**Java Interfaces**

Consider a Banking Scenario, There are many accounts, like Savings Account, Current Account, Demat Account and so on. We have a base Class Account which contains all the basic properties and methods of an Account. We do have some Maintainance Charges that apply to only some of the accounts. If you would like to enforce that the Savings Account & Current Account should have maintenance charges, then the simplest way is to ask your class implement the interface. If you do not implement the method in the class, it would raise a compilation error.

So, Java Interfaces essentially gives acts like a contract where its given that the methods declared in the interface has to be implemented in the class. Lets code the above Scenario.

Create a class named **Account** with following private attributes

|  |  |
| --- | --- |
| **Attribute** | **Datatype** |
| name | String |
| accountNumber | String |
| balance | double |
| startDate | String |

Include appropriate getters and setters.

Include appropriate parameterized constructors

Create an interface named **MaintenanceCharge** with the following method

|  |  |
| --- | --- |
| **Method** | **Description** |
| float calculateMaintancecharge(float no of years) | This method is used to calculate the maintenance charge |

Create a class named **CurrentAccount** which implements **MaintenanceCharge** interface  
Create a class named **SavingsAccount** which implements **MaintenanceCharge** interface

Create an another class named **Main** with main( ) method to test the above classes.

In Savings Account the maintenance amount will be 2\*m\*n+50.

In Current Account, the maintenance amount will be m\*n+200.

where m is the maintenance charge per year and n is the number of years.

Maintenance charge Rs.50  for saving account and 100 for the Current account.

**Note: Refere sample input and output for the specifications.**

**All text in bold corresponds to the input and remaining text corresponds to output.**

**Sample input and output 1:**

1.Current Account  
2.Savings Account  
**1**  
Name  
**SB**  
Account Number  
**12345**  
Account Balance  
**5000**  
Enter the Start Date(yyyy-mm-dd)  
**2013-04-22**  
Enter the Years  
**2**  
Maintenance Charge For Current Account 400.00

**Sample input and output 2:**  
1.Current Account  
2.Savings Account  
**2**  
Name  
**SB**  
Account Number  
**54321**  
Account Balance  
**3000**  
Enter the Start Date(yyyy-mm-dd)  
**2014-04-12**  
Enter the Years  
**5**  
Maintenance Charge For Savings Account 550.00

public class CurrentAccount implements MaintainanceCharge {

@Override

public float calculateMaintanceCharge(int noOfYear) {

return (100 \* noOfYear) + 200;

}

}

class Account {

String name;

String accountNumber;

double balance;

String startDate;

public Account(String name, String accountNumber, double balance, String startDate) {

super();

this.name = name;

this.accountNumber = accountNumber;

this.balance = balance;

this.startDate = startDate;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAccountNumber() {

return accountNumber;

}

public void setAccountNumber(String accountNumber) {

this.accountNumber = accountNumber;

}

public double getBalance() {

return balance;

}

public void setBalance(double balance) {

this.balance = balance;

}

public String getStartDate() {

return startDate;

}

public void setStartDate(String startDate) {

this.startDate = startDate;

}

}

public interface MaintainanceCharge{

float calculateMaintanceCharge(int noOfYear);

}

class SavingsAccount implements MaintainanceCharge {

@Override

public float calculateMaintanceCharge(int noOfYear) {

return (2 \* 50 \* noOfYear) + 50;

}

}

import java.text.DecimalFormat;

import java.util.\*;

public class Main {

public static void main(String args[]) {

Scanner s = new Scanner(System.in);

String name, sdate, account;

double accountBalance;

int years;

System.out.println("1.Current Account");

System.out.println("2.Savings Account");

int n = s.nextInt();

DecimalFormat df = new DecimalFormat("0.00");

switch (n) {

case 1:

System.out.println("Name");

name = s.next();

System.out.println("Account Number");

account = s.next();

System.out.println("Account Balance");

accountBalance = s.nextInt();

s.nextLine();

System.out.println("Enter the Start Date(yyyy-mm-dd)");

sdate = s.nextLine();

System.out.println("Enter the Years");

years = s.nextInt();

Account a = new Account(name, account, accountBalance, sdate);

CurrentAccount c = new CurrentAccount();

System.out

.println("Maintenance Charge For Current Account " + df.format(c.calculateMaintanceCharge(years)));

break;

case 2:

System.out.println("Name");

name = s.next();

System.out.println("Account Number");

account = s.next();

System.out.println("Account Balance");

accountBalance = s.nextInt();

s.nextLine();

System.out.println("Enter the Start Date(yyyy-mm-dd)");

sdate = s.nextLine();

System.out.println("Enter the Years");

years = s.nextInt();

Account a1 = new Account(name, account, accountBalance, sdate);

SavingsAccount sa = new SavingsAccount();

System.out

.println("Maintenance Charge For Savings Account " + df.format(sa.calculateMaintanceCharge(years)));

break;

default:

System.out.println("Invalid choice");

break;

}

}

}

**Interface**

The Interface defines a rule that any classes that implement it should override all the methods. Let's implement Interface in our application. We'll start simple, by including display method in the Stall interface. Now all types of stalls that implement the interface should override the method.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**  
  
Create an interface **Stall**with the following method

|  |  |
| --- | --- |
| **Method** | **Description** |
| void display() | abstract method. |

Create a class **GoldStall**which implements **Stall**interface with the following private attributes

|  |  |
| --- | --- |
| **Attribute** | **Datatype** |
| stallName | String |
| cost | Integer |
| ownerName | String |
| tvSet | Integer |

Create default constructor and a parameterized constructor with arguments in order **GoldStall(String stallName, Integer cost, String ownerName, Integer tvSet)**.  
Include appropriate getters and setters.  
  
Include the following method in the class **GoldStall**

|  |  |
| --- | --- |
| **Method** | **Description** |
| void display() | To display the stall name, cost of the stall, owner name and the number of tv sets. |

Create a class **PremiumStall**which implements **Stall**interface with following private attributes

|  |  |
| --- | --- |
| **Attribute** | **Datatype** |
| stallName | String |
| cost | Integer |
| ownerName | String |
| projector | Integer |

Create default constructor and a parameterized constructor with arguments in order **PremiumStall(String stallName, Integer cost, String ownerName, Integer projector)**.  
Include appropriate getters and setters.  
  
Include the following method in the class **PremiumStall.**

|  |  |
| --- | --- |
| **Method** | **Description** |
| void display() | To display the stall name, cost of the stall, owner name and the number of projectors. |

Create a class **ExecutiveStall**which implements **Stall**interface with following private attributes

|  |  |
| --- | --- |
| **Attribute** | **Datatype** |
| stallName | String |
| cost | Integer |
| ownerName | String |
| screen | Integer |

Create default constructor and a parameterized constructor with arguments in order **ExecutiveStall(String stallName, Integer cost, String ownerName, Integer screen)**.  
Include appropriate getters and setters.  
  
Include the following method in the class **ExecutiveStall.**

|  |  |
| --- | --- |
| **Method** | **Description** |
| void display() | To display the stall name, cost of the stall, owner name and the number of screens. |

Create a driver class named **Main** to test the above class.  
  
**Input Format:**  
The first input corresponds to choose the stall type.  
The next line of input corresponds to the details of the stall in CSV format according to the stall type.  
  
**Output Format:**  
Print “**Invalid Stall Type**” if the user has chosen the stall type other than the given type  
Otherwise, display the details of the stall.  
Refer to sample output for formatting specifications.  
  
**Note:** **All Texts in bold corresponds to the input and rest are output**  
  
**Sample Input and Output 1:**  
  
Choose Stall Type  
1)Gold Stall  
2)Premium Stall  
3)Executive Stall  
**1**  
Enter Stall details in comma separated(Stall Name,Stall Cost,Owner Name,Number of TV sets)  
**The Mechanic,120000,Johnson,10**  
Stall Name:The Mechanic  
Cost:120000.Rs  
Owner Name:Johnson  
Number of TV sets:10  
  
**Sample Input and Output 2:**  
  
ChooseStall Type  
1)Gold Stall  
2)Premium Stall  
3)Executive Stall  
**2**  
Enter Stall details in comma separated(Stall Name,Stall Cost,Owner Name,Number of Projectors)  
**Knitting plaza,300000,Zain,20**  
Stall Name:Knitting plaza  
Cost:300000.Rs  
Owner Name:Zain  
Number of Projectors:20  
  
**Sample Input Output 3:**  
  
ChooseStall Type  
1)Gold Stall  
2)Premium Stall  
3)Executive Stall  
**3**  
Enter Stall details in comma separated(Stall Name,Stall Cost,Owner Name,Number of Screens)  
**Fruits Hunt,10000,Uber,7**  
Stall Name:Fruits Hunt  
Cost:10000.Rs  
Owner Name:Uber  
Number of Screens:7  
  
**Sample Input Output 4:**  
  
ChooseStall Type  
1)Gold Stall  
2)Premium Stall  
3)Executive Stall  
**4**  
Invalid Stall Type

public class PremiumStall implements Stall {

private String stallName;

private Integer cost;

private String ownerName;

private Integer projector;

public PremiumStall(String stallName, Integer cost, String ownerName, Integer projector) {

this.stallName = stallName;

this.cost = cost;

this.ownerName = ownerName;

this.projector = projector;

}

public PremiumStall() {

}

public void display() {

System.out.println("Stall Name:" + stallName + "\nCost:" + cost + ".Rs\nOwner Name:" + ownerName

+ "\nNumber of Projectors:" + projector);

}

public String getStallName() {

return stallName;

}

public void setStallName(String stallName) {

this.stallName = stallName;

}

public Integer getCost() {

return cost;

}

public void setCost(Integer cost) {

this.cost = cost;

}

public String getOwnerName() {

return ownerName;

}

public void setOwnerName(String ownerName) {

this.ownerName = ownerName;

}

public Integer getProjector() {

return projector;

}

public void setProjector(Integer projector) {

this.projector = projector;

}

}

public class ExecutiveStall implements Stall {

private String stallName;

private Integer cost;

private String ownerName;

private Integer screen;

public ExecutiveStall(String stallName, Integer cost, String ownerName, Integer screen) {

this.stallName = stallName;

this.cost = cost;

this.ownerName = ownerName;

this.screen = screen;

}

public ExecutiveStall() {

}

public void display() {

System.out.println("Stall Name:" + stallName + "\nCost:" + cost + ".Rs\nOwner Name:" + ownerName

+ "\nNumber of Screens:" + screen);

}

public String getStallName() {

return stallName;

}

public void setStallName(String stallName) {

this.stallName = stallName;

}

public Integer getCost() {

return cost;

}

public void setCost(Integer cost) {

this.cost = cost;

}

public String getOwnerName() {

return ownerName;

}

public void setOwnerName(String ownerName) {

this.ownerName = ownerName;

}

public Integer getScreen() {

return screen;

}

public void setScreen(Integer screen) {

this.screen = screen;

}

}

public interface Stall {

void display();

}

import java.util.Scanner;

public class Main {

public static void main(String[] args) throws Exception {

// BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String[] arr = new String[3];

String stallDetails = "";

System.out.println("Choose Stall Type\n1)Gold Stall\n2)Premium Stall\n3)Executive Stall");

Scanner scan = new Scanner(System.in);

int n = scan.nextInt();

switch (n) {

case 1:

scan.nextLine();

System.out.println(

"Enter Stall details in comma separated(Stall Name,Stall Cost,Owner Name,Number of TV sets)");

stallDetails = scan.nextLine();

arr = stallDetails.split(",");

int cost = Integer.parseInt(arr[1]);

int num = Integer.parseInt(arr[3]);

GoldStall gStall = new GoldStall(arr[0], cost, arr[2], num);

gStall.display();

break;

case 2:

scan.nextLine();

System.out.println(

"Enter Stall details in comma separated(Stall Name,Stall Cost,Owner Name,Number of Projectors)");

stallDetails = scan.nextLine();

arr = stallDetails.split(",");

cost = Integer.parseInt(arr[1]);

num = Integer.parseInt(arr[3]);

PremiumStall pStall = new PremiumStall(arr[0], cost, arr[2], num);

pStall.display();

break;

case 3:

scan.nextLine();

System.out.println(

"Enter Stall details in comma separated(Stall Name,Stall Cost,Owner Name,Number of Screens)");

stallDetails = scan.nextLine();

arr = stallDetails.split(",");

cost = Integer.parseInt(arr[1]);

num = Integer.parseInt(arr[3]);

ExecutiveStall eStall = new ExecutiveStall(arr[0], cost, arr[2], num);

eStall.display();

break;

default:

System.out.println("Invalid Stall Type");

}

}

}

public class GoldStall implements Stall {

private String stallName;

private Integer cost;

private String ownerName;

private Integer tvSet;

public GoldStall(String stallName, Integer cost, String ownerName, Integer tvSet) {

this.stallName = stallName;

this.cost = cost;

this.ownerName = ownerName;

this.tvSet = tvSet;

}

public GoldStall() {

}

public void display() {

System.out.println("Stall Name:" + stallName + "\nCost:" + cost + ".Rs\nOwner Name:" + ownerName

+ "\nNumber of TV sets:" + tvSet);

}

public String getStallName() {

return stallName;

}

public void setStallName(String stallName) {

this.stallName = stallName;

}

public Integer getCost() {

return cost;

}

public void setCost(Integer cost) {

this.cost = cost;

}

public String getOwnerName() {

return ownerName;

}

public void setOwnerName(String ownerName) {

this.ownerName = ownerName;

}

public Integer getTvSet() {

return tvSet;

}

public void setTvSet(Integer tvSet) {

this.tvSet = tvSet;

}

}

**Round up - Interfaces**

In one of the earlier exercises in Interfaces, we looked at how banks encrypt & decrypt transaction details to ensure safety. We also saw understood that each of them complies with the methods specified by the Governing Agency in the form of an interface. To Round-off interfaces, let's look at a similar example, where the governing agency mandates that any customer who performs a transaction has to be notified through SMS, Email and a monthly e-statement. As expected, we define an interface Notification and three methods as specified below. Lets code this example.

Create an interface named **Notification** with the following methods

    notificationBySms( ),  
    notificationByEmail( ),  
    notificationByCourier( ).

Create a class named **ICICI** which implements **Notification**interface

Create a class named **HDFC** which implements **Notification**interface  
  
Create a class **BankFactory**with two methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| public Icici getIcici( ) | This method is used to return the object for ICICI class |
| public Hdfc getHdfc( ) | This method is used to return the object for HDFC class |

Create the **Main**class with main( ) method to test the above class.  
  
**Input and Output format:**  
The first integer corresponds to select the bank, the next integer corresponds to the type of the notification.  
If there is no valid input then display '**Invalid input**'.

**[Note: All text in bold corresponds to the input and remaining text corresponds to output]**  
  
**Sample Input and Output 1:**  
  
Welcome to Notification Setup  
Please select your bank:  
1)ICICI  
2)HDFC  
**1**  
Enter the type of Notification you want to enter  
1)SMS  
2)Mail  
3)Courier  
**1**  
ICICI - Notification By SMS  
  
**Sample Input and Output 2:**  
  
Welcome to Notification Setup  
Please select your bank:  
1)ICICI  
2)HDFC  
**2**  
Enter the type of Notification you want to enter  
1)SMS  
2)Mail  
3)Courier  
**3**  
HDFC - Notification By Courier  
  
**Sample Input and Output 3:**  
  
Welcome to Notification Setup  
Please select your bank:  
1)ICICI  
2)HDFC  
**3**  
Invalid Input

public class ICICI implements Notification {

@Override

public void notificationBySms() {

System.out.println("ICICI - Notification By SMS");

}

@Override

public void notificationByEmail() {

System.out.println("ICICI - Notification By Mail");

}

@Override

public void notificationByCourier() {

System.out.println("ICICI - Notification By Courier");

}

}

public class HDFC implements Notification {

@Override

public void notificationBySms() {

System.out.println("HDFC - Notification By SMS");

}

@Override

public void notificationByEmail() {

System.out.println("HDFC - Notification By Mail");

}

@Override

public void notificationByCourier() {

System.out.println("HDFC - Notification By Courier");

}

}

public interface Notification {

public void notificationBySms();

public void notificationByEmail();

public void notificationByCourier();

}

public class BankFactory {

public ICICI getIcici() {

ICICI icici = new ICICI();

return icici;

}

public HDFC getHdfc() {

HDFC hdfc = new HDFC();

return hdfc;

}

}

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Scanner;

public class Main {

public static void main(String[] args) throws IOException {

Scanner sc = new Scanner(System.in);

System.out.println("Welcome to Notification Setup");

System.out.println("Please select your bank:\n1)ICICI\n2)HDFC");

int bank = sc.nextInt();

switch (bank) {

case 1:

System.out.println("Enter the type of Notification you want to enter\n1)SMS\n2)Mail\n3)Courier");

int msg = sc.nextInt();

BankFactory b = new BankFactory();

ICICI c = b.getIcici();

switch (msg) {

case 1:

c.notificationBySms();

break;

case 2:

c.notificationByEmail();

break;

case 3:

c.notificationByCourier();

break;

default:

System.out.println("Invalid Input");

break;

}

break;

case 2:

System.out.println("Enter the type of Notification you want to enter\n1)SMS\n2)Mail\n3)Courier");

int msg1 = sc.nextInt();

BankFactory b1 = new BankFactory();

HDFC h = b1.getHdfc();

switch (msg1) {

case 1:

h.notificationBySms();

break;

case 2:

h.notificationByEmail();

break;

case 3:

h.notificationByCourier();

break;

default:

System.out.println("Invalid Input");

break;

}

break;

default:

System.out.println("Invalid Input");

break;

}

}

}

**Static Inner Class**

Write a program to calculate the area of the rectangle and triangle using the static inner class concept in java.

Create an outer class **Shape** with the following **public static**attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| value1 | Double |
| value2 | Double |

Create a static inner class **Rectangle** which have the outer class **Shape.**  
Include the following method in the **Rectangle** class

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| public Double computeRectangleArea() | Here Calculate and return the area of the rectangle by accessing the attributes value1(length) & value2(breadth) of Shape class. Area of the rectangle = (length \* breadth) |

Create a static inner class **Triangle** which have the outer class **Shape.**  
Include the following method in the **Triangle** class

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| public Double computeTriangleArea() | Here Calculate and return the area of the triangle by accessing the attributes value1(base) & value2(height) of Shape class. Area of the triangle = (1/2) \* (base \* height) |

Get the option for the shape to compute the area and get the attribute according to the shape option and set the values to the Shape class attributes. Calculate the area and print the area.  
While printing round off the area to 2 decimal formats.  
  
Create a driver class **Main** to test the above classes.  
  
**[Note: Strictly adhere to the object-oriented specifications given as a part of the problem statement. Use the same class names, attribute names and method names]**  
  
**Input Format**  
The first line of the input is an integer corresponds to the shape.  
The next line of inputs are Double which corresponds to,  
For Rectangle(Option 1) get the length and breadth.  
For Triangle(Option 2) get the base and height.  
  
**Output Format**  
The output consists area of the shape.  
Print the double value correct to two decimal places.  
Print “**Invalid choice**”, if the option for the shape is chosen other than the given options.  
Refer to sample output for formatting specifications.  
  
**[All text in bold corresponds to input and rest corresponds to output]  
Sample Input/Output 1:**  
Enter the shape  
1.Rectangle  
2.Triangle  
**1**  
Enter the length and breadth:  
**10  
25**  
Area of rectangle is 250.00  
  
**Sample Input/Output 2:**  
Enter the shape  
1.Rectangle  
2.Triangle  
**2**  
Enter the base and height:  
**15  
19**  
Area of triangle is 142.50  
  
**Sample Input/Output 3:**  
Enter the shape  
1.Rectangle  
2.Triangle  
**3**  
Invalid choice

Top of Form

Bottom of Form

public class Shape {

public static Double value1;

public static Double value2;

public static class Rectangle {

public Double computeRectangleArea() {

return (value1 \* value2);

}

}

public static class Triangle {

public Double computeTriangleArea() {

return ((0.5) \* (value1 \* value2));

}

}

}

import java.io.IOException;

import java.text.DecimalFormat;

import java.util.Scanner;

public class Main {

public static void main(String[] args) throws IOException {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the shape\r\n1.Rectangle\r\n2.Triangle");

int ch = sc.nextInt();

DecimalFormat df = new DecimalFormat("0.00");

switch (ch) {

case 1:

System.out.println("Enter the length and breadth:");

Shape.value1 = sc.nextDouble();

Shape.value2 = sc.nextDouble();

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

Double rectangle = r.computeRectangleArea();

System.out.println("Area of rectangle is " + df.format(rectangle));

break;

case 2:

System.out.println("Enter the base and height:");

Shape.value1 = sc.nextDouble();

Shape.value2 = sc.nextDouble();

Shape.Triangle t = new Shape.Triangle();

Double triangle = t.computeTriangleArea();

System.out.println("Area of triangle is " + df.format(triangle));

break;

default:

System.out.println("Invalid choice");

break;

}

}

}