**Assignments on OOPS**

1. What is singleton class. Confirm that singleton class cannot be inherited.

* A Singleton class in Java allows only one instance to be created and provides global access to all other classes through this single object or instance.
* Cannot inherit or instantiate a **static class.**

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

class Employee

{

int sal;

String name;

Employee()

{

name = null;

sal =0;

}

Employee(int sal,String name)

{

this.sal = sal;

this.name= name;

}

int getsal()

{

return sal;

}

}

class Manager extends Employee

{

int hra, ta;  
     Manager()  
     {  
          super();  
          hra = ta = 0;  
     }  
     Manager(String n, int sal, int h, int t)  
     {  
          super(n, sal);  
          hra = h;  
          ta = t;  
     }  
     int getSalary()  
     {  
          return (super.getSalary()+hra+ta);  
     }  
}

class Labour extends Employee

{ int hra, ta;  
     Labour()  
     {  
          super();  
          hra = ta = 0;  
     }  
     Labour(String n, int sal, int h, int t)  
     {  
          super(n, sal);  
          hra = h;  
          ta = t;  
     }  
     int getSalary()  
     {  
          return (super.getSalary()+hra+ta);  
     }  
}

public class Main  
{  
     public static void main(String[] args)  
     {  
          Manager m1 = new Manager("Dineshwari",25000,500,1500);  
          System.out.println("Salary of Manager= "+m1.getSalary());  
     }  
}

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.

class saving {

|  |
| --- |
|  |
|  |
|  | public void find(int f) { |
|  | System.out.println("Fixed Deposit "+f); |
|  | } |
|  | } |
|  | class Current{ |
|  | public void find(int c) { |
|  | System.out.println("Cash Credit "+c); |
|  | } |
|  | } |
|  | class Totalcash{ |
|  | public void find(int ssal,int csal) { |
|  | int total; |
|  | total=ssal+csal; |
|  | System.out.println("Total Cash : "+total); |
|  | } |
|  | } |
|  | public class second { |
|  |  |
|  | public static void main(String[] args) { |
|  |  |
|  | int fd,cd; |
|  | Saving s = new Saving(); |
|  | Current cu = new Current(); |
|  | Totalcash t= new Totalcash(); |
|  |  |
|  | Scanner s = new Scanner(System.in); |
|  | System.out.println("Fixed deposit"); |
|  | System.out.print("Enter Fixed deposit for savings account: "); |
|  | fd = s.nextInt(); |
|  |  |
|  | System.out.print("Enter Cash credit for Current account: "); |
|  | cd = s.nextInt(); |
|  | s.find(fd); |
|  | cu.find(cd); |
|  | t.find(fd,cd); |
|  |  |
|  | } |
|  |  |
|  | }   1. Test the following principles of abstract class  * If any class has any one of its method abstract then you must declare entire class abstract.   abstract class abstractclass  {  public void disp()  {  System.out.println(“Abstract class”);  }  }  public class second extends abstractclass{  public static void main(String[] args)  {  Abstractclass a = new second();  a.disp();  }  }  }   * Abstract class cannot be instantiated   Abstract classes cannot be instantiated, but they can be subclassed. When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract .   * When we extend an abstract class, we must either override all the abstract methods in sub class or declare subclass as abstract    A subclass must override all abstract methods of an abstract class. However, if the subclass is declared abstract, it's not mandatory to override abstract methods.   * Abstract class cannot be private   If a method of a class is private, cannot access it outside the current class, not even from the child classes of it. But, incase of an abstract method,cannot use it from the same class, need to override it from subclass and use. Therefore, the abstract method cannot be private.     * Abstract class cannot be final   **No,** An abstract class must be inherited by any derived class because a derived class is responsible to provide the implementation of abstract methods of an abstract class.   * You can declare a class abstract without having any abstract method.   **Yes**, we can declare an abstract class with no abstract methods in Java. An abstract class means that hiding the implementation and showing the function definition to the user.   1. Write the 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 Cube extends Shape{ | |  | void draw(){System.out.println("Drawing cube");} | |  | } | |  |  | |  | class third{ | |  | public static void main(String args[]){ | |  |  | |  | Shape s1=new Line(); | |  | Shape s2=new Rectangle(); | |  | Shape s3=new Cube(); | |  | s1.draw(); | |  | s2.draw(); | |  | s3.draw(); | |  |  | |  | } | |  | } | |
|  |  |

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 persisitence object at runtime & incokes persisit() method on it without knowing whether data is being saved in File or in Database.

|  |
| --- |
|  |
|  | abstract class Persistence{  abstract void persist(); |
|  | } |
|  | class FilePersistenece extends Persistence{ |
|  |  |
|  |  |
|  | void persist() { |
|  |  |
|  | System.out.println("Data Saved in File"); |
|  | } |
|  |  |
|  | } |
|  | class DatabasePersistenece extends Persistence{ |
|  |  |
|  |  |
|  | void persist() { |
|  |  |
|  | System.out.println("Data saved in Database"); |
|  | } |
|  |  |
|  | } |
|  | public class fourth extends Persistence{ |
|  |  |
|  | public static void main(String[] args) { |
|  |  |
|  | Persistence p=new fourth(); |
|  | p.persist(); |
|  | } |
|  |  |
|  |  |
|  | void persist() { |
|  |  |
|  | System.out.println("Persistence method"); |
|  |  |
|  | } |
|  | } |
|  |  |

1. Develop an application for Dessert shop.

|  |  |
| --- | --- |
|  | abstract class DessertItem{  abstract void getCost(); |
|  | } |
|  | class Candy extends DessertItem{ |
|  |  |
|  |  |
|  | void getCost() { |
|  |  |
|  | double v=0; |
|  | int oqty, amt[]= {20,30,40}; |
|  | double tax=0.2; |
|  | System.out.println("Candy dessert-Tax : "+tax); |
|  | Scanner s = new Scanner(System.in); |
|  | System.out.println("Enter quantity:"); |
|  | oqty=s.nextInt(); |
|  | v = amt[0]\*oqty\*20\*tax; |
|  | System.out.println("Your order has been placed!! It costs :"+val+" dollar"); |
|  | }  } |
|  | class Cookie extends DessertItem{ |
|  |  |
|  |  |
|  | void getCost() { |
|  |  |
|  | double v=0; |
|  | int oqty, amt[]= {20,30,40}; |
|  | double tax=0.2; |
|  | System.out.println("Cookie dessert-Tax : "+tax); |
|  | Scanner s = new Scanner(System.in); |
|  | System.out.println("Enter quantity:"); |
|  | oqty=s.nextInt(); |
|  | v = amt[0]\*oqty\*20\*tax; |
|  | System.out.println("Your order has been placed!! It costs :"+val+" euro"); |
|  | }  } |
|  | class Icecream extends DessertItem{ |
|  |  |
|  |  |
|  | void getCost() { |
|  |  |
|  | double v=0; |
|  | int oqty, amt[]= {20,30,40}; |
|  | double tax=0.2; |
|  | System.out.println("Icecream dessert-Tax : "+tax); |
|  | Scanner s = new Scanner(System.in); |
|  | System.out.println("Enter quantity:"); |
|  | oqty=s.nextInt(); |
|  | v = amt[0]\*oqty\*20\*tax; |
|  | System.out.println("Your order has been placed!! It costs :"+val+" rupees"); |
|  | } |
|  | }  public class fourth extends DessertItem{  public static void main(String[] args) {   |  | | --- | |  | |  | |

String dessert[]={“Candy”,”Cookie”,”Icecream”};

int amt[]={20,30,25};

int qty[]={10,20,30};

DessertItem d= new fourth();

DessertItem ca=new Candy();

DessertItem co=new Cookie();

DessertItem ic=new Icecream();

Scanner s=new Scanner(System.in);

int f;

do

{

System.out.println(“DESSERT SHOP WELCOMES”);

System.out.println(“a.Owner \n b.Customer”);

c=s.next().charAt(0);

switch(c)

{

case ‘a’:

System.out.println(“Dessert need to add: ”);

f=s.nextInt();

System.out.println(dessert[f-1]+" Quantity Available is: "+qty[f-1]);

System.out.println("Enter the quantity to add:");

qty[f-1]=qty[f-1]+sc.nextInt();

System.out.println("Candy : "+qty[0]+"Cookie : "+qty[1]+"Icecream : "+qty[2]);

System.out.println(“Dessert added!!!”);

}

break;

case ‘b’ :

System.out.println(“Customer”);

System.out.println(“Dessert want to place order:”);

System.out.println(“Enter choice:”);

f=s.nextInt();

switch(f)

{

case 1:

System.out.println(“Candy”);

System.out.println(“Candy quantity available is ” +qty[f-1]);

System.out.println(“Per Candy costs : "+amt[f-1]+"dollars”);

ca.getcost();

break;

case 2:

System.out.println(“Cookie”);

System.out.println(“Cookie quantity available is ” +qty[f-1]);

System.out.println(“Per Cookie costs : "+amt[f-1]+"euro”);

co.getcost();

break;

case 3:

System.out.println(“Icecream”);

System.out.println(“Icecream quantity available is ” +qty[f-1]);

System.out.println(“Per Icecream costs : "+amt[f-1]+"rupees”);

ic.getcost();

break;

}

}}

while(ch!=2);

}

void getcost()

{

System.out.println(“Main”);

}

}