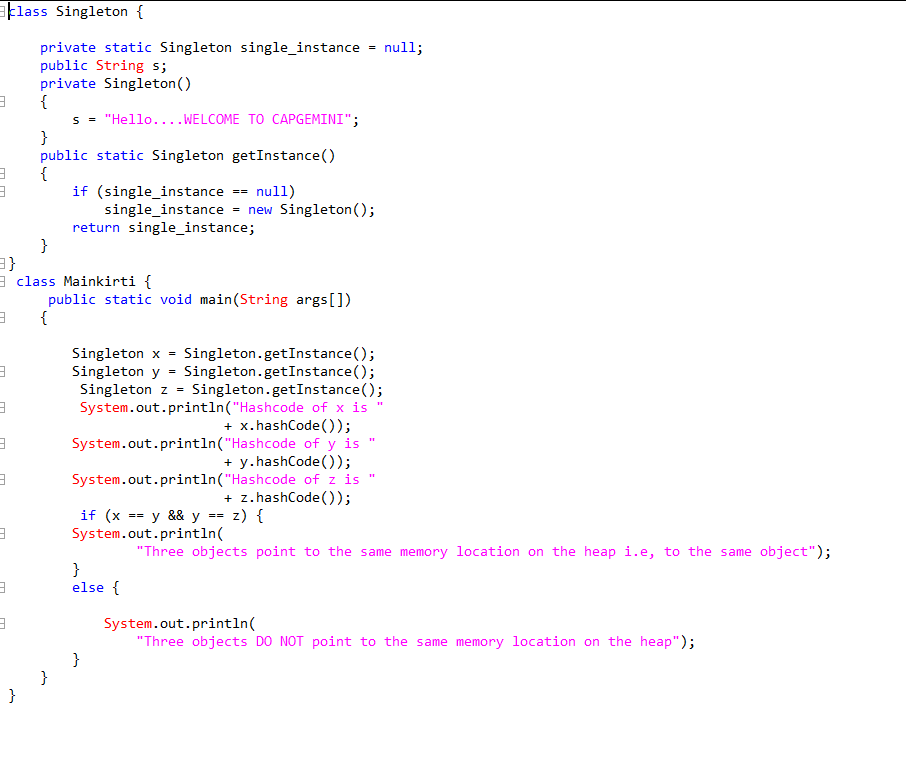
1.Singletone class confirm that singleton class cannot be inherited?



2.Write a program that describes the hierarchy of an organistion 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 totl sal of all the emp use polymorphism i.e method overriding

package Main;

public class Employee1 {

public static void main(String [] args) {

Employee e=new Employee();

e.sal();

Manager m=new Manager();

m.sal();

Labour l=new Labour();

l.sal();

e.add();

m.add();

l.add();

}

}

class Employee{

public static void add() {

int sal1=10000;

int sal2=20000;

int add=sal1+sal2;

System.out.println(add);

}

public void sal() {

System.out.println("salary");

}

}

class Manager extends Employee{

public void sal() {

int sal1=10000;

int sal2=20000;

int inciative=2000;

int add1=sal1+sal2+inciative;

System.out.println(add1);

}

}

class Labour extends Employee{

public void sal() {

int sal1=10000;

int sal2=20000;

int over=1000;

int add2=sal1+sal2+over;

System.out.println(add2);

}

}

**3rd question:-**

package Main;

public class Banking {

public static void main(String[] args) {

Bank b=new Bank();

int t= b.totalcash();

Saving s= new Saving();

int fix = s.fixeddepo();

Current c=new Current();

int cu=c.currentdeppo();

int add=fix+cu;

System.out.println("total cash..."+ add);

}

}

class Bank{

int getamount() {

return 0;

}

public static int totalcash() {

int cash=2000000;

return cash ;

}

}

class Saving extends Bank{

int getamount() {

}

public static int fixeddepo() {

int fixedamount=500000;

return fixedamount ;

}

}

class Current extends Bank{

int getamount() {

return 0;

}

public static int currentdeppo() {

int deposite = 600000;

return deposite;

}

}

**4th question:-**

ABSTRACT CLASS CANNOT BE PRIVATE

ANS: YES, BECAUSE WE CANNOT ACCESS THEM IN ANY SUBCLASS AND CANT PROVIDE IMPLEAMNETATION TO THAT

ABSTRACT CLASS

5.ABSTRACT CLASS CANNOT BE FINAL

ANS: YES, BECAUSE WE CAN ACCESS THEM IN ANY SUBCLASS BUT CAN'T PROVIDE IMPLEAMNETATION TO THAT

ABSTRACT CLASS OR CAN'T BE OVERRIDDEN.

6.WE CAN DECLARE THE CLASS AS ABSTRACT WITHOUT HAVING ABSTRACT METHOD

ANS:YES,BUT IF WE DO THAT WE CAN'T INSTATNTIATE THAT CLASS.

**5th question:-**

abstract class Shape{

abstract void draw();

}

class Line extends Shape

{

public void draw()

{

System.out.println("Draw a line of cm 20");

}

}

class Rectangle extends Shape

{

public void draw()

{

System.out.println("Draw a rectangle of length 15 breadth 25");

}

}

class Cube extends Shape

{

public void draw()

{

System.out.println("Draw a cube");

}

}

class MainShape{

public static void main(String[] args)

{

Line line = new Line();

Rectangle rectangle = new Rectangle();

Cube cube = new Cube();

line.draw();

rectangle.draw();

cube.draw();

}

}

5. WRITE THE CLASSES LINE,RECTANGLE,CUBE ETC.& DRAW ALL SHAPES:

abstract class Shape{

abstract void draw();

}

class Line extends Shape

{

public void draw()

{

System.out.println("Draw a line of cm 20");

}

}

class Rectangle extends Shape

{

public void draw()

{

System.out.println("Draw a rectangle of length 15 breadth 25");

}

}

class Cube extends Shape

{

public void draw()

{

System.out.println("Draw a cube");

}

}

class MainShape{

public static void main(String[] args)

{

Line line = new Line();

Rectangle rectangle = new Rectangle();

Cube cube = new Cube();

line.draw();

rectangle.draw();

cube.draw();

}

}

**6th question:-**

public abstract class Persistence {

public abstract void persist();

}public class FilePersistence extends Persistence{

@Override

public void persist()

{

System.out.println("Overiding in FIle persistence");

}

}

public class DatabasePersistence extends Persistence{

@Override

public void persist()

{

System.out.println("Overiding in FIle persistence");

}

}

public class Client{

public static void main(String[] args) {

Persistence persistence = new Persistence() {

@Override

public void persist() {

System.out.println("");

System.out.println("1.The client created the object of Persistance which is abstract class");

System.out.println("");

System.out.println("2.And then clients is doing following invocation of abstract method");

System.out.println("");

System.out.println("3.The client is invoking the persist method without knowing that data is saved in file and database'");

}

};

persistence.persist();

}

}

**7th question:-**

DessertItem.java

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

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;  
}

@Override  
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

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 int getQuantity() {  
return quantity;  
}

public double getPricePerDozen() {  
return pricePerDozen;  
}

public void setPricePerDozen(double pricePerDozen) {  
this.pricePerDozen = pricePerDozen;  
}

public void setQuantity(int quantity) {  
this.quantity = quantity;  
}

@Override  
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

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;  
}

@Override  
public double getCost() {  
double total = (numberOfScoops \* pricePerScoop + toppingPrice);  
return Math.round(100 \* total );  
}

public String toString()  
{  
String s = String.format("%-50s $%.2f\n\t %d scoops @ $%.2f/scoop + $%.2f", getName(), getCost()/100, numberOfScoops, pricePerScoop, toppingPrice);  
return s;  
}  
}

DessertShop.java

public class DessertShop {  
public static void main(String[] args) {  
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);

}  
}