Phase 1

11-07-2022

Day 1

Html, css, js, react js ----🡪 React Native

Android

Angular Ionic framework

Day 2 : 12-07-2022

Day 3 : 13-07-2022

Git is a open source sub version control tool which help to record or track the flow of application Or project. It is use to share the code for one team to another team.

3 types of sub version control tool we can use

1. Local version control : RCS Revision control System
2. Central Version Control : SVN : Server and Client. In this version control we get only Remote repository (Remote location folder. Whey client to that repository). They push and pull.
3. Distributed Sub Version control : Git : In this type of tool we get local as well as remote repository.

Git is a distribution sub version control which provided a features local repository which help to connect remote repository ie GitHub, GitLab, AWS, Azure or any cloud remote repository.

Open the git terminal

git --version

git init : This command is use to make the folder as local repository

This command you have to execute only one time.

git status : this command is use to check the current status of local repository

This command execute again and again to check the current status of local repository.

git add filename :This command is use to move the file from local file system to staging area.

Or

git add . : this command is use to add more than one file or folder present in current location.

git commit –m “message1” This command is use to move file or folder from staging area to local repository.

14-07-2022 : Day 4

git branch : git branch is like a pointer which hold more than one commit details.

Git provide by default one branch and the name of the branch may be master or main.

git branch : This command is use to display all branches present in current folder.

git branch branchName : This command is use to create the branch

git checkout branchName : This command is use to move or switch from one branch to anther branch.

Or

git checkout –b branchName : This command is use to create the new branch and switch to that banch.

git merge branchName: This command is use to merge user-defined branch code to current rnach

git branch –D branchName : This command is use to delete the branch

git checkout –b branchname

remote repository

git hub, gitlab, any cloud vendor

please create own git hub account with your personal email id

git branch –m branchName : This command is use to rename the branch

git remote add origin URL This command is use to connect your local repository with remote repository. This command only once.

git push –u origin main This command is use to push the local repository code to remote.

git clone URL : it is use to download remote repository in local machine.

git pull : this command is use to get new update contents from remote repository to existing repository in local machine.

Java : Java is a platform independent and pure object oriented programming language.

Initial name of Java is Oak.

They rename to Java in Nov 1995.

Java developed by Games Josling and team.

It was part of sun micro system and belong to Oracle.

Version of Java

1. 18 version

Day 5:

18-07-2022

object : object is any real world entity. Object is a concept.

Properties or state-🡪 have -🡪 name,age, height, color etc 🡪 variables or fields etc.

Person

Behavior -🡪do/does -🡪 teaching, sleeping, talking, typing etc 🡪 functions or methods.

Bank

Animal

Wheel, price, color etc

Car

Start, appliedGear, moving, stop etc

Employee

Etc

class : blue print of object or template of object or it user-defined data types which is use to describe the object.

syntax of the class

class ClassName {

fields or variable

methods or functions

}

Pre-defined method ie main method and this method must be part of class.

Class name must be follow pascal naming rules.

If class contains one word first letter of class start with upper case like Demo, Test, Employee

If class contains more than one word each word first letter upper case, EmployeeDetails, ManagerInfo etc.

class Demo {

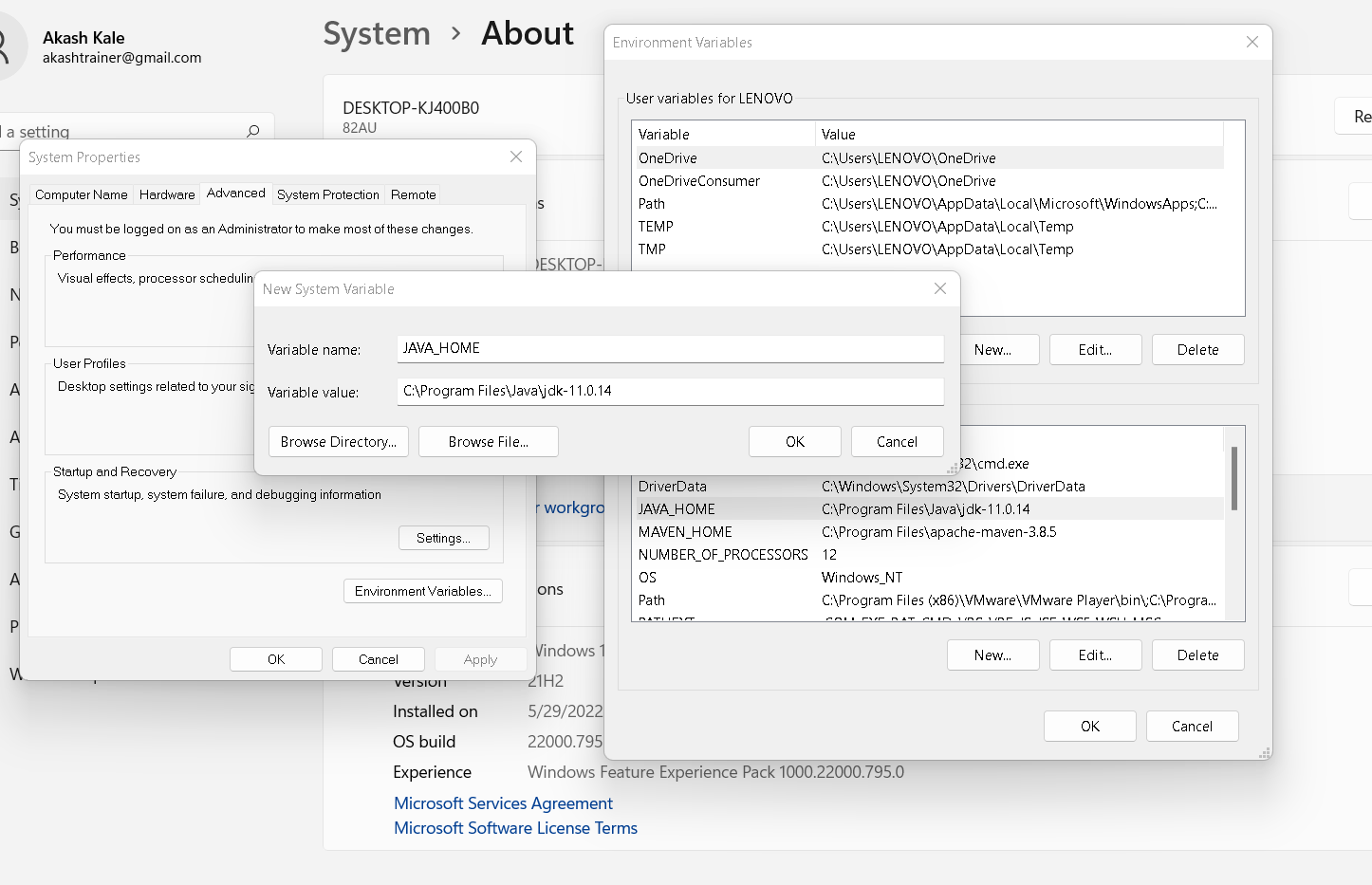
public static void main(String args[]) {

System.out.println(“Welcome to Java”);

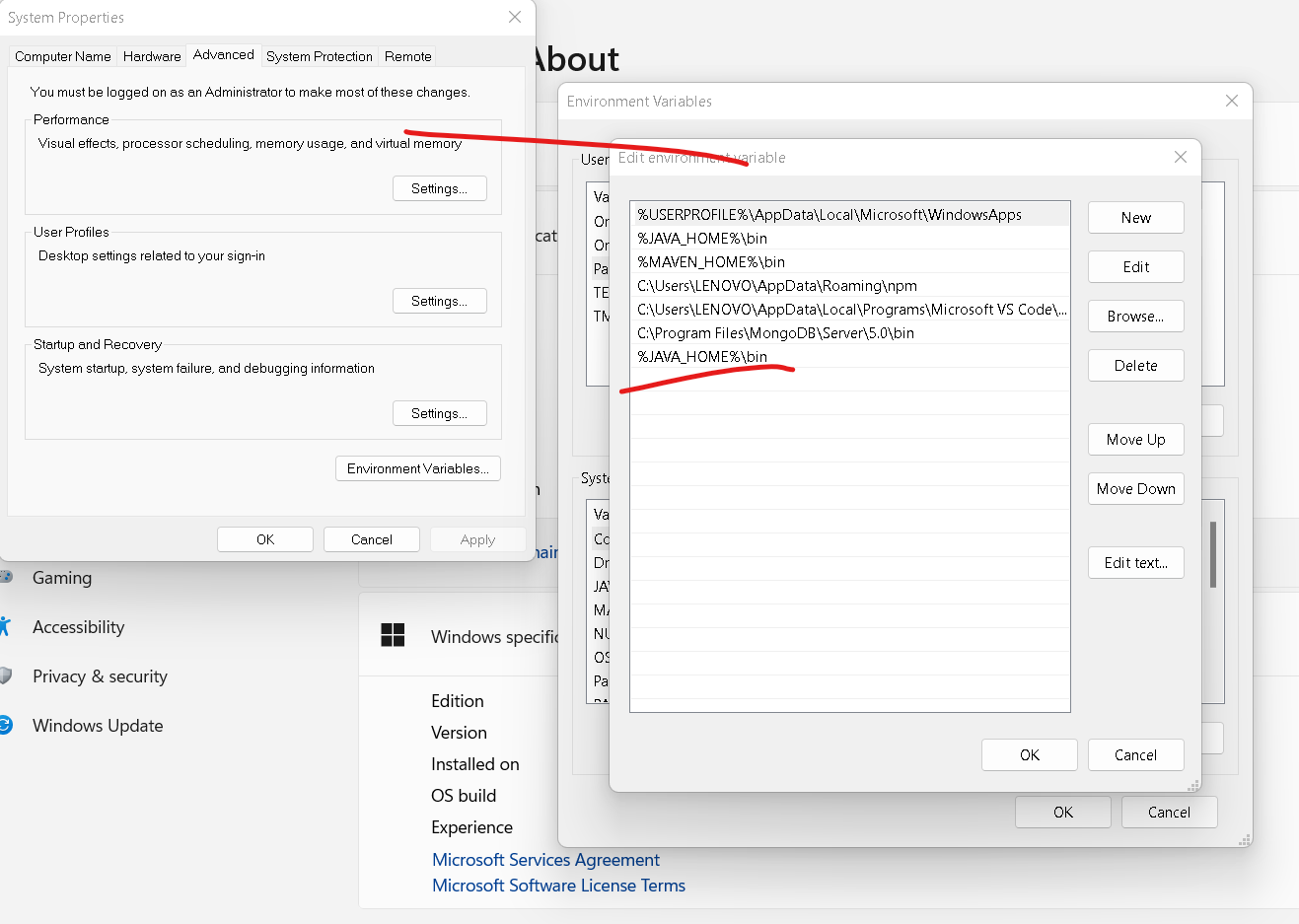
}

}

We have to save the program with className.java



Path



javac Demo.java : it is use to compile the program

java Demo : it is use to run the program

Demo.java

class Demo {

public static void main(String args[]) {

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

System.out.println("Welcome to Java once again");

System.out.print("Welcome again");

System.out.printf("Welcome to Java again\n");

}

}

Data types

It is a type of data which tells what type of data it can hold.

2 types of data types

1. Primitive data types : it is use to store only value

8 types of primitive data types

1. byte 1 byte
2. short 2 byte
3. int 4 byte
4. long : it is use to store the value without decimal. 8 byte
5. float 4 byte
6. double : it is use to store the value with decimal 8 byte
7. char : any single character 2 byte
8. boolean : true or false 1 bit

1. Non primitive data type or reference data type : it is use to store the value as well as reference of another data types.

class Demo {

public static void main(String args[]) {

int a;

int b=10;

System.out.println(b);

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

System.out.printf("value of b is = %d\n",b);

}

}

Type casting : converting one data type to another data type is known as type casting.

2 types

1. Implicit type casting :
2. Explicit type casting :

Int family

--------🡪 implicit type casting -----🡪

byte short int long

🡨-- explicit type casting --------------

Implicit type casting

int float

Explicit type casting

Type casting example

class Demo {

public static void main(String args[]) {

/\*byte a=10;

short b=a; //implicit type casting

System.out.println(a);

System.out.println(b);

short c=10;

byte d =(byte)c; // explicit type casting

System.out.println(c);

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

//float x = 100.10; // error

//float x = (float)100.10;

//float x = 100.10f;

//double x = 100.10;

float x = 100.10f;

int y = (int)x; // explicit type casting

System.out.println(x);

System.out.println(y);

}

}

Operator

1. Arithmetic operator
2. Logical operator
3. Assignment operator
4. Conditional operator
5. Increment and decrement operator
6. Ternary operator

If statement

1. Simple if
2. If else
3. If else if
4. Switch statement : it I use to execute set of statement base upon user or programmer decision.

Syntax

switch(label) {

case v1: block1;

break;

case v2: block2;

break;

case v3: block3;

break;

default : wrongblock

break;

}

switch, case, break and default are keywords.

Switch statement program

class Demo {

public static void main(String args[]) {

int choice =10;

switch(choice) {

case 1:System.out.println("block1");

break;

case 2:System.out.println("block2");

break;

case 3:System.out.println("block3");

break;

default : System.out.println("Wrong choice");

break;

}

System.out.println("finish");

}

}

Taking the value through keyword in java

1. Using Scanner class.
2. Using DataInputStream class
3. BufferedReader class
4. Command line interface

Scanner : it is a pre defined class part of util package. Package is a collection of classes and interface.

Syntax to create the Scanner class object.

Scanner obj =new Scanner(System.in);

obj.nextInt();

obj.nextFloat();

obj.nextBoolean();

obj.next(); it is use to receive the string value. It is use to take only one word

obj.nextLine(); : it is use to take more than one word

looping

it is use to execute the task again and again till the condition become false.

while loop

do while loop

for loop

for each loop or enhanced loop : this type of loop we will use with array or collection of classes.

Non primitive or reference data type

1. array
2. class (it may be pre-defined or user-defined class)
3. interface (it may be pre-defined or user-defined)
4. enum

array : array is use to store more than value of same types.

Syntax

datatype arrayName[];

int abc[];

int a;

syntax of for each loop

for(datatype variableName: arrayName) {

}

Array declaration with initialization and display the value using for loop as well as for each loop

class Demo {

public static void main(String args[]) {

//int abc[10]; //valid in C or C++ but not in java

int abc[]; // valid in java

int xyz[]={10,20,30,40,50,60,100,200,140,670,80,90};

System.out.println("Value of 0 index position "+xyz[0]);

System.out.println("Value of 1 index position "+xyz[1]);

System.out.println("Size of the array is "+xyz.length);

System.out.println("Retrieve the value using for loop");

for(int i=4;i<xyz.length;i++) {

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

}

System.out.println("Retrieve the value using for each loop");

for(int n : xyz) {

System.out.println(n);

}

}

}

Memory creation of array

Syntax

datatype arrayName[]=new datatype[size];

int abc[]=new int[10]; here abc can hold 10 value of type int.

float xyz[]=new float[100]; here xyz can hold 100 value of type float.

String : In java String is a pre-defined class part of lang package. It is also known as reference data types. By default every java program import lang package.

Combination of more than one character enclosed in double quotes.

Syntax to create the String class object.

String str1 = “Welcome to Java Training”; // literal style

String str2 = new String(“Welcome to Java Training”); // creating using new keyword.

String class methods

import java.util.\*;

class Demo {

public static void main(String args[]) {

String str1 = "Welcome to Java Training";

String str2 = new String("Welcome to Java Training");

System.out.println(str1);

System.out.println(str2);

System.out.println(str1.length());

System.out.println(str2.length());

System.out.println(str2.toUpperCase());

System.out.println(str2.toLowerCase());

System.out.println(str2.substring(2));

System.out.println(str2.substring(2,10));

System.out.println(str2.indexOf('a'));

System.out.println(str2.lastIndexOf('a'));

}

}

==

It will check the value as well as reference code or hashcode.

.equals()

It will check only value doesn’t matter it may be same memory or different memory.

Example

import java.util.\*;

class Demo {

public static void main(String args[]) {

String name1 = "Raj Deep";

String name2 = "Raj Deep";

String name3 = new String("Raj Deep");

String name4 = new String("Raj Deep");

if(name3==name4) {

System.out.println("Equal");

}else {

System.out.println("Not Equal");

}

if(name3.equals(name4)) {

System.out.println("Equal");

}else {

System.out.println("Not Equal");

}

}

}

Day 6:

19-07-2022

object and class

object : is any real world entity

class : blue print of object or template of object.

Car

Simple example of user defined object

class Car {

int wheel;

float price;

String color;

void start() {

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

}

void appliedGear() {

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

}

void moving() {

System.out.println("Car is moving");

}

void stop() {

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

}

}

class App {

public static void main(String args[]) {

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

//start();

Car innova = new Car();

innova.start();

innova.stop();

}

}

Types of fields or variables

3 types

1. instance variable
   1. The variable which declared inside a class but outside method including main method is known as instance variable.
   2. Instance variable hold default value according to their data types. Like int family 0, float family 0.0, char space, boolen false, String null.
   3. We can access all instance variable inside a all methods but method must be non static and it must be part of same class.
2. local variable
   1. The variable which declared inside a method is known as local variable.
   2. Local variable doesn’t hold default value we have to initialize while using that variable.
   3. The scope of the variable within that method where it declared.
3. static variable : we will see that concept later on

instance variable and local variable example

class Car {

int wheel;

float price;

String color;

void start() {

int temp=100;

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

System.out.println("Wheel "+wheel);

System.out.println("Price "+price);

System.out.println("Color "+color);

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

}

void appliedGear() {

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

}

void moving() {

System.out.println("Car is moving");

}

void stop() {

String msg="Welcome";

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

System.out.println("Wheel "+wheel);

System.out.println("Price "+price);

System.out.println("Color "+color);

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

}

}

class App {

public static void main(String args[]) {

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

//start();

Car innova = new Car();

innova.start();

innova.stop();

}

}

Creating more than one object and set the value for instance variable from main method

class Car {

int wheel;

float price;

String color;

void displayCarInfo() {

System.out.println("Wheel "+wheel);

System.out.println("Price "+price);

System.out.println("Color "+color);

}

}

class App {

int b;

public static void main(String args[]) {

int a;

Car innova = new Car();

innova.displayCarInfo();

innova.wheel = 4;

innova.color="Gray";

innova.price = 1800000;

innova.displayCarInfo();

Car ertiga = new Car();

ertiga.displayCarInfo();

ertiga.wheel = 4;

ertiga.color = "White";

ertiga.price = 1200000;

ertiga.displayCarInfo();

}

}

Constructor : it is a type of special method which help to create the object.

Pts

1. Constructor have same name as class itself.
2. Constructor doesn’t contains return type not even void also.
3. Constructor no need to call it will call automatically when we create the object.

Simple example of constructor

class Car {

Car() {

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

}

void displayCarInfo() {

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

}

}

class App {

public static void main(String args[]) {

Car innova = new Car();

innova.displayCarInfo();

}

}

Method, empty constructor and parameterized constructor

class Operation {

int a,b,sum;

Operation() {

a=10;

b=20;

}

Operation(int x, int y){

a=x;

b=y;

}

void setValue(int x, int y) {

a=x;

b=y;

}

void add() {

sum = a+b;

}

void display() {

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

}

}

class App {

public static void main(String args[]) {

Operation op1 = new Operation(); op1.display();

Operation op2 = new Operation(); op2.add(); op2.display();

Operation op3 = new Operation(); op3.setValue(100,200); op3.add(); op3.display();

Operation op4 = new Operation(11,22); op4.add(); op4.display();

Operation op5 = new Operation();

op5.setValue(1,2);

op5.setValue(3,4);

op5.setValue(5,6);

op5.add();

op5.setValue(7,8);

op5.display();

Operation op6 = new Operation(100,200);

op6.setValue(2,3);

op6.add();

op6.display();

}

}

In the life of the object if we want to perform any task only one time then we have to use empty parameter constructor. If we want to perform the task more than one time that type of task we have to do inside a method.

Encapsulation : Binding or wrapping data (fields or variables) and code (method or function ) in a single unit is known as encapsulation.

Class :

If instance variable and local variable have same name then local variable hide the visibility of instance variable. To refer to instance variable we have to use this keyword. This keyword is use to refer the current object.

this.instancevariableName

Example

class Employee {

private int id;

private String name;

private float salary;

/\*public void setValue(int id1, String name1, float salary1) {

id = id1;

name = name1;

//salary = salary1;

if(salary1 < 0) {

salary = 8000;

}else {

salary = salary1;

}

}\*/

public void setValue(int id, String name, float salary) {

this.id = id;

this.name = name;

//this.salary = salary;

if(salary < 0) {

this.salary = 8000;

}else {

this.salary = salary;

}

}

void displayEmpInfo() {

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

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

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

}

}

class App {

public static void main(String args[]) {

Employee emp1 = new Employee();

//emp1.id =100;

//emp1.name="Ravi Kumar";

//emp1.salary = -12000;

emp1.setValue(100,"Ravi Kumar",-12000);

emp1.displayEmpInfo();

}

}

Inheritance : Inheritance is use to inherits the properties and behavior of old class to new class.

class OldClass { super class or base class or parent class

properties

behavior

}

class NewClass extends OldClass{ sub class or derived class or child class

properties

behavior

}

class A {

void dis1() {

System.out.println("A class method");

}

}

class B extends A{

void dis2() {

System.out.println("B class method");

}

}

class App {

public static void main(String args[]) {

A obj1 = new A();

obj1.dis1();

B obj2 = new B();

obj2.dis2();

obj2.dis1();

}

}

Types of inheritance

1. Single inheritance : One super class and one sub class

class A { }

class B extends A { }

1. Multilevel inheritance

One super class and n number of sub class connected one by one

class A { }

class B extends A{ }

class C extends B { }

class D extends C { }

1. Hierarchical inheritance

One super class and n number of sub class connected to super class

class A { }

class B extends A { }

class C extends A{ }

class D extends A{ }

1. Multiple inheritance

More than one super class and one sub class

class A { }

class B { }

class C extends A, B { } : Java doesn’t support this type of inheritance . this type of inheritance java support indirectly using interface.

OOPs relationship

Is a relationship

Has a relationship

Manager is a Employee

Developer is a Employee

Employee has a Address

class Employee {

id,name,salary // primitive property

}

class Manager extends Employee {

numbeOfEmp

Address add = new Address();

}

class Developer extends Employee{

projectName

}

class ProjectManager extends Manager {

typeOfProject

}

class Address {

city, state

}

Has relationship

1. Association
2. Aggregation
3. Composition

Association : if we want to achieve has a relationship we have to create the object of one class in another class

class A {

B obj1 = new B(); zero or 1 or many

}

class B {

A obj1 = new A(); zero or 1 or many

}

Aggregation : aggregation is a type of association but known as weak association.

class Employee {

Address add = new Address(); zero or 1 or many

}

class Address {

Employee emp = new Employee() not good relationship

}

Composition :composition is a type of association but known as strong association.

class Student {

StudentHistory sh = new StudentHistory();

}

class StudentHistory {

}

Is a and has a relationship example

import java.util.\*;

class Employee {

private int id;

private String name;

private float salary;

Scanner sc = new Scanner(System.in);

Address add = new Address();

public void readEmp() {

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

id = sc.nextInt();

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

name = sc.next();

System.out.println("Enter the salary");

salary = sc.nextFloat();

}

public void disEmp() {

System.out.println("id is "+id);

System.out.println("name is "+name);

System.out.println("salary is "+salary);

}

}

class Manager extends Employee {

int numberOfEmp;

public void readMgr() {

readEmp();

System.out.println("Enter the number of employee working under him");

numberOfEmp = sc.nextInt();

add.readAdd();

}

public void disMgr() {

disEmp();

System.out.println("Number of emp working "+numberOfEmp);

add.disAdd();

}

}

class Address {

private String city;

private String state;

Scanner ss = new Scanner(System.in);

public void readAdd() {

System.out.println("Enter the city");

city = ss.next();

System.out.println("Enter the state");

state = ss.next();

}

public void disAdd() {

System.out.println("city is "+city);

System.out.println("state is "+state);

}

}

class App {

public static void main(String args[]) {

Manager mgr = new Manager();

//mgr.readEmp();

mgr.readMgr();

//mgr.disEmp();

mgr.disMgr();

}

}

20-07-2022

package com;

class Employee {

int id;

String name;

float salary;

public Employee() {

id=123;

name="Unknown";

salary = 8000;

}

Employee(int id, String name, float salary){

this.id = id;

this.name=name;

this.salary = salary;

}

void display() {

System.out.println("id is "+id);

System.out.println("name is "+name);

System.out.println("salary is "+salary);

}

}

public class ConstructorExample {

public static void main(String[] args) {

Employee emp1 = new Employee(); emp1.display();

Employee emp2 = new Employee(100, "Ravi", 14000); emp2.display();

}

}

Polymorphism : one name many forms or implementation.

2 types of polymorphism

1. Compile time : static binding or early binding : method overloading

The method have same name and same method signature but different parameter list (type of parameter list or number of parameter list). We can achieve method overloading in same class or different class.

public void area(int r) {

double area = 3.142\*r\*r;

}

public void area(int b, int h) {

double area = 0.5\*b\*h;

}

1. Run time : dynamic binding or late binding : method overriding

The method have same name and same method signature ie number of parameter list, type of parameter list and return type must be same.

To achieve method overriding we require inheritance concept mandatory.

Run time polymorphism example

**package** com;

**class** Bike {

**public** **void** speed() {

System.***out***.println("60km/hr");

}

}

**class** Honda **extends** Bike {

**public** **void** color() {

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

}

}

**class** Pulsar **extends** Bike {

**public** **void** speed() {

System.***out***.println("90km/hr");

}

**public** **void** color() {

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

}

}

**class** Tvs **extends** Bike {

**public** **void** color() {

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

}

**public** **void** speed() {

**super**.speed(); // calling super class speed method

System.***out***.println("20km/hr");

}

}

**public** **class** RuntimePolymoprhism {

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

Honda hh = **new** Honda(); hh.color(); hh.speed(); // 60 km/hr

Pulsar pu = **new** Pulsar(); pu.color(); pu.speed(); // 90 km/hr

Tvs tvs = **new** Tvs(); tvs.color(); tvs.speed(); // 60 km/hr 20 km/hr

}

}

abstract, final and static keywords

abstract keyword :

1. Abstract keyword we can use with class and method but not with variable.
2. Abstract method is known as the method without body or without curly braces or incomplete method.

abstract returnType methodName(parameterList);

abstract void speed();

1. If class contains one or more abstract we have to declare that class as abstract class.

abstract class className {

}

1. Whichever class extends abstract class that class must be provide the body for all abstract method belong to that class. That class can ignore only if that class itself is a abstract class.
2. We can’t create the object of abstract class.
3. Abstract class can contains normal as well as abstract method ie it can contains zero or 1 or many abstract method.

Abstract keyword example

**package** com;

**abstract** **class** Bike {

**abstract** **void** speed();

**void** mailage() {

System.***out***.println("55km/hr");

}

}

**class** Honda **extends** Bike {

**void** color() {

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

}

**void** speed() {

System.***out***.println("50km/hr");

}

}

**public** **class** AbstractExample {

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

//Bike bb = new Bike();

Honda hh = **new** Honda();

hh.speed();

hh.color();

hh.mailage();

}

}

Final keyword

1. Final keyword we can use with variable, method and class.
2. Final variable : to declare a constant variable in java we use final keyword. We can’t change the value of final variable
3. Final method : if method is final we can’t override that method in sub class but we can use it.
4. Final class : if class is final we can’t extends or inherits that class.

Static keyword

1. Static keyword we can use with variable and method but not with class. (if class is inner class then we can use static keyword but not with outer class).
2. Static variable : if variable is static we can assign the value for that variable with help of class name.
3. Static method : if method is static we can call that method with help of class name.
4. We can assign the value of static variable even through object also as well as we can call static method with help of object also.
5. Inside a non static method we can access static as well as non static variable directly
6. Inside static method we can access only static variable directly we can’t access non static variable directly we have to create the object of that class.

Static example

**package** com;

**class** Abc {

**int** a; // non static variable

**static** **int** *b*; // static variable

**void** dis1() {

System.***out***.println("Non static method");

System.***out***.println("a "+a);

System.***out***.println("b "+*b*);

}

**static** **void** dis2() {

System.***out***.println("Static method");

Abc obj = **new** Abc();

System.***out***.println("a "+obj.a);

System.***out***.println("b "+*b*);

}

}

**public** **class** StaticExample {

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

Abc obj1 = **new** Abc();

obj1.a=10;

Abc.*b*=20;

obj1.*b*=30;

obj1.dis1();

Abc.*dis2*();

obj1.*dis2*();

}

}

21-07-2022

For every class java provided one static memory. This memory available when our class get load and if any variable are static they are part of that memory. If we create object then heap memory. Number of object equal to number of heap memory.

StaticAndHeap Memory example

**package** com;

**class** Info {

**int** a; // instance variable

**static** **int** *b*; // static variable

**void** display() {

System.***out***.println(" instance variable "+a);

System.***out***.println("static variable "+*b*);

}

}

**public** **class** StaticAndHeapMemory {

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

Info obj1 = **new** Info();

Info obj2 = **new** Info();

obj1.a=10;

obj1.*b*=20;

Info.*b*=30;

obj2.a=40;

obj2.*b*=50;

Info.*b*=60;

obj1.display(); // a= 10 b=60

obj2.display(); // a= 40 b= 60

}

}



Interface : interface is a type of reference data type. Which is also known as 100% pure object class.(till java7).

Syntax

interface interfaceName {

fields;

methods;

}

By default all fields in interface are public, static and final.

By default all methods in interface are public and abstract.

interface Abc {

public static final int A=0;

public abstract void dis1();

}

Interface all variable are final or constant and all methods are abstract.

Like a class one interface can extends another interface but interface can extends more than one interface.

interface Abc {

int A=10;

void dis1();

}

interface Mno {

int B=20;

void dis2();

}

interface Xyz extends Abc,Mno{

int C=30;

void dis3();

}

class Demo implements Abc,Mno{

dis1() and dis2() provide the body.

}

Using interface we can achieve multiple inheritance.

Class always implements interface and class can implements more than one interface.

Whichever class implements one or more interface that class must be provide the body for all abstract methods mandatory.

Interface we can’t create the object.

While overriding with access specifiers

Super class / interface sub class

public public

protected public

protected

default (nothing) public

protected

default (nothing)

private we can’t override the method.

interface Example

**package** com;

**interface** Abc {

**int** ***A***=10;

**void** dis1();

}

**interface** Mno {

**int** ***B***=20;

**void** dis2();

}

**interface** Xyz **extends** Abc,Mno{

**int** ***C***=30;

**void** dis3();

}

**class** Demo **implements** Abc,Mno {

**public** **void** dis1() {

System.***out***.println("This is Abc interface method");

}

**public** **void** dis2() {

System.***out***.println("This is Xyz interface method");

}

}

**public** **class** InterfaceExample {

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

Demo obj = **new** Demo();

obj.dis1();

obj.dis2();

}

}

Difference between interface and abstract class

Run time polymorphism with object creation

Super class reference and sub class object with super class is normal class example

**package** abc;

**class** A {

**void** dis1() {

System.***out***.println("A class dis1 method");

}

}

**class** B **extends** A {

**void** dis1() {

System.***out***.println("A class dis1 method override by B class");

}

**void** dis2() {

System.***out***.println("B class dis2 method");

}

}

**public** **class** Test1 {

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

A obj1 = **new** A(); obj1.dis1(); // super creaiting super class object and same class reference. possible

B obj2 = **new** B(); obj2.dis1(); obj2.dis2(); // sub class creating sub class object and same class reference possible

///B obj3 = new A(); // creating super class object and sub class referene not possible

A obj4 = **new** B(); // creating sub class object and super class reference possible. run time polymorphism

// with help of that reference we can call only those method which belong to super class or override method in sub class.

obj4.dis1();

//obj4.dis2(); // this method purely belong to sub class.

}

}

Creating sub class object and super class reference and super class is type of abstract class

**package** mno;

**abstract** **class** A {

**abstract** **void** dis1();

}

**class** B **extends** A {

**void** dis1() {

System.***out***.println("A class dis1 method override by B class");

}

**void** dis2() {

System.***out***.println("B class dis2 method");

}

}

**public** **class** Test2 {

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

//A obj1 = new A(); // we can create object of super class because it is abstract class

B obj2 = **new** B(); obj2.dis1(); obj2.dis2();

A obj3 = **new** B(); // We can create sub class object and super class reference super class can be

// be abstract class. with that reference we can call only those method which belong to abstract class.

obj3.dis1(); // body is provided by sub class.

}

}

Sub class object and interface reference

**package** xyz;

**interface** A {

**void** dis1();

}

**class** B **implements** A {

**public** **void** dis1() {

System.***out***.println("A interface dis1 method override by B class");

}

**void** dis2() {

System.***out***.println("B class dis2 method");

}

}

**public** **class** Test3 {

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

//A obj1 = new A(); // we can create object of interface

B obj2 = **new** B(); obj2.dis1(); obj2.dis2();

A obj3 = **new** B(); // We can create sub class object and interface reference

// with help ofthat reference we can call only those method which belong to interface.

obj3.dis1(); // body is provided by sub class.

}

}

abstraction : hiding the internal implementation without knowing background details.

In java we can achieve 100% abstraction using interface.

Example

**package** com;

**interface** Team1 {

**public** **int** add(**int** x, **int** y);

}

**interface** Team2 {

**public** **int** sub(**int** x, **int** y);

}

**class** Manager **implements** Team1, Team2 {

**public** **int** add(**int** x, **int** y) {

**return** x+y;

}

**public** **int** sub(**int** x, **int** y) {

**return** x-y;

}

**public** **void** ownMethod() {

System.***out***.println("manager own method");

}

}

**public** **class** Test4 {

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

Manager mgr = **new** Manager();

System.***out***.println(mgr.add(100, 50));

System.***out***.println(mgr.sub(100, 60));

mgr.ownMethod();

Team1 t1 = **new** Manager();

System.***out***.println(t1.add(100, 40));

//System.out.println(t2.sub(40,20));

Team2 t2 = **new** Manager();

System.***out***.println(t2.sub(30, 20));

}

}

this, super, this() and super()

**package** com;

**class** A {

**int** x=10;

}

**class** B **extends** A {

**int** x=20; // sub class variable hide the visibility of super class variable.

**public** **void** dis() {

**int** x=30; // local variable hide the visiblity of instance variable

System.***out***.println("x super class variable "+**super**.x);

System.***out***.println("x instance variable "+**this**.x);

System.***out***.println("x "+x);

}

}

**public** **class** ThisAndSuperKeyword {

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

B obj1 = **new** B();

obj1.dis();

}

}

super() and this()

this() it is use to call same class constructor like a constructor chaining for same class. It must be first statement inside a constructor.

super() : it is use call sub class constructor to super class constructor like a constructor chaining from sub class to super class. It must be first statement inside a sub class constructor. By default every sub class contains super() parameter which always internally call super class empty constructor.

Example

**package** com;

**class** Employee {

**int** id;

String name;

**float** salary;

**public** Employee() {

**this**.id =123;

**this**.name = "Unknown";

**this**.salary = 8000;

}

**public** Employee(**int** id) {

**this**(); // calling empty constructor using this()

**this**.id = id;

}

**public** Employee(**int** id, String name) {

**this**(id); // calling constructor with id a parameter

**this**.name = name;

}

**public** Employee(**int** id, String name, **float** salary) {

**this**(id,name);

**this**.salary = salary;

}

**void** dis() {

System.***out***.println(" id is "+id);

System.***out***.println("name is "+name);

System.***out***.println("salary is "+salary);

}

}

**class** Manager **extends** Employee {

**int** numberOfEmp;

**public** Manager(**int** id, String name, **float** salary, **int** numberOfEmp) {

// super(); // it will call super class empty constructor

**super**(id,name,salary); // it wll call super class parameter constructor with parameter it will call line number 19

**this**.numberOfEmp = numberOfEmp;

}

**void** disMgr() {

System.***out***.println("Number of employee working under him "+numberOfEmp);

}

}

**public** **class** SuperAndThisParameterExample {

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

// Employee emp1 = new Employee(); emp1.dis();

// Employee emp2 = new Employee(1); emp2.dis();

// Employee emp3 = new Employee(2,"Ravi"); emp3.dis();

// Employee emp4 = new Employee(3, "Ajay", 14000); emp4.dis();

Manager mgr1 = **new** Manager(100, "Mahesh", 45000, 10);

mgr1.dis();

mgr1.disMgr();

}

}

package and access specifiers

package is a collection of classes and interfaces. Package is just like a directory when more than one classes and interface which have same name but functionality wise they work differently.

2 types

1. User-defined package
2. Pre-defined package

Education

school college

Attendance Attendance

Date

java.util.\*;

java.sql.\*;

access specifiers it is use to expose the visibility of class, variable and methods etc.

4 types

1. private : we can use private access specifiers with all except class and local variable.

Scope : within a same class.

1. default : we can use with call.

Scope : within a same package

1. protected : we can use protected access specifiers with all except class and local variable.

Scope : within a same package other package if it is sub class

1. public : we can use with all except local variable.

Scope : same package as well as other package.

Class A

Dis1()

A class method

Class B

Dis1()

B class method

Class C extends A, B

Interface A

Dis1()

Interface B

Dis1()

Class C implements A, B

Dis1

It is A and B interface method

22-07-2022

Exception Handling:

Exception is a pre-defined object or memory in java which occurs when unexpected or abnormal condition occurs during the execution of a program is known as exception. To handle generated exception using some technique is known as exception handling.

java

compile program run program

javac

after compile program it will create the

.class file. The create bytecode which we can’t understand

These byte code can be understand by jvm.

Compile time error run time error

Syntax error or type error.

Run time error

Error Exception

Error and Exception both are pre-defined classes part of lang package.

Error : it is type of error which generate at the run time which we can’t handle it . JVM crash, software or hardware issue or out of memory etc.

Exception : it is a type of error which generated at the run the which we can handle it. Divided by zero or array index bound exception ie wrong index position.

By default every java program internally extends Object class. So Object class is super class for all pre-defined or user-defined class.

Object

Throwable

Error Exception

Checked exception unchecked exception

SQLException

IOException

FileNotFoundException RuntimeException

Etc

ArrayIndexOutOfBoundsException

ArithmeticException

NumberFormatException

Etc

To handle both the type of exception java provided 5 keywords.

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

try and catch block

syntax

try {

}catch(Exception e) {

}

Try with single catch block is use to handle all type of exception. This catch block is use to handle generic exception.

Try with multiple catch block

try {

}catch(ArithmeticException e) {

}catch(ArrayIndexOutOfBoundsException e) {

}catch(NullPointerException e) {

}

Try block : the code which generate exception one line code or more than one line we have to keep in try block

Catch block : this block execute only if any exception generate. This block also known as exception solver block.

Finally block : finally block will execute 100% sure any exception generate or not.

try

catch catch catch catch finally

catch finally catch

catch finally

finally block mainly use to close the resources.

try {

open file

read and write operation

}catch(Exception e) {

}finally {

Close the file

}

throw : this keyword is use to raise or generate any pre-defined or user-defined exception according to our requirements.

Syntax

throw new Exception();

or

throw new ExceptionSubClass()

throws throws keyword we can use with method signature to throw the exception to caller method.

void display() throws Exception,ExceptionSubClass {

}

Unchecked exception check at the run time and all unchecked exception are sub class of RuntimeException class. We can avoid un checked exception.

Checked exception it check twice at compile time as well as run time. We have to handle the checked exception mandatory we can’t avoid it.

Multi threading Overview

Program : Set of instruction to perform a specific task.

Processor : Processor is responsible to execute the task.

Process : time taken to execute the code or program in execution.

Thread : Thread is a small execution of a code within a process. Thread also known as light weighted process. It take less resources or memory of our machine.

Java is by default thread base programming language.

Inside a main method in java always default thread execute.

To find the default thread details running inside a mam method

Thread is a pre-defined class part of lang package which contains lot of pre-defined method.

currentThread is a pre-defined method part of thread class and it is static method and this method return type is Thread class reference.

Thread t = Thread.currentThread();

System.out.println(t); Thread[main,5,main]

Main -🡪 name of the thread

5 -🡪 priority of the thread

Main 🡪 group of the thread.

Default thread priority is 5. We can set min 1 and max 10. We can’t set less 1 and more than 10.

Range of the priority between 1 to 10.

public void display() {

}

public int display1() {

return 10;

}

public String display2() {

return “Welcome”;

}

public static Thread currentThread() {

//coding

return t;

}

Multi tasking :

Task1 Task2 Task3

Using process base

Using thread base

In Java we can create more than one thread using two ways

1. extends Thread
   1. we have to create normal class and that class must be extends Thread class.
   2. We have to create the thread class reference.
   3. With help of reference we have to call start() method. Start is a pre-defined method which help to start the thread.
   4. Start method internally call run() method of Thread . run method is a part of thread class which contains empty body.
   5. If we want to execute any custom code we have to override run method inside that class. The class which extend Thread class.
2. implements Runnable interface
   1. we have to create the normal class and that class must be implements Runnable interface.
   2. Runnable interface contains run method so we have to override mandatory.
   3. Now have to create the reference of thread class. To call start method.

25-07-2022

Difference between creating thread using extends and implements.

Booking the ticket is one of the type of task. Multiple people do at the same time. Multiple people equal to multiple thread.

Synchronization : synchronization is concept which help to block or lock the thread. It will allow to use only one thread all resource at time.

To achieve the synchronization we can use synchronized keyword.

This keyword we can use with method or block.

Wait, notify and notifyAll

wait, notify and notifyAll method are belong to Object class.

Wait method is use to suspend the thread. And notify method is use to resume or call back suspended thread.

These three method work property inside a synchronized method only.

Using these concept we can inner thread communication.

Consumer and producer example for wait and notify

File handling using IO package

IO package which provide set of classes and interfaces which help to do file handling program.

Stream : flow of data or it is a abstraction between source and destination.

Stream

byte char

Input Output Input Ouput

InputStream OutputStream Reader Writer

All four are abstract classes.

DataInputStream DataOutputStream InputStreamReader OutputStreamWriter

BufferedInputStream BufferedOutputStream BufferedReader BufferedWriter

FileInputStream FileOutputStream FileReader FileWriter

ObjectInputSteram ObjectOutputStream

PrintStream PrintWriter

Java provided System pre-defined class which contains three constant or final and static field ie in, out and err.

in is a reference of InputStream . Here System.in consider as keyword reference

out and err is a reference of PrintStream . Here System.out consider as output reference or console.

InputStream reference always refer to standard input device ie keyword.

OutputStream reference always refer to standard output device ie console.

Scanner sc = new Scanner(System.in);

System.out.println();

Byte wise operation

Input : keyboard

Output : console

Java provided buffer classes which help to improve performance 1000 time more. Buffer is temporary memory.

File : IO package provided pre-defined class File which help to check the file properties read mode, write model, size, exists, file or directory. Even we can create empty file as well as delete the file.

Do {

Switch() {

Case 1:

Case 2:

}

}while()

1: display all the files in ascending order.

2:

1. Creating new file in current directory with or without data.
2. Delete the file
3. Search the file present or not.
4. Exit sub option
5. Exit main option

26-07-2022

Collection framework with data structure

Data structure is use to store the proper or organized manner which help manipulate on that data every easily like adding, removing, searching, sorting, retrieving.

While storing data we have to remember few points

1. Memory
2. Time complexity

To improve memory and time complexity we have to use set of algorithms.

In some language we use all algorithms from scratch. Some language they provide pre-defined API which help to use all algorithms.

In Java Collection framework is like a data structure.

Stack ss = new Stack();

ss.push(100);

int a=10;

a=20;

int abc[];

abc is a type of array which can hold more than one value of type int.

structure : in C language we can use structure concept to store different type of values. But java doesn’t support structure.

class : class is one of type of user-defined data type which help to store different type of values.

class Employee {

int id;

String name;

float salary;

}

Employee emp = new Employee();

emp.id=100;

emp.name= “Ravi”;

emp.salary = 12000;

object array : object array is use to store more than one objects.

className referenceName[]=new className[size];

Employee employees[]=new Employee[100];

employees[0]=new Employee();

employees[1]=new Employee();

employees[2]=new Employee();

employees[3]=new Employee();

JavaBean class :

It is a type of normal class but we have to follow few rules while creating java bean class

1. class must be public.
2. All variable in java bean must be private.
3. For each variable we have to provide setter and getter method.
4. Setter method is use to set the value and setter method access specifiers must be public
5. Getter method is use to get the value and getter method access specifiers must be public
6. JavaBean class also known as 100% pure encapsulation class.

public class Employee {

private int id;

private String name;

private float salary;

public void setId(int id){

this.id = id;

}

public int getId() {

return this.id;

}

}

Whenever we display any class reference in println it will call internally toString() of Object class. That method return string as as [packageName.classname@code](mailto:packageName.classname@code).

Limitation of array object.

1. Array object is fixed memory size.
2. It allow to store only same class object like Employee or any other class object.
3. It doesn’t provide any pre-defined method which help to add, remove, iterate or search element or value from array object.

Collection Framework : Collection framework provided set of classes and interface which help to store the collection of object or elements of primitive type or user-defined as well as pre-defined objects (may be same or different types).

It provided lot of pre-defined method help to do manipulation on those values like adding, removing, searching, sorting, iterating etc.

Collection Framework hierarchy

All api is a part of util package.

Collection ---🡪 interface

Set List Queue **Map**

All are interfaces Set, List and Queue internally extends Collection but Map doesn’t extends Collection.

Set : set is use to store the collection of elements. Set doesn’t allow duplicate. Under set some classes maintain order or Unorder (random) or sorted.

Set classes

HashSet : HashSet maintain the element in unorder format.

LinkedHashSet : LinkedHashSet class internally extends HashSet. That class doesn’t provide any extra method. It only maintain the elements order.

TreeSet : TreeSet is a type of Set API internally implements SortedSet interface. SortedSet interface extends Set interface. SortedSet interface provide algorithms to display the elements in ascending order. In TreeSet we can stored in same type values. This class provided few extra methods like headset, tailset, subset etc.

HashSet Vs LinkedHashSet

HashSet Vs TreeSet

LinkedHashSet vs TreeSet

List : list is use to store the collection of elements. List allow duplicate and it maintain the order.

List classes

Stack ­ Stack is a type of List which allow duplicate. Stack use first in last out or last in first out features.

ArrayList

LinkedList

Vector

Queue : it is a type of data structure which provide the feature first in first out. It allow duplicate.

ArrayList

PriorityQueue

Map : it is use to store the information in the form of key value pairs. Key is unique and value may be duplicate.

HashMap

LinkedHashMap

TreeMap

Service class or service layer : this class contains pure business methods.

27-07-2022

ArrayList : ArrayList is a type of collection class which provided set of methods which help to add and remove element very easily from array.

Normal array vs ArrayList

1. Normal array is use to store same type of values. But by nature ArrayList allow to store any type of values.
2. Normal array fixed in memory size. ArrayList dynamic memory.
3. Using normal array to add or remove between elements is more complex. But using ArrayList we can take the help of pre-defined method to do operation very easily.

LinkedList : LinkedList is a type of List API in java. It is use node to store the value. Node divided into 2 or 3 parts base open type of LinkedList.

1. Singular linked list
2. Double linked list
3. Circular linked list



By default LinkedList object consider as double linked list.

Difference between ArrayList and LinkedList

Vector : Vector is known as legacy class means old class. In Vector class all methods are by default synchronized. So it is a thread safe.

Work is safe. Performa wise slow.

Difference between ArrayList and Vector

Queue : Queue is do the operation first in first out

The implementation of Queue is PriorityQueue and LinkedList.

It store the elements in key – value format.

HashMap

LinkedHashMap

TreeMap

Hashtable

Retrieve the element or object from collection of classes one by one

1. For each loop : for set, list , queue but not for map
2. Iterator : for set, list etc
3. ListIterator : list

For each loop is a type of loop and Iterator and ListIterator is a type of interface which provide set of methods to retrieve the elements one by one.

Collection Framework with Generics

CollectionClass <Type>objectReferenceName=new CollectionClass<Type>();

Type : Integer, Float, Double, String, Character, or any User-definedClassName like Employee, Customer etc

Collection framework with Complex or user-defined objects.

Employee, Account, Customer, Product, etc --🡪 JavaBean class

Variable are private , constructor and setter and getter methods and toString method.

EmployeeService, AccountService, CustomerService --🡪 pure business logic

CRUD Operation , Create, Delete, Update and Retrieve employee

Service class not responsible to take the value through keyboards. Don’t create Scanner class object in Service layer.

Main class which contains menu driven operation

1 add, 2 delete, 3 update and 4 retrieve

App.java

**package** com.main;

**import** com.bean.Account;

**import** com.service.AccountService;

**public** **class** App {

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

AccountService as = **new** AccountService();

Account acc = **new** Account();

acc.setAccno(1);

acc.setName("Raj");

acc.setAmount(500);

String result = as.createAccount(acc);

System.***out***.println(result);

Account acc1 = **new** Account();

acc1.setAccno(2);

acc1.setName("Ravi");

acc1.setAmount(700);

String result1 = as.createAccount(acc1);

System.***out***.println(result1);

Account searchAccountDetails = as.findAccount(3);

**if**(searchAccountDetails==**null**) {

System.***out***.println("Account details not present");

}**else** {

System.***out***.println(searchAccountDetails); // toString method

}

}

}

AccountService

**package** com.service;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**import** java.util.List;

**import** com.bean.Account;

**public** **class** AccountService {

List<Account> listOfAccount = **new** ArrayList<Account>();

**public** String createAccount(Account account) {

**int** flag=0;

**if**(listOfAccount.size()==0) {

listOfAccount.add(account);

**return** "Account created";

}**else** {

Iterator<Account> li = listOfAccount.iterator();

**while**(li.hasNext()) {

Account acc = li.next();

**if**(acc.getAccno()==account.getAccno()) {

flag++;

**break**;

}

}

}

**if**(flag>0) {

flag=0;

**return** "Account number must be unique";

}**else** {

listOfAccount.add(account);

**return** "Account created";

}

}

**public** Account findAccount(**int** accno) {

Iterator<Account> li = listOfAccount.iterator();

**while**(li.hasNext()) {

Account acc = li.next();

**if**(acc.getAccno()==accno) {

**return** acc;

}

}

**return** **null**;

}

**public** **float** findAccountBalance(**int** accno) {

Iterator<Account> li = listOfAccount.iterator();

**while**(li.hasNext()) {

Account acc = li.next();

**if**(acc.getAccno()==accno) {

**return** acc.getAmount();

}

}

**return** -1;

}

**public** List<Account> getAllAccount() {

**return** listOfAccount;

}

}

Account.java

**package** com.bean;

**public** **class** Account {

**private** **int** accno;

**private** String name;

**private** **float** amount;

**public** **int** getAccno() {

**return** accno;

}

**public** **void** setAccno(**int** accno) {

**this**.accno = accno;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** **float** getAmount() {

**return** amount;

}

**public** **void** setAmount(**float** amount) {

**this**.amount = amount;

}

**public** Account() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** Account(**int** accno, String name, **float** amount) {

**super**();

**this**.accno = accno;

**this**.name = name;

**this**.amount = amount;

}

@Override

**public** String toString() {

**return** "Account [accno=" + accno + ", name=" + name + ", amount=" + amount + "]";

}

}

Arrays.sort(arrayvarialbeName);

Arrays.sort(arrayVariableName);

28-07-2022

Collection Framework provided pre-defined utilities classes which help to do searching and sorting.

Arrays : This class provided lot of static method which help to do searching and sorting on primitive array of type int, float, chat, string etc.

Collections : This class provided lot of static method which help to do searching and sorting on List of type int, float, chat, string or even complex object etc.

When we pass the list reference in Collections.sort(listReference). it will check the list reference hold what type of values

If list reference hold primitive value as well as string value it doesn’t give any error. It hold complex or user-defined object it will show some error.

Because by default all wrapper classes (Integer, Float, Char,Double) and String class internally implements Comparable and that interface provide one method compareTo(T) which provide logic to do the sorting.

Comparator

Difference between Comparable and Comparator

Primitive data types Wrapper classes

byte Byte

short Short

int Integer

long Long

float Float

double Double

char Character

boolean Boolean

int a; a is a variable of type int. on a variable we can do only mathematical operation.

Wrapper classes wrap primitive value and provide set of methods which help to convert primitive to object and vice-versa.

Wrapper classes us to do type casting.

int a=10;

Integer b = new Integer(a);

int c = b.intValue();

float d = b.floatValue();