:

Calling Person constructor

Calling Employee class constructor

Printing default name from person class : Default

Printing name from Employee class : Marti

**return** age;

    }

 public **void** setAge(**int** age) {

**this**.age = age;

    }

    public **void** printName()

    {

**super**.printName();

        System.out.println("Printing name from Employee class : "+**this**.name);

    }

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

    {

        Employee e1=**new** Employee();

        e1.printName();

    }

}

**Final Keyword in Java Using Example**

Final keyword can be associated with:

* Variable
* method
* class

Final is often used when you want restrict others from doing any changes.

Final Variable:

If you make any variable final then you are not allowed to change its value later.It will be constant.If you try to change value, then compiler will give you error.

**Generics in Java:**

Generics in java was introduced in java 5

Generics specifies the class or method for different types

Generics are used mainly for collection where parameter types for the collection are specified not the base type

Example:

ArrayList a1=new ArrayList();

ArrayList<String> a1=new arrayList();

Base Type Parameter Type

Use Of Generics in java:

1.Provide type safty:

* Arrays are Type safe

String [] s=new String[5]

S[0]=”abc”

S[1]=”bn”

S[2]=3;//error

…

S[6]=”fghj”

* Collection are not type safe

ArrayList a1=new ArratList();

A1.add(“dcf”)

A1.add(3);

2.Reslove Type casting

* Arrays do not require type casting while accessing the element of arrays

String [] s=new String[5]

S[0]=”abc”

S[1]=”bn”

String value=s[1]=bn

* In Collection Type casting is used

ArrayList a1=new ArraytList();

A1.add(“dcf”)

String value=a1.get[0]//error

String value=(String)a1.get[0]

Types Of Generics:

* Generics Class
* Generics Method

Generics in Class:

We can create a class that can be used with any type of data. Such a class is known as Generics Class.

Example Program:

package com.generictypes;  
  
public class GenericClassExample {  
 public static void main(String[] args) {  
 BankAccount<String> obj1= new BankAccount<>("shalini");  
 BankAccount<Integer>obj2= new BankAccount<>(4);  
 obj1.display();  
 obj2.display();  
  
 }  
}  
class BankAccount<T>{  
 T t1;  
 BankAccount(T t1){  
 this.t1=t1;  
 }  
 public void display(){  
 System.*out*.println(t1);  
 }  
}

Generics Method:

Similar to the generics class, we can also create a method that can be used with any type of data. Such a class is known as Generics Method.

Syntax:public<T> void methodName(T t)

Where T-generic type,t-variable

Example Program:

package com.generictypes.genericmethod;  
  
  
  
public class GenericMethodExample {  
 public static void main(String[] args) {  
 BankAccountMethod<String> obj1= new BankAccountMethod<>("shalini");  
 BankAccountMethod<Integer>obj2= new BankAccountMethod<>(4);  
 Integer[] money1={100,20,40,500};  
 Double [] money2={10.45,23.45,100.89};  
 obj1.depositMoney(money1);  
 obj2.depositMoney(money2);  
 }  
}  
class BankAccountMethod<T>{  
 T variableT;  
 BankAccountMethod(T variableT){  
 this.variableT=variableT;  
 }  
 public <T> void depositMoney(T[] variableMoney){  
 for (T i:variableMoney){  
 System.*out*.print(i+" ");  
 }  
 System.*out*.println();  
 }  
}

**Serialization**

Java Provides a mechanism is called Serialization

Serialization means converting an object into a byte stream or sequence of bytes

Eg:

Serializer

Objects

Memory(To store object into memory)

DataBase(To store object into database)

FILE(To store object into file)

A java object is serializable if its, class or any subclass implements java.io.Serializable,interface or any subinterface implements java.io.Externalizable

Entire process in jvm independent {that means:

Once you have run or close or execute the program the object has destroyed}object can be serialized on one platform and deserialized on entirely different platform

If you working on serialization and deserialization the two class are used they are ObjectInputStream and ObjectOutputStream

**SerializationExample**

package com.serializationanddeserialization.serializationexample;  
  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.io.ObjectOutputStream;  
import java.io.Serializable;  
  
public class SerializationExample {  
 public static void main(String[] args) {  
 Employee employee=new Employee();  
 employee.employeeName="shalini";  
 employee.employeeAddress="KollemCode";  
 employee.employeeSalary="40.5K";  
 employee.employeeId="Asd104";  
 try {  
 FileOutputStream fileOutputStream=new FileOutputStream("C:/Users/DELL/Desktop/suleka.txt");  
 ObjectOutputStream objectOutputStream=new ObjectOutputStream(fileOutputStream);  
 objectOutputStream.writeObject(employee);  
 objectOutputStream.close();  
 fileOutputStream.close();  
 System.*out*.println("Serialized data is saved in suleka.txt file");  
  
 }catch (IOException ex){  
 ex.printStackTrace();  
 }  
  
 }  
  
}  
class Employee implements Serializable{  
 public String employeeName;  
 public String employeeAddress;  
 public String employeeSalary;  
 public String employeeId;  
}

DeserializationExample:

package com.serializationanddeserialization.serializationexample;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.io.ObjectInputStream;  
  
public class DeserializationExample {  
 public static void main(String[] args) throws IOException,ClassNotFoundException {  
 Employee employee=null;  
 try{  
 FileInputStream fileInputStream=new FileInputStream("C:/Users/DELL/Desktop/suleka.txt");  
 ObjectInputStream objectInputStream=new ObjectInputStream(fileInputStream);  
 employee=(Employee) objectInputStream.readObject();  
 objectInputStream.close();  
 fileInputStream.close();  
  
 }  
  
 finally{  
 System.*out*.println("Employee Details:");  
 System.*out*.println("Name of employee is:"+employee.employeeName);  
 System.*out*.println("Address of employee is:"+employee.employeeAddress);  
 System.*out*.println("Salary of employee is:"+employee.employeeSalary);  
 System.*out*.println("Id of employee is:"+employee.employeeId);  
 }  
 }  
}

**Java Serialization with Transient Keyword:**

Java transient keyword is used in serialization ,if you define any data member as transient keyword, it will not be serialized

Let us take an example:I have decleared class a Employee,It has three data members id,name,age.if you serialize the object,all the value will be serialized but I don’t want to serialize one value eg:age then we can declare the age data member as transient

package com.serializationanddeserialization.serializationexample.transientkeyword;  
  
import java.io.Serializable;  
  
public class Employee implements Serializable {  
 private int id;  
 private String name;  
 private transient int age;  
 Employee(int id,String name,int age){  
 super();  
 this.id=id;  
 this.name=name;  
 this.age=age;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public void setAge(int age) {  
 this.age = age;  
 }  
}

package com.serializationanddeserialization.serializationexample.transientkeyword;  
  
import java.io.\*;  
  
public class SerializationDemo {  
 public static void main(String[] args) throws IOException {  
 SerializationDemo serializationDemo=new SerializationDemo();  
 serializationDemo.writeEmployeeObject();  
  
 }  
 private void writeEmployeeObject() throws FileNotFoundException , IOException {  
 FileOutputStream fileOutputStream=null;  
 ObjectOutputStream objectOutputStream=null;  
 try{  
 fileOutputStream=new FileOutputStream("lekha.txt");  
 objectOutputStream=new ObjectOutputStream(fileOutputStream);  
 Employee employee=new Employee(101,"sha",30);  
 System.*out*.println("id is"+employee.getId());  
 System.*out*.println("name is"+employee.getName());  
 System.*out*.println("Age is"+employee.getAge());  
 objectOutputStream.writeObject(employee);  
 System.*out*.println("successfully written employee object to the file");  
 }  
 finally {  
 if(objectOutputStream !=null){  
 objectOutputStream.close();  
 }  
  
 }  
 }  
}

package com.serializationanddeserialization.serializationexample.transientkeyword;  
  
import java.io.\*;  
  
public class DeSerializationDemo {  
 public static void main(String[] args) throws IOException, ClassNotFoundException {  
 DeSerializationDemo deSerializationDemo=new DeSerializationDemo();  
 deSerializationDemo.readEmployeeObject();  
  
 }  
 private void readEmployeeObject()throws FileNotFoundException,  
 ClassNotFoundException, IOException{  
 FileInputStream fileInputStream=null;  
 ObjectInputStream objectInputStream=null;  
 try {  
 fileInputStream=new FileInputStream("lekha.txt");  
 objectInputStream=new ObjectInputStream(fileInputStream);  
 Employee employee=(Employee) objectInputStream.readObject();  
 System.*out*.println("id is"+employee.getId());  
 System.*out*.println("name is"+employee.getName());  
 System.*out*.println("Age is"+employee.getAge());  
  
 System.*out*.println("successfully read employee object to the file");  
 }  
 finally {  
 if(objectInputStream !=null){  
 objectInputStream.close();  
 }  
  
 }  
}}

**InstanceOf keyword using java:**

The java instanceOf keyword is used to test whethere the object is an instance of specified type{varible or class or subclass or interface}

Syntax:

Objectreference variable instanceOf(class or interface type)

Eg:

Value instanceOf String

#using variable:

public class UsingVariable {  
 public static void main(String[] args) {  
// usingVariableChecking();  
 }  
 public static void usingVariableChecking(){  
 String name="shalini";  
 boolean result=name instanceof String;  
 System.*out*.println("The result is:"+result);  
 //variable value null means output is false  
 String namenull=null;  
 boolean resultnull=namenull instanceof String;  
 System.*out*.println("The result is:"+resultnull);  
 }  
}

#using class

class Car{  
  
}  
class Vehicle extends Car{  
 public static void main(String[] args) {  
 Car car=new Vehicle();  
 boolean resultClass=car instanceof Vehicle;  
 System.*out*.println("The result is:"+resultClass);  
 }  
}

#using class and interface:

//using class and interface  
interface Pizza{  
 public void back();  
}  
class Vegpizza implements Pizza{  
  
 @Override  
 public void back() {  
 System.*out*.println("Bake veg pizza");  
 }  
}  
class NonVegPizza implements Pizza{  
  
 @Override  
 public void back() {  
 System.*out*.println("Bake Non veg pizza");  
 }  
}  
class InstanceOfKeyword{  
 public static void main(String[] args) {  
 Pizza pizza=new Vegpizza();  
 Pizza nonPizza=new NonVegPizza();  
 *test*(pizza);  
 *test*(nonPizza);  
 }  
 static void test(Pizza pizza){  
 if(pizza instanceof Vegpizza){  
 pizza.back();  
 }  
 if(pizza instanceof NonVegPizza){  
 pizza.back();  
 }  
  
 }  
}

Object Cloning in Java:

* The object cloning is a way to create exact copy of an object. The clone() method of object class is used to clone an object
* The java.lang,Cloneable interface must be implemented by the class whose object clone we want to create
* If we don’t implement Cloneable interface clone() method generate CloneNotSupportException
* The clone() method is defined in the object class

Syntax:

Protected(access specifier) objectclone()throws CloneNotsupportExecption

Why use clone() method?

* The clone() method saves the extra processing task for creating the exact copy of an object
* If we perform it by using the new keyword,it will take a lot of processing time to performed that is why we use object cloning

Note:We have not implementes the Cloneable interface in the student class ,cloneNotSupportedException is come in programe

Advantages of object cloning

* You have don’t need to write lengthy and repetive code
* It is the easiest and most efficient way to copy the object
* Clone() is the fastest way to copy arrays

package com.cloneableobject;  
  
public class Student implements Cloneable{  
 int studentId;  
 String studentName;  
 Student(int studentId,String studentName){  
 this.studentId=studentId;  
 this.studentName=studentName;  
 }  
 public Object clone() throws CloneNotSupportedException{  
 return super.clone();  
 }  
  
 public static void main(String[] args) {  
 try {  
 Student studentOne=new Student(101,"shalini");  
 Student studentTwo=(Student)studentOne.clone();  
 System.*out*.println(studentOne.studentId+" "+studentOne.studentName);  
 System.*out*.println(studentTwo.studentId+" "+studentTwo.studentName);  
  
 }  
 catch (CloneNotSupportedException c){  
 c.printStackTrace();  
 }  
 }  
}

Collection in Java:

Iterable is the super interface of collection

Collection is a object that represent a group of object

Collection is the sub interface of itreable interface

Collection is super interface of list,queue,set

**LIST:**

List is the sub interface of collection interface

List can be maintain the Ordered

The Implement classes of the List Interface are

ArrayList.LinkedList,Vector,Stack

Duplicate value can be stored in list interface

List interface includes the methods are:add(),addAll(),get(),set(),remove() etc

**ArrayList:**

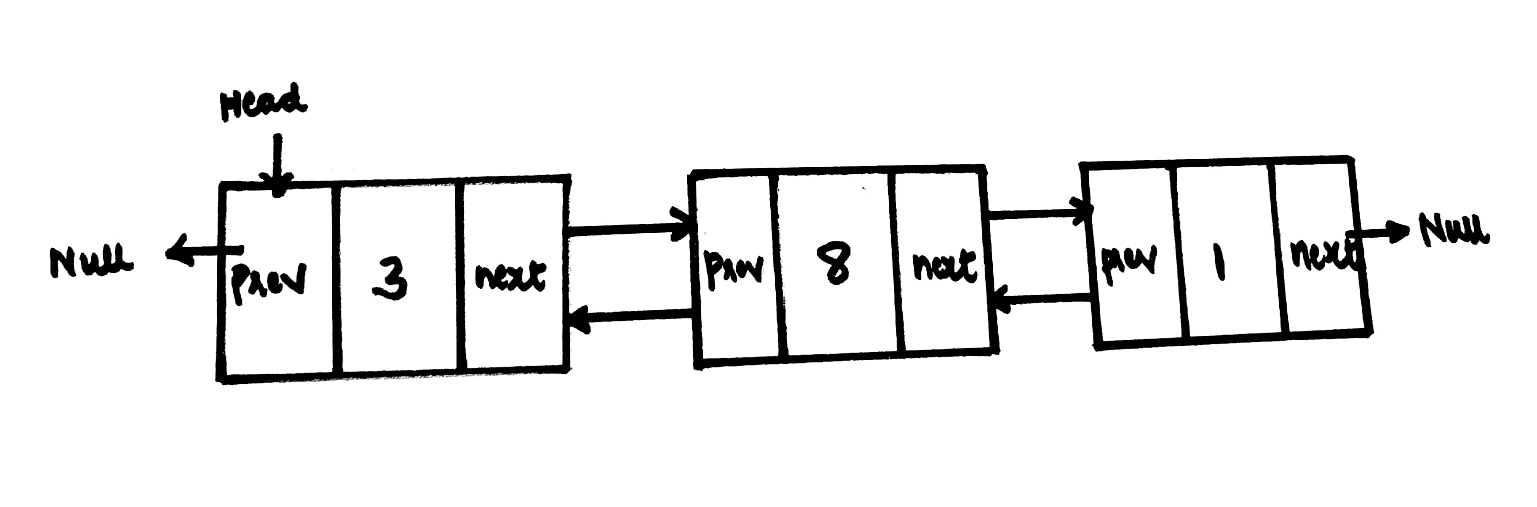
* ArrayList is implementation of list interface.
* ArrayList is not synchonized(so not thread safe)
* ArrayList is implemented using array as internal data structure.It can be dynamically resized .
* ArrayList increases half of its size when its size is increased.
* ArrayList is the sub class of List Interface
* Duplicate elements are allows in arrayList

#### **Why to choose ArrayList vs Array:**

1. Array is fixed length data structure If array is full , you can not add element to it, where as ArrayList in java can dynamically grow and shrink as per our need.
2. You can use generics with ArrayList but not with Array
3. ArrayList have predefined methods which can be used to perform operations.
4. private void implemetArrayListClass() {  
    ArrayList<String> studentlist = new ArrayList<String>();  
    studentlist.add(null);  
    studentlist.add("Arun");  
    studentlist.add("Asha");  
    studentlist.add("Anu");  
    studentlist.add("Hanna");  
    studentlist.add(null);  
    System.*out*.println(studentlist);  
    //Another list creation  
    List<String> studentlistone = new ArrayList<String>();  
    studentlistone.add("shalu");  
    studentlistone.add("shalini");  
    studentlistone.add("lini");  
    studentlistone.add("shal");  
    System.*out*.println(studentlistone);  
    //addAll()->using add the elements one to another  
    studentlist.addAll(studentlistone);  
    System.*out*.println(studentlist);  
    //get() method returns the element present in specified index position.  
    System.*out*.println(studentlist.get(4));//hanna  
    //set() used to replace the element at the specified  
    // position in this list with the specified element.  
    System.*out*.println(studentlist.set(2, "Aric"));  
    System.*out*.println(studentlist);  
    //size() return the length of the list  
    System.*out*.println(studentlist.size());  
    //toArray() convert a list into an array  
    // Create a new array of String type  
    // size of array is same as the ArrayList  
    String[] arr = new String[studentlistone.size()];  
     
    // Convert ArrayList into an array  
    studentlistone.toArray(arr);  
    // print all elements of the array  
    System.*out*.print("Array are: ");  
    //for each loop syntax....for(data type item:array)  
    for (String item : arr) {  
    System.*out*.print(item + ", ");  
    }  
    System.*out*.println();  
    //contains() used to checking if the specified element is exist in the given list or not  
    System.*out*.println("contains method");  
    if (studentlistone.contains("shalini")) {  
    System.*out*.println("shalini present in list");  
    } else if (studentlistone.contains("Ammu")) {  
    System.*out*.println("not present in studentlistone");  
    } else {  
    System.*out*.println("shalini not present in list");  
    }  
    //iterator()used to retriving object one by one from collection.  
    //it is a forwared direction,not backword direction  
    Iterator<String> iterator = studentlistone.iterator();  
    while (iterator.hasNext()) {//hasNext() used toThe hasNext() method helps us to  
    // find the last element of the List. It checks if the List has the next element or not.  
    // If the hasNext() method gets the element during traversing in the forward direction,  
    // returns true, else returns false and terminate the execution.  
    System.*out*.print(iterator.next());//The next() method perform the iteration  
    // in forward order. It returns the next element in the List  
    }  
    System.*out*.println();  
    //clone() used for copy the list(return to the object so typecasting is used)  
    ArrayList studentlistThree = (ArrayList) studentlist.clone();  
    System.*out*.println("cloned object:" + studentlistThree);  
    //remove method()  
    String removestring = studentlistone.remove(1);  
    System.*out*.println("Remove the elements:" + removestring);  
    studentlist.removeAll(studentlist);  
    System.*out*.println("Remove all the elements:" + studentlist);  
    //sublist()  
    System.*out*.println("sublist" + studentlistone.subList(0, 3));  
    //for each loop  
    /\* ArrayList<String> studentlist = new ArrayList<String>();  
    studentlist.add("Arun");  
    studentlist.add("Asha");  
    studentlist.add("Anu");  
    studentlist.add("Hanna");  
    System.out.println(studentlist);  
    for(String list:studentlist){  
    System.out.println(list);  
    }  
    //java 8 lampda expression  
    studentlist.foreach(list->System.out.println(list));  
    //method reference java 8  
    studentlist.forEach(System.out::println);  
   }\*/  
     
     
   }

**LinkedList:**

A **Linked List** is a **linear, dynamic** Data Structure that stores**homogenous** data(data of the same type) following a sequence in **non-contiguous** memory locations. Each value in the List is represented as a **Node** that holds the **reference** to its next consecutive Node in a different memory location.



This can be traversed from the first node of the list to the last node of the list and vice versa, and this is called Doubly Linked list.

The above figure demonstrates a doubly linked list.

Data is the data stored in the node and each node consists of two pointers namely the previous pointer and the next pointer.

The previous pointer points, as the name suggests to the previous node that is part of the list.

The pointer after the current one, points to the next node on the list.

### **Using Object[] array of Linked Lists in Java**

We can create an Array of Linked Lists by using the **Object[]** type array.

***Important points to note:***

* We will create **linkedList objects** and initialize each list with dummy values using the **add()** method.
* We will then create an **Object** class-type array and initialize the array with the lists.
* As the **Object** class is the direct parent to all Java classes, it will readily accept the Linked Lists as input.
* Then, we print each List in the **Object[]** array using the **toString()** method.

**Note:**The advantage of using Object[] arr is that it allows us to store Linked Lists having elements of any data type as the Object class is a parent to all classes.

package com.collection. package com.collection.example.linkedlistnew;  
  
import java.util.ArrayList;  
import java.util.LinkedList;  
  
public class LinkedListExampleProgram {  
 public static void main(String[] args) {  
// usingObjectArrayOfLinkedList();  
 *arrayListOfLinkedList*();  
  
 }  
 static void usingObjectArrayOfLinkedList(){  
 // We create a Linked List of String type.  
 LinkedList<String> listOne = new LinkedList<String>();  
 listOne.add("Apple");  
 listOne.add("Mango");  
 listOne.add("Orange");  
  
 LinkedList<String> listTwo = new LinkedList<String>();  
 listTwo.add("Banana");  
 listTwo.add("Guva");  
  
 // We create a Linked List of Integer type.  
 LinkedList<Integer> listThree = new LinkedList<Integer>();  
 listThree.add(1);  
 listThree.add(2);  
 listThree.add(3);  
 // We can add all types of list into Object array.  
 Object[] array = {listOne, listTwo, listThree};  
  
 System.*out*.println("The array of Linked Lists using Object[] array:");  
// System.out.println();  
 for(Object listobj : array)  
 {  
 System.*out*.println(listobj.toString());  
 }  
 }  
 static void arrayListOfLinkedList(){  
 // We create a Linked List of String type.  
 LinkedList<String> listOne = new LinkedList<String>();  
 listOne.add("milk");  
 listOne.add("Rice");  
  
 LinkedList<String> listTwo = new LinkedList<String>();  
 listTwo.add("Pencil");  
 listTwo.add("pen");  
  
 ArrayList<LinkedList> arrayList = new ArrayList<LinkedList>();  
 // we add the Linked Lists  
 arrayList.add(listOne);  
 arrayList.add(listTwo);  
 System.*out*.println("The ArrayList of Linked List is: ");  
 // Use the for each method to print each Linked list  
 arrayList.forEach(System.*out*::println);  
 }  
  
}

**Another Example:**

public void implementLinkedListClass(){  
 LinkedList<String> studentlist = new LinkedList<>();  
 studentlist.add(null);  
 studentlist.add("Arun");  
 studentlist.add("Asha");  
 studentlist.add("Anu");  
 studentlist.add("Hanna");  
 studentlist.add(null);  
 System.*out*.println(studentlist);  
 //offer() used to element adds in the last position  
 studentlist.offer("Harini");  
 System.*out*.println("add the element in last using offer():"+studentlist);  
 //offerFirst()  
 studentlist.offerFirst("John");  
 System.*out*.println("add the element in list using offerFirst():"+studentlist);  
 //offerLast()  
 studentlist.offerLast("Tamil");  
 System.*out*.println("add the element in last using offerLast():"+studentlist);  
 //peek()This method retrieves the elements, but does not remove  
 System.*out*.println(" the element in last using peek():"+studentlist.peek());  
 System.*out*.println("the element in last using peekFirst():"+studentlist.peekFirst());  
 System.*out*.println(" the element in last using peekLast():"+studentlist.peekLast());  
 //poll()This method retrieves and removes the list of element  
 System.*out*.println(" the element in last using poll():"+studentlist.poll());  
 System.*out*.println(studentlist);  
 System.*out*.println(" the element in last using pollFirst():"+studentlist.pollFirst());  
 System.*out*.println(studentlist);  
 System.*out*.println(" the element in last using pollLase():"+studentlist.pollLast());  
 System.*out*.println(studentlist);  
 //Arraylist method are same as LinkedList  
  
 }

**Set:**

Set interface is the sub interface of Collection interface

It represents the unordered set of elements which doesn't allow us to store the duplicate items.

We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

**MULTITHREADING IN JAVA**

Thread can be called a lightweight process(ie,small process and it want take much time execute)

Java supports multithreading.So it allows your application to perform two or more tasks concurrently

{ie,multiple tasks can be executed concurrently}

Whenever we call main method in java ,it actually creates a single main thread

If we want to create more threads to executes task concurrently,we can use multithreading

Thread can be created in Two ways

1.By extending thread class

2.By implementing Runnable Interface

BY EXTENDING THREAD CLASS:

You can create your own thread by extending thread class and override run method.

You need to create object of that class and call start() method on it to execute thread as different threads

#Normally call the method:

package com.multithreading;  
class Hi{  
 public void show(){  
 for(int i=1;i<5;i++){  
 System.*out*.println("hi");  
 }  
 }  
}  
class Hello{  
 public void show(){  
 for(int i=1;i<5;i++){  
 System.*out*.println("hello");  
 }  
 }  
}  
  
public class ThreadDemo {  
 public static void main(String[] args) {  
 Hi hi=new Hi();  
 Hello hello=new Hello();  
 hi.show();  
 hello.show();  
  
 }  
}

#try catch using thread sleep()

package com.multithreading;  
class HiOne {  
 public void show(){  
 for(int i=1;i<5;i++){  
 System.*out*.println("hi");  
 try{  
 Thread.*sleep*(500);  
 }  
 catch (Exception e){e.printStackTrace();}  
 }  
 }  
}  
class HelloOne{  
 public void show(){  
 for(int i=1;i<5;i++){  
 System.*out*.println("hello");  
 try{Thread.*sleep*(500);}  
 catch (Exception e){e.printStackTrace();}  
 }  
 }  
}  
public class ThreadTryCatch {  
 public static void main(String[] args) {  
 HiOne hiOne=new HiOne();  
 HelloOne helloOne=new HelloOne();  
 hiOne.show();  
 helloOne.show();  
 }  
}

#extends thread class:

package com.multithreading;  
class HiOneEx extends Thread{  
// public void show(){  
 public void run(){  
 for(int i=1;i<5;i++){  
 System.*out*.println("hi");  
 try{Thread.*sleep*(500);} catch (Exception e){e.printStackTrace();}  
 }  
 }  
}  
class HelloOneEX extends Thread{  
// public void show(){  
 public void run(){  
 for(int i=1;i<5;i++){  
 System.*out*.println("hello");  
 try{Thread.*sleep*(500);} catch (Exception e){e.printStackTrace();}  
 }  
 }  
}  
public class ThreadExtend {  
 public static void main(String[] args) {  
 HiOneEx hiOneEx=new HiOneEx();  
// hiOneEx.show();  
 HelloOneEX helloOneEX=new HelloOneEX();  
// helloOneEX.show();  
 hiOneEx.start();  
 try{Thread.*sleep*(10);

} catch (Exception e){

e.printStackTrace();}  
 helloOneEX.start();  
  
 }  
}

**By implementing Runnable Interface:**

You need to implement Runnable interface and override public void run() method

You need to instantiate the class(instance class),pass created object to Thread Constructor and call start() method

package com.multithreading;  
class HiOneExRun implements Runnable{  
  
 public void run(){  
 for(int i=1;i<5;i++){  
 System.*out*.println("hi");  
 try{Thread.*sleep*(500);}  
 catch (Exception e){e.printStackTrace();}  
 }  
 }  
}  
class HelloOneEXRun implements Runnable{  
  
 public void run(){  
 for(int i=1;i<5;i++){  
 System.*out*.println("hello");  
 try{Thread.*sleep*(500);}  
 catch (Exception e){e.printStackTrace();}  
 }  
 }  
}  
public class RunnableInterface {  
 public static void main(String[] args) {  
 Runnable hiOneExRun=new HiOneExRun();  
 Runnable helloOneEXRun=new HelloOneEXRun();  
 Thread t1=new Thread(hiOneExRun);  
 Thread t2=new Thread(helloOneEXRun);  
 t1.start();  
 try{Thread.*sleep*(10);} catch (Exception e){e.printStackTrace();}  
 t2.start();  
  
 }  
}

**Multithreading in Synchronized Keyword:**

**Step1:**

class Counter{  
 int count;  
 public void increment(){  
 count++;  
 }  
}  
public class SynnDemo {  
 public static void main(String[] args) {  
 Counter c=new Counter();  
 c.increment();  
 c.increment();  
 System.*out*.println("Count is:"+c.count);  
 }  
}

**step2:**

**#call increment in 1000times:**

class Counter{  
 int count;  
 public void increment(){  
 count++;  
 }  
}  
public class SynnDemo {  
 public static void main(String[] args)throws Exception {  
 Counter c=new Counter();  
 Thread t1=new Thread(new Runnable() {  
 @Override  
 public void run() {  
 for(int i=1;i<=1000;i++){  
 c.increment();  
 }  
 }  
 });  
 t1.start();  
 t1.join();/\* join means wait for a thread to terminate\*/  
 System.*out*.println("count is:"+c.count);  
  
 }  
}

**step 3:**

**#call increment in 2000times:**

class Counter{  
 int count;  
 public void increment(){  
 count++;  
 }  
}  
public class SynnDemo {  
 public static void main(String[] args)throws Exception {  
 Counter c=new Counter();  
 Thread t1=new Thread(new Runnable() {  
 @Override  
 public void run() {  
 for(int i=1;i<=1000;i++){  
 c.increment();  
 }  
 }  
 });  
 Thread t2=new Thread(new Runnable() {  
 @Override  
 public void run() {  
 for(int i=1;i<=1000;i++){  
 c.increment();  
 }  
 }  
 });  
 t1.start();  
 t2.start();  
 t1.join();  
  
 t2.join();  
 System.*out*.println("count is:"+c.count);  
  
 }  
}

**In this above program the output come in random order**

Note:t1 and t2 threads are using increment method in same time so that can come in issues

1.If t1 is executing increment method t2 is should not be executed

2.If t2 is e executing increment method t1 is should not be executed

3.If you want only one thread work in this increment method we can achieved the method is called Synchronized

4.we can using synchronized keyword only one thread work in increment method.ie:t1 is working increment method t2 is wait, then t2 is work t1 is wait

class Counter{  
 int count;  
  
 public synchronized void increment(){  
 count++;  
 }  
}  
public class SynnDemo {  
 public static void main(String[] args)throws Exception {  
 Counter c=new Counter();  
 Thread t1=new Thread(new Runnable() {  
 @Override  
 public void run() {  
 for(int i=1;i<=1000;i++){  
 c.increment();  
 }  
 }  
 });  
 Thread t2=new Thread(new Runnable() {  
 **@Override  
 public void run() {  
 for(int i=1;i<=1000;i++){  
 c.increment();  
 }  
 }  
 });  
 t1.start();  
 t2.start();  
 t1.join();  
  
 t2.join();  
 System.*out*.println("count is:"+c.count);  
  
 }  
}**

**Java 8 Features**

**Lamda Expression:**

Java 8 introduced lambda expression

The expression through which we can represent at Anonymous function

Anonymous means Nameless or unknow function

ie means the method we don’t have any name or access specifier

Syntax:

Parameter Expression body

() -> sout(“lambda expression”);

For example:

Class AnonymousFunction{

Main(){

}

Public static void NormalMethod(){

Sout(“This is normal method”);

}

() ->{

Sout(“This anonymous function”);

}}

Lamda expression can be applicable for functional interface

The method which is present in the functional interface for that particular method only write in the lamda expression{you know lambda expression first should know functional interface}

**Functional interface:**

The interface who contains only one abstract method but can have multiple default and static method is called functional interface

Example:

Runnable ->run()

Callable->call()

Comparable->compareTo()

Comparator->compare()

Example:

public interface FunctionalInterfaceOne {  
 public static void main(String[] args) {  
 FunctionalInterfaceOne functionalInterfaceOne=new FunctionalInterfaceOne() {  
 @Override  
 public void methodOne() {  
 System.*out*.println("abstract method");  
 }  
 @Override  
 public void defaultMethodOne(){  
 FunctionalInterfaceOne.super.defaultMethodOne();  
 }  
 @Override  
 public void defaultMethodTwo(){  
 FunctionalInterfaceOne.super.defaultMethodTwo();  
 }  
 };  
 functionalInterfaceOne.methodOne();  
 functionalInterfaceOne.defaultMethodOne();  
 functionalInterfaceOne.defaultMethodTwo();  
 *staticMethodOne*();  
 *staticMethodTwo*();  
  
 }  
 void methodOne();//abstract method  
 default void defaultMethodOne(){  
 System.*out*.println("Default method One");  
 }  
 default void defaultMethodTwo(){  
 System.*out*.println("Default method Two");  
 }  
 static void staticMethodOne(){  
 System.*out*.println("static method One");  
 }  
 static void staticMethodTwo(){  
 System.*out*.println("static method Two");  
 }  
  
}

**How to represented lambda Expression and functional interface:**

**#interface calculator**

package com.java8;  
  
public interface Calculator {  
 void switchOn();  
}

**#implements (functional interface)**

package com.java8;  
  
public class CalculatorusingFunInterface implements Calculator{  
 @Override  
 public void switchOn() {  
 System.*out*.println("Switch is one");  
 }  
  
 public static void main(String[] args) {  
 CalculatorusingFunInterface obj=new CalculatorusingFunInterface();  
 obj.switchOn();  
 }  
}

**#implements using Lambda expression**

public interface Calcu {  
 void onSwitch();  
}  
class CalcuImpFun {  
 public static void main(String[] args) {  
 Calcu calcuobj=()->{  
 System.*out*.println("Switch is on");  
 };  
 calcuobj.onSwitch();  
 }  
  
  
}

**Various way to used lambda Expression:**

**#1.In method we have only one statement print,no need to the curly braces**

package com.java8;  
  
public class CalculatorUsingLambdaExpression {  
 public static void main(String[] args) {  
 Calculator calculator=()->  
 System.*out*.println("Switch is On");  
 calculator.switchOn();  
 }  
}

**#2.One argument passing**

public interface Calcu {  
void sum(String input);  
}  
class CalcuImpFun {  
 public static void main(String[] args) {  
 Calcu calcuobj=(input)->{  
 System.*out*.println("Hello"+" "+input);  
 };  
 calcuobj.sum("shalini");  
 }

**#3.Two argument passing:**

public interface Calcu {  
void sum(int a,int b);  
}  
class CalcuImpFun {  
 public static void main(String[] args) {  
 Calcu calcuobj=(a,b)->{  
 System.*out*.println("Addition of two number is:"+(a+b));  
 };  
 calcuobj.sum(3,6);  
 }

**4.apply business logic**

public interface Calcu {  
void sub(int a,int b);  
}  
class CalcuImpFun {  
 public static void main(String[] args) {  
 Calcu calcuobj=(a,b)->{  
 if(a>b){  
 throw new RuntimeException("message");  
 }  
 else {  
 System.*out*.println("sub of two number is:"+(a-b));  
 }  
 };  
 calcuobj.sub(3,6);  
 }  
  
  
}

**Predefined Functional interface:**

Predefined functional interface are:

* Consumer Interface
* Predicate Interface
* Supplier Interface

**Consumer functional interface**:

Consumer<T> is an built in functional interface introduced in java 8

The consumer interface has only one single method is called accept().It accept a single argument of any data type and does not return any result

Syntax:void accept(T t);

Where:

Accept() – consumer method

T- Generic Type

t – input or argument

Example:

# using Lambda Expression

package com.java8.functionalinterface.c.p.s;  
  
import java.util.Arrays;  
import java.util.List;  
import java.util.function.Consumer;  
  
public class ConsumerLambdaExample {  
 public static void main(String[] args) {  
 Consumer<Integer>consumer=t-> System.*out*.println("The value of t is"+t);  
 consumer.accept(10);  
  
//........create a list using forEach method{forEach is a consumer interface}  
 List<String> fruitlist= Arrays.*asList*("apple","mango","Banana");  
 fruitlist.stream().forEach(t-> System.*out*.println("The fruitsList are:"+fruitlist));  
}}

#using functional interface

package com.java8.functionalinterface.c.p.s;  
  
import java.util.function.Consumer;  
  
public class ConsumerImplExample implements Consumer<Integer> {  
 @Override  
 public void accept(Integer t) {  
 System.*out*.println("The value of t is:"+t);  
  
 }  
  
 public static void main(String[] args) {  
 /\* ConsumerImplExample consumerExample=new ConsumerImplExample();  
 consumerExample.accept(10);\*/  
 //another calling  
 Consumer<Integer>consumer=new ConsumerImplExample();  
 consumer.accept(10);  
  
 }  
  
  
}  
//void accept(T t)

Predicate Functional interface:

It is used for conditional check

It has only one single method is called test()

It may be true or false depending on the value of its variable

Syntax:

Boolean test(T t);

Example

# using lambda expression:

package com.java8.functionalinterface.c.p.s;  
  
import java.util.function.Predicate;  
  
public class PredicateLambdaEx {  
 public static void main(String[] args) {  
 Predicate<Integer> predicate=t->{  
 if(t%2==0){  
 System.*out*.println("the number is even");  
 return true;  
 }  
 else {  
 System.*out*.println("The number is odd");  
 return false;  
 }  
  
 };  
 System.*out*.println(predicate.test(4));  
  
 }  
}

#using functional interface

package com.java8.functionalinterface.c.p.s;  
  
import java.util.function.Predicate;  
  
public class PredicateImpl implements Predicate<Integer> {  
 public boolean test(Integer t){  
 if(t%2==0){  
 System.*out*.println("The number is even number");  
 return true;  
 }  
 else {  
 System.*out*.println("the number is odd number");  
 return false;  
 }  
 }  
  
 public static void main(String[] args) {  
 Predicate<Integer> predicate=new PredicateImpl();  
 System.*out*.println(predicate.test(4));  
 }  
}

#using filter method:

package com.java8.functionalinterface.c.p.s;  
  
import java.util.Arrays;  
import java.util.List;  
import java.util.function.Predicate;  
  
public class PredicateFilterMethod {  
 public static void main(String[] args) {  
 Predicate<Integer>predicate=t->t%2==0;  
 System.*out*.println(predicate.test(4));  
 List<Integer>listOne= Arrays.*asList*(1,2,3,4,5);  
 listOne.stream().filter(predicate).forEach(t-> System.*out*.println("print the number"+t));  
 }  
}

Supplier Functional interface:

Supplier can be used in all context where, there is no input but an output expected

Syntax:

T get();

(Or)

Which does not take any argument and produce result of type

It has a functional method called T get()

Example:

#using functional interface:

package com.java8.functionalinterface.c.p.s;  
  
import java.util.function.Supplier;  
  
public class SupplierImpl implements Supplier<String> {  
 @Override  
 public String get() {  
 return "Hi How are uu";  
 }  
  
 public static void main(String[] args) {  
 Supplier<String> supplierObj=new SupplierImpl() ;  
 System.*out*.println(supplierObj.get());  
  
  
 }  
}

#using lambda expression:

package com.java8.functionalinterface.c.p.s;  
  
import java.util.Arrays;  
import java.util.List;  
import java.util.function.Supplier;  
  
public class SupplierLambda {  
 public static void main(String[] args) {  
 Supplier<String> supplierObj=()->"Hi";  
 System.*out*.println(supplierObj.get());  
// .......using list........  
 List<String> listObj= Arrays.*asList*("a","b","c");  
 System.*out*.println(listObj.stream().findAny()  
 .orElseGet(supplierObj));  
  
 }  
}  
//T get()

**Stream Api**

* Stream means flow of data or sequence of data element
* Stream api is used to process of collection of object
* Stream is a sequence of objects that support various methods which can be pipelined to produce the desired result
* A Stream is not a data structure instead it takes input from the collections, Arrays, I/O channels
* Streams don’t change the original data structure ,they only provide the result as per the pipelined

What is stream pipeline in java 8?

* Source of stream-Collections/IO channels/File
* Intermediate/Non – terminal operations
* Terminal operations

Operation 3

Operation 1

Operation 2

Non terminal operations

Stream pipeline

**Terminal operation**

**Source of stream**

Example:

public class PipeLineMethod {  
 public static void main(String[] args) {  
 //stream pipeline flow  
 //1.source of data  
 List<Integer>integerList= Arrays.*asList*(1,2,3,4,5,6);  
 //2.convert source into a java stream  
 Stream<Integer>integerStream=integerList.stream();  
 //3.non terminal operatin  
 Stream<Integer>fileStream=integerStream.filter(ele->ele>3);  
 //4.\non terminal operatin  
 Stream<Integer>computedStream=fileStream.map(n->n\*n);  
 //5.terminal operation  
 computedStream.forEach(a-> System.*out*.println(a));  
 }  
}

Features of Java 8 stream

* Stream does’t store data ,it operates on the source data structure
* Operation performed on a stream does not modify its source
* Non terminal/intermediate operations generate another stream
* Stream operation are functional interface, easy to lambda expression
* Element of a stream are only visited one during a life of a stream

Why we need Stream:

* Functional Programming (i.e. If we have use functional interface, we can represented by lambda expression )
* Code Reused(ie. Length of code will be less)
* Bulk Operation(ie. We have used for collections, that perform Stream in api)

**Method of Stream:**

**Two types of method:**

* **Filter -> used for conditional check**
* **ForEach -> used for iteration**

**Example of forEach Method:**

package com.java8.functionalinterface.c.p.s.stream.foreachmethod.filtermethod;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
import java.util.function.Consumer;  
  
public class ForEachMethod {  
 public static void main(String[] args) {  
 List<String>list=new ArrayList<>();  
 list.add("apple");  
 list.add("Banana");  
 list.add("Mango");  
 list.add("Grapes");  
// normal method  
 for(String itemobj:list){  
 System.*out*.println(itemobj);  
 }  
// using stream  
 list.stream().forEach(t-> System.*out*.println(t));  
 ////map/////  
 Map<Integer,String> map=new HashMap<>();  
 map.put(1,"apple");  
 map.put(2,"banana");  
 map.put(3,"mango");  
 map.put(4,"grapes");  
 //directely through forEachMethod  
 map.forEach((key,value)-> System.*out*.println(key+":"+value));  
 //stream through iterate  
 map.entrySet().stream().forEach((obj-> System.*out*.println(obj)));  
 //internal structure of forEach method  
 Consumer<String>consumer=(t-> System.*out*.println(t));  
 for (String sobj:list){  
 System.*out*.println(sobj);  
 }  
  
 }  
}

**Filter Method Example:**

package com.java8.functionalinterface.c.p.s.stream.foreachmethod.filtermethod;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
public class FilterMethod {  
 public static void main(String[] args) {  
 List<String> list = new ArrayList<>();  
 list.add("shalini");  
 list.add("stalin");  
 list.add("shalu");  
 list.add("lini");  
// normal method  
 for (String itemobj : list) {  
 if (itemobj.startsWith("s")) {  
 System.*out*.println(itemobj);  
 }  
 }  
// stream method using list  
 list.stream().filter(t->t.startsWith("s")).forEach((t-> System.*out*.println(t)));  
// stream method using map  
 Map<Integer,String>map=new HashMap<>();  
 map.put(1,"a");  
 map.put(2,"b");  
 map.put(3,"c");  
 map.put(4,"d");  
 map.entrySet().stream().filter(k->k.getKey()%2==0).forEach(obj-> System.*out*.println(obj));  
 }  
}

**Java 8 Optional<?>**

* Java 8 introduced optional class
* Optional class avoid Null Pointer Exception

Example program in Optional class

package com.java8.optionalclass;  
  
import java.util.Arrays;  
import java.util.List;  
  
public class CustomerOptional {  
 private int id;  
 private String name;  
 private String email;  
 private List<String> phoneNumbers;  
// ...generate getters and setters  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public List<String> getPhoneNumbers() {  
 return phoneNumbers;  
 }  
  
 public void setPhoneNumbers(List<String> phoneNumbers) {  
 this.phoneNumbers = phoneNumbers;  
 }  
//......generate constructor.......  
 public CustomerOptional(int id, String name, String email, List<String> phoneNumbers) {  
 this.id = id;  
 this.name = name;  
 this.email = email;  
 this.phoneNumbers = phoneNumbers;  
 }  
//........generate toString.............  
 @Override  
 public String toString() {  
 return "CustomerOptional{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", email='" + email + '\'' +  
 ", phoneNumbers=" + phoneNumbers +  
 '}';  
 }  
}  
class OptionalDemoOne{  
 public static void main(String[] args) {  
 System.*out*.println(new CustomerOptional(1,"shalini","shalinistellus@gmail.com", Arrays.*asList*("2345678","56789432")));

/\*....output....  
CustomerOptional{id=1, name='shalini', email='shalinistellus@gmail.com', phoneNumbers=[2345678, 56789432]}  
 \*/

}  
}

Static Methods in optional class:

1.empty() -> Returns an empty Optional instance.

2.of() -> Returns an Optional with the specified present non-null value.

3.ofNullable -> Returns an Optional describing the specified value, if non-null, otherwise returns an empty Optional.

//.........empty()........  
 Optional<Object> emptyOptional = Optional.*empty*();  
 System.*out*.println(emptyOptional);  
//....run...o/p:optional.empty.....

**//.......of().....(email is present means)**// CustomerOptional customerOptional=new CustomerOptional(1,"shalini","abc@gmail.com", Arrays.asList("2345678","56789432"));  
// Optional<String>emailOptional=Optional.of(customerOptional.getEmail());  
// System.out.println(emailOptional);  
 /\*....output..  
 Optional[abc@gmail.com]  
 \*/  
  **//.......of().....(email is null means)**  
 CustomerOptional customerOptional1=new CustomerOptional(1,"shalini",null, Arrays.*asList*("2345678","56789432"));  
 Optional<String>emailOptional1=Optional.*of*(customerOptional1.getEmail());  
 System.*out*.println(emailOptional1);  
 /\*....output..  
 Exception in thread "main" java.lang.NullPointerException  
 at java.base/java.util.Objects.requireNonNull(Objects.java:208)  
 at java.base/java.util.Optional.of(Optional.java:113)  
 at com.java8.optionalclass.OptionalDemoOne.main(CustomerOptional.java:84)  
  
 \*/

**//.......ofNullable()..................(email is null means)**  
 /\* CustomerOptional customerOptional=new CustomerOptional(1,"shalini",null, Arrays.asList("2345678","56789432"));  
 Optional<String>emailOptional2=Optional.ofNullable(customerOptional.getEmail());  
 System.out.println(emailOptional2);//optional.empty  
  
 \*/  
**//.......ofNullable()..................(email is present means)** /\*CustomerOptional customerOptional=new CustomerOptional(1,"shalini","shalini@gmail.com", Arrays.asList("2345678","56789432"));  
 Optional<String>emailOptional3=Optional.ofNullable(customerOptional.getEmail());  
 System.out.println(emailOptional3);//Optional[shalini@gmail.com]  
  
 \*/  
**//.......ofNullable()..................(email is present means and get() method)** CustomerOptional customerOptional=new CustomerOptional(1,"shalini","shalini@gmail.com", Arrays.*asList*("2345678","56789432"));  
 Optional<String>emailOptional4=Optional.*ofNullable*(customerOptional.getEmail());  
 System.*out*.println(emailOptional4.get());//shalini@gmail.com

**/...........how to avoid NosuchElementException.........................**  
 CustomerOptional customerOptional=new CustomerOptional(1,"shalini","shalini@gmail.com", Arrays.*asList*("2345678","56789432"));  
// CustomerOptional customerOptional=new CustomerOptional(1,"shalini",null, Arrays.asList("2345678","56789432"));  
 Optional<String>emailOptional5=Optional.*ofNullable*(customerOptional.getEmail());  
 if(emailOptional5.isPresent()){  
// System.out.println(emailOptional5.get());/\* email value present means output isshalini@gmail.com\*/  
 System.*out*.println(emailOptional5.map(String::toUpperCase).orElseGet(()->"default mail"));/\*email value present means output is SHALINI@GMAIL.COM\*/  
 }  
 else {  
// System.out.println("return false");/\* email value null means output is return false\*/  
// System.out.println(emailOptional5.orElse("default@gmail.com"));/\* email value null means output is default@gmail.com\*/  
// System.out.println(emailOptional5.orElseThrow(()->new IllegalArgumentException("email not present")));/\* Exception in thread "main" java.lang.IllegalArgumentException: email not present\*/  
 System.*out*.println(emailOptional5.map(String::toUpperCase).orElseGet(()->"default mail"));/\* email value null means output is default mail\*/  
 }  
  
  
  
 }  
  
  
}