Write a singletone class.Confirm that singletone class cannot be inherited.

package pgm1;

import java.util.Scanner;

public class emp {

public static void main(String[] args)

{

manager m=new manager();

m.emp();

m.incentive();

labour l=new labour();

l.emp();

l.OT();

}

}

package pgm1;

public class employee {

void emp()

{

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

}

}

package pgm1;

import java.util.Scanner;

public class manager extends employee {

void incentive()

{

Scanner scan=new Scanner(System.in);

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

int ince;

ince=scan.nextInt();

ince=ince+10000;

System.out.println("salary of the manager:"+ince);

}

}

package pgm1;

import java.util.Scanner;

public class labour extends employee {

void OT()

{

Scanner scan=new Scanner(System.in);

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

int OT;

OT=scan.nextInt();

OT=OT+10000;

System.out.println("salary of the labour:"+OT);

}

}

Output:

employee salary

incentive:

1000

salary of the manager:11000

employee salary

OT:

200

salary of the labour:10200

1. write program to consider saving and current bank holder.

package polymorphism;

public class poly {

public static void main(String[] args)

{

bank b ;

b=new saving();

b.display();

b=new current();

b.display();

}

}

package polymorphism;

class bank {

void display()

{

System.out.println("account");

}

}

package polymorphism;

public class saving extends bank{

void display()

{

int number=10000;

System.out.println("saving account holder");

System.out.println("Fixed deposit:"+number);

}

}

package polymorphism;

public class current extends bank {

void display()

{

int number=5000;

System.out.println("Current account holder");

System.out.println("credit cash:"+number);

}

}

Output:

saving account holder

Fixed deposit:10000

Current account holder

credit cash:5000

3. Test the following principle regarding abstraction

i. if any class has any of its method abstract then you must declare entire class abstract

package abstarct;

public class demo extends call {

public void dispaly2()

{

System.out.println("HI");

}

public static void main(String[] args)

{

demo d= new demo();

d.dispaly();

}

}

**package** abstarct;

**abstract** **class** call {

**void** display()

{

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

}

}

Output: Not necessary that abstract class have abstract method and methods donot require implementation for declaration

ii. When we extend an abstract class ,we must override all the abstract method in subclass or declare sub class as abstract

package abstarct;

public class demo extends call {

public void dispaly2()

{

System.out.println("HI");

}

public static void main(String[] args)

{

demo d= new demo();

d.dispaly2();

}

}

package abstarct;

abstract class call {

void display()

{

System.out.println("hello");

}

}

Output: HI

iii. Abstarct class cannot be private

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

Implicit super constructor call() is not visible for default constructor. Must define an explicit constructor

Iv . Abstarct class cannot be final

Yes it can’t be a final because final is not a access specifier

v. you can declare abstract class without abstract method

yes we can create abstract class without a method.

1. Develop an application for dessert shop. The application should allow the owner to add items like candy,cookie,ice cream in shop storage. Also customer should be able to place an order.

DessertItem.java

package oops;

public abstract class DessertItem {

protected String name;

public DessertItem()

{

name="";

}

public DessertItem(String name1)

{

name=name1;

}

public String getName()

{

return name;

}

public void setName(String name1)

{

name=name1;

}

public abstract double getCost();

}

Candy.java

package oops;

public class candy extends DessertItem {

private double weight;

private double pricePerPound;

public candy()

{

super();

weight=0;

pricePerPound=0;

}

public candy(String name,double w,double prc)

{

super(name);

weight=w;

pricePerPound=prc;

}

public double getWeight()

{

return weight;

}

public void setWeight(double weight)

{

this.weight=weight;

}

public double getPricePerPound()

{

return pricePerPound;

}

public void setPricePerPound(double pricePerPound)

{

this.pricePerPound=pricePerPound;

}

public double getCost()

{

double total=weight\*pricePerPound;

total=Math.round(total\*100);

return total;

}

public String toString()

{

String s=String.format("%-50s$%2f\n\t%.2f lbs@$.2f",getName(),getCost()/100,weight,pricePerPound);

return s;

}

}

Cookie.java

package oops;

public class cookie extends DessertItem {

private int quantity;

private double pricePerDozen;

public cookie()

{

super();

quantity=0;

pricePerDozen=0;

}

public cookie(String name,int qty,double prc)

{

super(name);

quantity=qty;

pricePerDozen=prc;

}

public double getQuantity()

{

return quantity;

}

public double getPricePerDozen()

{

return pricePerDozen;

}

public void setPricePerDozen(double pricePerDozen)

{

this.pricePerDozen=pricePerDozen;

}

public void setQuantity(int quantity)

{

this.quantity=quantity;

}

public double getCost()

{

double total=pricePerDozen/12\*quantity;

total=Math.round(total\*100);

return total;

}

public String toString()

{

String s=String.format("%-50s $.%2f\n\t%d cookies@ $%.2f per Dozen",getName(),getCost()/100,quantity,pricePerDozen);

return s;

}

}

IceCream.java

package oops;

public class IceCream extends DessertItem

{

private int numberOfScoops;

private double pricePerScoop;

private double toppingPrice;

public IceCream()

{

super();

numberOfScoops=0;

pricePerScoop=0;

toppingPrice=0;

}

public IceCream(String name,int scoops,double prcPerScoop,double toppings)

{

super(name);

numberOfScoops=scoops;

pricePerScoop=prcPerScoop;

toppingPrice=toppings;

}

public int getnumberOfScoops()

{

return numberOfScoops;

}

public void setnumberOfScoops(int numberOfScoops)

{

this.numberOfScoops=numberOfScoops;

}

public double getPricePerScoop()

{

return pricePerScoop;

}

public void setPricePerScoop(double pricePerScoop) {

this.pricePerScoop=pricePerScoop;

}

public double getToppingPrice()

{

return toppingPrice;

}

public void setToppingPrice(double toppingPrice)

{

this.toppingPrice=toppingPrice;

}

public double getCost()

{

double total=(numberOfScoops\*pricePerScoop+toppingPrice);

return Math.round(100\*total);

}

public String toString()

{

String s=String.format("%-50s$%.2f\n\t%dscoops@$%.2f/scoop+$%.2f",getName(),getCost()/100,numberOfScoops,pricePerScoop,toppingPrice);

return s;

}

}

DessertShop.java

package oops;

public class DessertShop {

public static void main(String arg[])

{

candy item1=new candy("Peanut Butter Fudge",2.25,3.99);

cookie item2=new cookie("Oatmeal Raisin cookies",4,3.99);

IceCream item3=new IceCream("VAnilla Ice Cream",2,1.05,0.45);

System.out.println(item1);

System.out.println(item2);

System.out.println(item3);

}

}

1. Write a Singleton class. Confirm that singleton class cannot be inherited.

Singleton.java

package oops;

public class Singleton {

private static Singleton single\_instance = null;

public String s;

private Singleton()

{

s = "Hello I am a string part of Singleton class";

}

public static Singleton getInstance()

{

if (single\_instance == null)

single\_instance = new Singleton();

return single\_instance;

}

}

Main.java

package oops;

public class main {

public static void main(String args[])

{

Singleton x = Singleton.getInstance();

Singleton y = Singleton.getInstance();

Singleton z = Singleton.getInstance();

System.out.println("Hashcode of x is " + x.hashCode());

System.out.println("Hashcode of y is " + y.hashCode());

System.out.println("Hashcode of z is " + z.hashCode());

if (x == y && y == z)

{

System.out.println("Three objects point to the same memory location on the heap i.e, to the same object");

}

else {

System.out.println("Three objects DO NOT point to the same memory

location on the heap");

}

}

}