1). Consider a scenario where a customer who owns a Credit Card (containing Indian currency only) purchases some products from three different shops A, B, and C. The shops A, B, and C accept only **Rupees**, **Dollar** and **Pound** currency, respectively. Suppose that the Bank is the only authority to manage and convert the currency. Create an interface with abstract functions convertRupees, convertDollars, and convertPounds in java. Create a **Bank** class which implements the interface and describes all the functions for converting the currency and returning the outstanding balance of the customer after decreasing the price of the purchased products.

Assume 1 dollar is equal to 70 rupees and 1 pound is equal to 100 rupees.

Note : If balance is less than the total purchase amount, then print "Insufficient balance"

Input Format :

First line specifies the total balance of the Credit Card in Rupees
Second line is given as a space separated list of prices of all different products.

- Price of product from shop A which accepts Rupee currency.
- Price of product from shop B which accepts Dollar currency.
- Price of product from shop C which accepts Pound currency.

Output Format :

One line indicating the outstanding balance of the Credit Card after purchase(in Rupees)

Sample Input :

10000
800 50 40

Sample Output :

1700

2). Consider the following scenario: Flipkart wants to integrate HDFC and ICICI banks into their shopping cart so that their customers can initiate payment through HDFC or ICICI bank. Since Flipkart does not possess its own bank, Flipkart needs to implement the required functions from these banks to perform customers' transactions. Create two interfaces (**HDFC** and **ICICI**) with a transaction function in java. Create a class **Flipkart** that implements both the interfaces and override their function in order to perform the transactions and reflect the total amount added to Flipkart's account and the remaining balance in the customer's account.

Input Format :

First line specifies the number of transactions, n.
Each of the n lines that follow is space separated list of the following transaction details:

- Name of the bank
- Customer Bank account balance
- Product price

Output Format :

Two space separated list:

- Total amount of Flipkart account (initial balance is 0)
- Remaining balance of customer's account

Sample Input :

```
HDFC  3000  400
ICICI   5000   300
HDFC  10000  4000
```

Sample Output :

```
400  2600
700  4700
4700  6000
```

3). Create an interface '**ShapeArea'** with an abstract method **area()** that takes a single argument. Create two different classes **Circle** and **Square** which implement the ShapeArea interface and compute the area of the respective shape. (Assume pi=3.14)

Input format:

2 lines of input:

- Name of the shape
- A value representing the radius/length of the corresponding shape.

Output format:

- Area of the shape

Sample Input1:

Circle
10

Sample Output1:

314

Sample Input2:

Square
30

Sample Output2:

900

4). Write a Java program to implement voting in SAC elections and display the results with the support of interfaces. Create a class **ElectionPost**, which has attributes **nameOfthePost** (String)**, listOfCandidates** (ArrayList of String) and **votesEntered** (ArrayList of String). There is a ***display***

method in the class which displays the name of the post and the election candidate's name (in Ascending order of their name). There is an interface **SortVotes** which has two methods **ascendDisplay** and **descendDisplay** which should be implemented in the class to display the details by sorting the candidates according to their count of votes in ascending order and descending order respectively.

Hint: For sorting the options, you can make use of the sort method in java.util package, Collections.sort() with suitable arguments.

Input Format:

- Line 1: Enter the name of the post
- Line 2: Number of candidates, n
- Next n lines, enter the name of each candidate
- Next line: Total number of votes, m
- Next m lines, enter the name of the candidate who received the vote

Output Format:

The output displays the following details:

- Name of the post and the list of candidates in separate lines
- Name and total number of votes received by each candidate in ascending order of number of votes
- Name and total number of votes received by each candidate in descending order of number of votes

Sample Input:

```
General Secretary
3
SALEENA JOSEPH
MANU J
ASHISH TOBY
14
SALEENA JOSEPH
SALEENA JOSEPH
MANU J
MANU J
SALEENA JOSEPH
SALEENA JOSEPH
SALEENA JOSEPH
ASHISH TOBY
ASHISH TOBY
ASHISH TOBY
SALEENA JOSEPH
SALEENA JOSEPH
SALEENA JOSEPH
SALEENA JOSEPH
```

Sample Output:

General Secretary
SALEENA JOSEPH
MANU J
ASHISH TOBY
MANU J - 2
ASHISH TOBY - 3
SALEENA JOSEPH - 9
SALEENA JOSEPH - 9
ASHISH TOBY - 3
MANU J - 2

5)**.** Create a class **Student** with attributes ***name, rollno,*** *and* ***address*** in java. Create an interface **OrderStudents** with two methods named **OrderByRollno** and **OrderByName**. Implement these methods in the class Student to display the student details by sorting the students according to the ascending order of their roll numbers and names, respectively. For sorting you can use the sort method in java.util package.

Input Format :

First line specified 'n' number of students.
Next n line is given as a space separated list of students details:

- Name of student
- Roll No of student
- Address of student

Output Format :

Sorted list of students based on roll no.
Sorted list of students based on student name.

Sample Input :

111 Babita Hyderabad
131 Ajay Banglore
121 Ritika Jaipur

Sample Output :

//Sorted by Rollno
111 Babita Hyderabad
121 Ritika Jaipur
131 Ajay Banglore

//Sorted by Name
131 Ajay Banglore
111 Babita Hyderabad

121 Ritika Jaipur

6). There is a class **Book**, which has the attributes ***bookID***, ***title*** and ***author***. There are two categories of books, **TextBooks** and **ReferenceBooks**. The text books can be borrowed by a user while the reference books cannot be. There is an extra attribute ***status*** (default value is Available) and ***borrowedUser*** in the class TextBooks. There is an interface **Borrowable** which has methods ***checkIn*** and ***checkOut***. The class Book implements the **Borrowable** interface (Think and decide whether class **Book** is an abstract class or not).

The functionality of the *checkIn* and *checkOut* methods in TextBooks class is as follows:
checkIn : should set ***status*** attribute as Borrowed and should set the value of borrowedUser.
checkOut : should set ***status*** attribute as Available

The functionality of the *checkIn* and *checkOut* methods in ReferenceBooks class is as follows:
checkIn : display "Invalid"
*checkOut : display "*Cannot be borrowed"

Sample Input and Output:

1. Add Reference Book
2. Add Text Book
3. Check-Out
4. Check-In
5. List Books
6. Exit

Enter your choice: 1
Enter ID, Title and Author (Line by line)
101
Data Structures and Algorithms
Cormen

Enter your choice: 2
Enter ID, Title and Author (Line by line)
102
Programming Ruby
Thomas

Enter your choice: 5
ReferenceBook: 101: Data Structures and Algorithms: Cormen
TextBook: 102: Programming Ruby: Thomas: Available

Enter your choice: 3
Enter Book ID: 101
Cannot be borrowed

Enter your choice: 3
Enter Book ID: 102
Enter Username: Ram

Enter your choice: 5
ReferenceBook: 101: Data Structures and Algorithms: Cormen
TextBook: 102: Programming Ruby: Thomas: Borrowed by Ram

Enter your choice: 4
Enter Book ID: 102

Enter your choice: 5
ReferenceBook: 101: Data Structures and Algorithms: Cormen
TextBook: 102: Programming Ruby: Thomas: Available

Enter your choice: 6

7). Assume that Ram is the owner of a Factory who owns so many cars and trucks. He needs to keep track of his vehicles based on the total tax that should be paid every month. For this purpose, implement a management system with the help of OOP. In the design of the management system, it is decided that there should be a class **Vehicle** that stores the ***modelNumber***, ***rateBought,*** ***fuelType*** *(*possible values are "petrol" and "diesel"*)* and ***numberOfWheels***. Create the derived classes, **Car** that has one more attribute ***numberOfPassengers*** and **Truck** that has the extra attribute ***loadLimit***. For Car, the *numberOfWheels* is 4 and for Truck *numberOfWheels* is 6 by default. There should be one interface **TaxCalculatable,** which has a method ***calculateTax***. **Vehicle** class should implement this interface, and the tax calculations should be as follows.

For Car with *fuelType* petrol , total tax amount is (*rateBought**0.1 * *numberOfPassengers*)*0.5
For Car with *fuelType* diesel, total tax amount is (*rateBought**0.1 * *numberOfPassengers*)*0.4
For Truck with *fuelType* petrol , total tax amount is (*rateBought**0.1 * *loadLimit*0.002*)*0.5
For Truck with *fuelType* diesel, total tax amount is (*rateBought**0.1 * *loadLimit*0.002*)*0.4

Write a Java program to create a vehicle management system where a list of vehicles of these two derived classes should be stored and displayed. While displaying the information about the stored vehicles, display vehicles in the ***decreasing*** order of total tax.

Sample input and output:

1. Add Vehicle
2. Display Vehicles
3. Exit

Enter your choice:1
a.Car
b.Truck
Enter your choice:a
ModelNumber:Maruti800
Rate:200000
FuelType(petrol/diesel):petrol
Passengers:4

1. Add Vehicle
2. Display Vehicles
3. Exit
Enter your choice:1
a.Car
b.Truck
Enter your choice:b
ModelNumber:A100
Rate:2500000
FuelType(petrol/diesel):petrol
Loadlimit:400

1. Add Vehicle
2. Display Vehicles
3. Exit
Enter your choice:2
A100 petrol 100000
Maruti800 petrol 40000

1. Add Vehicle
2. Display Vehicles
3. Exit
Enter your choice:3