Greenfield Training Assignment(Day 3 and 4)

Inheritance is one of the divisions of object-oriented programming because it allows

the creation of hierarchical classes. Using inheritance, we can create a general class

that defines traits common to a set of related items such as member variables and methods . This class can then be inherited by other, more specific classes, each adding those things that are unique to it. In the terminology of Java, a class that is inherited is called a superclass. The class that does the inheriting is called a subclass. Therefore, a subclass is a specialized version of a superclass. It inherits all of the members defined by the superclass and adds its own, unique elements.

Member Access - Although a subclass includes all of the members of its superclass, it cannot access those members of the superclass that have been declared as private.

Final - If a class or a method in a class is declared as final , then such class or method cannot be extended by other subclasses.

Abstract Classes and Methods - Creating a class for only inheritance , static and we canto create an instance of that class

Abstract methods can be only declared in abstract classes. It can contains normal or abstract mehods.We use absract classes when we really want to use inheritance.

Example Program

final class Parent{

public void add(int a,int b){

int c = a+b;

System.out.println("Sum = "+c);

}

}

abstract class ABC{

public abstract void subtract(int c,int d);

}

class DEF extends ABC{

@Override

public void subtract(int e,int f) {

// TODO Auto-generated method stub

int g = e - f;

System.out.println("Ans = "+g);

}

class HIJ extends Parent{// Generates error as final class can’t be extended

void add(int x,int y,int z){

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

}

}

class lab6 {

public static void main(String[] args) {

// TODO Auto-generated method stub

DEF o = new DEF();

o.subtract(4,3);

}

}

Interfaces - Interfaces are syntactically similar to classes, but they lack instance variables, and, as a general rule, their methods are declared without any body. In practice, this means that you can define interfaces that don’t make assumptions about how they are implemented. Once it is

defined, any number of classes can implement an interface. Also, one class can implement

any number of interfaces. We can’t create an instance of an interface , the advantages of interface is that it can inherit the functions and can give different function declaration.However we can create a pointer for the interface.No data variables , no actual methods , only abstract methods can be used in an interface. , to implement the interface , write code for all instance methods. We can also use interfaces as a pointer.

Example Program

interface Sort {

public void dosort();

}

class Bubble implements Sort{

@Override

public void dosort() {

System.out.println("Bubble - dosort");

}

}

class Shell implements Sort{

@Override

public void dosort() {

System.out.println("Shell - dosort");

}

}

public class demo3 {

public static void main(String[] args) {

String ch ="s";

Sort sort = null;

if (ch.equals("b"))

sort = new Bubble();

else

sort = new Shell();

sort.dosort();

}

}

Exception Handling

A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code. When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error. That method may choose to handle the exception itself, or pass it on. Either way, at some point, the exception is caught and processed. Exceptions can be generated by the Java run-time system, or they can be manually generated by your code. Exceptions thrown by Java relate to fundamental errors that violate the rules of the Java language or the constraints of the Java execution environment. Manually generated exceptions are typically used to report some error condition to the caller of a method. Java exception handling is managed via five keywords: try, catch, throw, throws, and finally.

Example Program

class Exc2 {

public static void main(String args[]) {

int d, a;

try { // monitor a block of code.

d = 0;

a = 42 / d;

System.out.println("This will not be printed.");

} catch (ArithmeticException e) {

System.out.println("Division by zero.");

}

System.out.println("After catch statement.");

}

}

Collection APIs - The collection framework is used to group certain data of using wrapper classes , some of the types are:

1.List

2.ArrayList

3.Hash Map

4.Map

Example Programs

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class demo {

public static void main(String[] args) {

List l1 = new ArrayList();

l1.add("string");

l1.add(300);

l1.add(400.300);

System.out.println(l1);

List<String> list = new ArrayList<>();

list.add("AA");

list.add("BB");

list.add("CC");

list.add("DD");

list.add("DD");

Scanner sc = new Scanner(System.in);

System.out.println("Enter The pos");

int a = sc.nextInt();

System.out.println("Enter The String");

String b = sc.next();

list.add(a,b);

list.add(1,"FF");

list.set(0,"HH");

System.out.println(list);

System.out.println("Enter an element you would like to remove");

String c = sc.next();

list.remove(c);

System.out.println(list);

}

}

Example Program

import java.util.Map;

import java.util.TreeMap;

public class demo5 {

public static void main(String[] args) {

Map<String,Integer> map =new TreeMap<>();

map.put("ABC", 90);

map.put("DEF", 54);

map.put("GHI", 08);

System.out.println(map);

map.put("JKL",99);

map.put("MNO",45);

System.out.println(map);

System.out.println("Marks for ABC= " + map.get("ABC"));

}

}

MultiThreading

A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. Thus, multithreading is a

specialized form of multitasking.

Example Program

class Threads {

public static void main(String args[]) {

Thread t = Thread.currentThread();

System.out.println("Current thread: " + t);

t.setName("My Thread");

System.out.println("After name change: " + t);

try {

for(int n = 5; n > 0; n--) {

System.out.println(n);

Thread.sleep(1000);

}

} catch (InterruptedException e) {

System.out.println("Main thread interrupted");

}

}

}