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

**public** **class** Singleton {

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

// **TODO** Auto-generated method stub

Abc obj1 = Abc.*getInstance*();

System.***out***.println("obj1 " + obj1.toString());

}

}

**class** Abc

{

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

**private** Abc()

{

}

**public** **static** Abc getInstance()

{

**return** *obj*;

}

}

**Output:** obj1 Abc@379619aa

1. Write a program that describes the hierarchy of an organization. Here we need to write 3 classes Employee, Manager & Labour where Manager & Labour are the sub classes of the Employee. Manager has incentive & labour has over time. Add the functionality to calculate total salary of all the employees. Use polymorphism i.e., method overriding.

**class** Employees {

**public** **static** **int** *b*=10000;

**int** salary() {

**return** *b*;

}}

**class** Manager **extends** Employees{

**int** salary() {

**return** *b*+20000;

}}

**class** Labour **extends** Employees{

**int** salary()

{

**return** *b*+10000;

}

}

**public** **class** Polymorphism {

**static** **void** Salary(Employees e) {

System.***out***.println(e.salary());

}

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

Employees e1=**new** Manager();

System.***out***.println("Manager salary:");

*Salary*(e1);

Employees e2=**new** Labour();

System.***out***.println("Labour salary:");

*Salary*(e2);

}

}

**Output:** Manager salary:

30000

Labour salary:

20000

1. Write a program to consider saving & current account in the bank. Saving account holder has ‘Fixed Deposits whereas Current account holder has cash credit. Apply polymorphism to find out total cash in the bank.

**public** **class** Polymorphism1 {

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

{

bank b ;

b=**new** saving();

b.display();

b=**new** current();

b.display();

}

}

**class** bank {

**void** display()

{

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

}

}

**class** saving **extends** bank{

**void** display()

{

**int** number=50000;

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

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

}

}

**class** current **extends** bank {

**void** display()

{

**int** number=10000;

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

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

}

}

**Output:** saving account holder

Fixed deposit:50000

Current account holder

credit cash:10000

1. Test the following principles of an abstract class:

* If any class has any of its method abstract then you must declare entire class abstract.

**class** Demo {

**void** call()

{

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

}

**abstract** **void** playmusic()

{

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

}

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

// **TODO** Auto-generated method stub

Demo d1 = **new** Demo();

d1.call();

}

}

**Error:** Multiple markers at this line

-The abstract method playmusic in type Demo can only be defined by an abstract class

* Abstract class cannot be instantiated.

**abstract** **class** Demo {

**void** phone()

{

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

}

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

// **TODO** Auto-generated method stub

Demo d1 = **new** Demo();

d1.phone();

}

}

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

Cannot instantiate the type Demo

at Demo.main(Demo.java:10)

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

**class** Demo {

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

// **TODO** Auto-generated method stub

Demo2 obj= **new** Demo2();

obj.call();

obj.playmusic();

}

}

**abstract** **class** Demo1

{

**abstract** **void** call();

**abstract** **void** playmusic();

}

**class** Demo2 **extends** Demo1{

**void** call()

{

System.***out***.println("phone rings");

}

**void** playmusic()

{

System.***out***.println("playing music");

}

}

**Output:**  phone rings

playing music

* Abstract class cannot be private.

When I made the above abstract, the Output: Exception in thread “main” java.lang.Error: Unresolved compilation problems:

Illegal modifier for the class Demo1; only public, abstract & final are permitted.

* Abstract class cannot be final.

**Output:** Exception in thread “main” java.lang.Error: Unresolved compilation problems:

The class Demo1 can be either abstract or final not both

The type Demo2 cannot subclass the final class Demo1

* You can declare a class abstract without having any abstract method

Yes, we can declare a class abstract class without having any abstract method means that it can not be instantiated on its own.

1. Write a classes Line, Rectangle, Cube etc & make the Shape as their base class. Add an abstract draw() method in the class Shape & draw all shapes.

**abstract** **class** Shape {

**abstract** **void** draw();

}

**class** Line **extends** Shape{

**void** draw()

{

System.***out***.println("Drawing line");

}

}

**class** Rectangle **extends** Shape{

**void** draw() {

System.***out***.println("Drawing rectangle");

}

}

**class** Circle **extends** Shape{

**void** draw() {

System.***out***.println("Drawing circle");

}

}

**public** **class** Shapes {

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

Shape l=**new** Line();

l.draw();

Shape r=**new** Rectangle();

r.draw();

Shape c=**new** Circle();

c.draw();

}

}

**Output:** Drawing line

Drawing rectangle

Drawing circle

1. Write an abstract class ‘Persistence’ along with two subclasses ‘FilePersistence’ & ‘DatabasePersistence’. The base class with have an abstract method persist() which will be overridden by its sub classes. Write a client who gets the Persistence object at runtime & invokes persist() method on it without knowing whether data is being saved in File or in Database.

**abstract** **class** Presistence

{

**abstract** **void** presist();

}

**class** FilePresistenece **extends** Presistence{

**void** presist() {

System.***out***.println("Data Saved in File");

}

}

**class** DatabasePresistenece **extends** Presistence{

**void** presist() {

System.***out***.println("Data saved in Database");

}

}

**public** **class** Client **extends** Presistence {

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

// **TODO** Auto-generated method stub

Presistence obj = **new** Client();

obj.presist();

}

**void** presist() {

System.***out***.println("Persist method from client");

}

}

**Output:** Persist method from client

1. Develop an application for Dessert shop. The application should allow owner to add items like Candy, Cookie or IceCream in the shop storage. Also customers should be able to place an order.

Dessertitem is an abstract class having an abstract method getCost(). Every dessert item has tax associated. Candy item is solding in dollar currency, Cookie in Euro currency & iceCream in Rupees currency. The sub classes are supposed to override these methods. When we run the application, it should ask us of storage. If role is customer, then we should be able to place an order. The currency conversion rates are:

1. Dollar = 60 rupees
2. Euro = 70 rupees.

**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();

}

**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;

}

}

**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;

}

}

**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;

}

}

**public** **class** DessertShop {

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

// **TODO** Auto-generated method stub

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);

}

}

**Output:** Peanut Butter Fudge $8.980000

2.25 lbs@$.2f

Oatmeal Raisin cookies $.1.330000

4 cookies@ $3.99 per Dozen

Vanilla Ice Cream $2.55

2scoops@$1.05/scoop+$0.45