OOPS ASSIGNMENT-A2

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

Singleton class is a class that can have only one object at time.

Answer - I created 2 Class here first is Singleton, java and another is Main.java

Class 1= Singleton.java

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

**private** **static** Singleton *single\_instance* = **null**;

**public** String str;

**private** Singleton()

{

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

}

**public** **static** Singleton getInstance()

{

**if** (*single\_instance* == **null**)

*single\_instance* = **new** Singleton();

**return** *single\_instance*;

}

}

Class 2= Main.java

**public** **class** main {

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

{

Singleton a = Singleton.*getInstance*();

Singleton b = Singleton.*getInstance*();

Singleton c = Singleton.*getInstance*();

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

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

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

**if** (a == b && b == c)

{

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");

}

}

}

Output:

Hashcode of a is 1023487453

Hashcode of b is 1023487453

Hashcode of c is 1023487453

Three objects point to the same memory location on the heap i.e, to the same object

2. Program to describe the hierarchy of an organization. To calculate the salary

Answer - I created 4 Class here which is Main.java, Employee.java, Manager.java, Labour.java

Class 1= Main.java

**package** salary.org;

**import** java.util.Scanner;

**public** **class** Main {

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

{

Manager m=**new** Manager();

m.emp();

m.incentive();

Labour l=**new** Labour();

l.emp();

l.OT();

}

}

Class 2= Employee.java

**package** salary.org;

**public** **class** Employee {

**void** emp()

{

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

}

}

Class 3= Manager.java

**package** salary.org;

**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);

}

}

Class 4= Labour.java

**package** salary.org;

**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:

5000

salary of the manager:15000

employee salary

OT:

2000

salary of the labour:12000

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

Class 1= BankSystm.java

**package** banking\_systm;

**public** **class** BankSystm {

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

{

Bank b ;

b=**new** Saving();

b.display();

b=**new** Current();

b.display();

}

}

Class 2= Bank.java

**package** banking\_systm;

**class** Bank {

**void** display()

{

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

}

}

Class 3= Saving.java

**package** banking\_systm;

**public** **class** Saving **extends** Bank{

**void** display()

{

**int** number=12000;

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

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

}

}

Class 4= Current.java

**package** banking\_systm;

**public** **class** Current **extends** Bank {

**void** display()

{

**int** number=8000;

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

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

}

}

Output:

saving account holder

Fixed deposit:12000

Current account holder

credit cash:8000

4. Test the following principle regarding abstraction

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

Class 1= Abstr.java

**package** abstract\_class;

**public** **class** Abstr **extends** Call {

**public** **void** dispaly2()

{

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

}

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

{

Abstr d= **new** Abstr();

d.display();

}

}

Class 2= Call.java

**package** abstract\_class;

**abstract** **class** Call {

**void** display()

{

System.***out***.println("Hello Word");

}

}

Output: Hello Word

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

Class 1= Abstr.java

**package** abstract\_class;

**public** **class** Abstr **extends** Call {

**public** **void** dispaly2()

{

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

}

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

{

Abstr d= **new** Abstr();

d.dispaly2();

}

}

Class 2= Call.java

**package** abstract\_class;

**abstract** **class** Call {

**void** display()

{

System.***out***.println("Hello Word");

}

}

Output: Hey

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.

7. 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.

Class 1= Item.java

**package** oops;

**public** **abstract** **class** Item {

**protected** String name;

**public** Item()

{

name="";

}

**public** Item(String name1)

{

name=name1;

}

**public** String getName()

{

**return** name;

}

**public** **void** setName(String name1)

{

name=name1;

}

**public** **abstract** **double** getCost();

}

Class 2= Candy.java

**package** oops;

**public** **class** Candy **extends** Item {

**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;

}

}

Class 3= Cookie.java

**package** oops;

**public** **class** Cookie **extends** Item {

**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$.%2f\n\t%dcookies@$%.2f per Dozen",getName(),getCost()/100,quantity,pricePerDozen);

**return** s;

}

}

Class 4= Icecream.java

**package** oops;

**public** **class** Icecream **extends** Item

{

**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;

}

}

Class 5= Shop.java

**package** oops;

**public** **class** Shop {

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

{

Candy item1=**new** Candy("Peanut Butter Fudge",3.25,5.99);

Cookie item2=**new** Cookie("Oatmeal Raisin cookies",5,4.99);

Icecream item3=**new** Icecream("Vanilla Ice Cream",3,2.05,0.75);

System.***out***.println(item1);

System.***out***.println(item2);

System.***out***.println(item3);

}

}

Output:

Peanut Butter Fudge $19.470000

3.25 lbs@$.2f

Oatmeal Raisin cookies $.2.080000

5cookies@$4.99 per Dozen

Vanilla Ice Cream $6.90

3scoops@$2.05/scoop+$0.75

6. 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