جامعة أم القرى

كلية الحاسب الآلي بالليث

Programming Contest

PROBLEMS

2018-2019

Ali AL NAJJAR 11/4/18 Problems

Roots of a Quadratic Equation Problem Code: P1

Write a program to take the values for *A*, *B*, *C* of a quadratic equation *A*X2+B*X+C=0* and then find all the roots of the equation. It is guaranteed that $A \neq 0$ and that the equation has at least one real root.

$$x1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$x2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Input:
- First line will contain the value of *A*
- Second line will contain the value of *B*
- Third line will contain the value of *C*

Output:
Output two lines. First line contains the value of root 1 (x1) and second line contains the value of root 2 (x2). Your output will be considered to be correct if the difference between your output and the actual answer is not more than 10-6

Constraints
- $-1000 \leq A \leq 1000$
- $-1000 \leq B \leq 1000$
- $-1000 \leq C \leq 1000$

Sample Input:
- 1
- -8
- 15

Sample Output:
- 5
- 3

Decrement OR Increment Problem Code: P2

Write a program to obtain a number N and increment its value by 1 if the number is divisible by 4 otherwise decrement its value by 1.

Input:
- First line will contain a number N.

Output:
Output a single line, the new value of the number.
Constraints
- 0≤N≤1000

Sample Input:
- 5

Sample Output ***:***
- 4

EXPLANATION:
- Since 5 is not divisible by 4 hence, its value is decreased by 1.

Sum OR Difference Problem Code: P3

Write a program to take two numbers as input and print their difference if the first number is greater than the second number otherwise print their sum.

Input:

- First line will contain the first number (N1)
- Second line will contain the second number (N2)

Output:

Output a single line containing the difference of 2 numbers (N1−N2) if the first number is greater than the second number otherwise output their sum (N1+N2).

Constraints

- −1000≤N1≤1000
- −1000≤N2≤1000

Sample Input:

```
82

28
```

Sample Output:

```
54
```

Minimum Maximum Problem Code: P4

Student loves to play with arrays by himself. Today, he has an array A consisting of N distinct integers. He wants to perform the following operation on his array A.

Select a pair of adjacent integers and remove the larger one of these two. This decreases the array size by 1. Cost of this operation will be equal to the smaller of them.
Find out minimum sum of costs of operations needed to convert the array into a single element.

Input

First line of input contains a single integer T denoting the number of test cases. First line of each test case starts with an integer N denoting the size of the array A. Next line of input contains N space separated integers, where the i-th integer denotes the value Ai.

Output

For each test case, print the minimum cost required for the transformation.

Constraints

1 ≤ T ≤ 10

2 ≤ N ≤ 50000

1 ≤ Ai ≤ 105

Explanation

Test 1: Student will make only 1 move: pick up both the elements (that is, 3 and 4), remove the larger one (4), incurring a cost equal to the smaller one (3).

Small factorials Problem Code: P5

You are asked to calculate factorials of some small positive integers.

Input

An integer t, 1<=t<=100, denoting the number of testcases, followed by t lines, each containing a single integer n, 1<=n<=100.

Output

For each integer n given at input, display a line with the value of n!

Example

Sample input:

```
4

1

2

5

3
```

Sample output:

```
1

2

120

6
```

Decimal to Binary Problem Code: p6

Write a Java program to convert a decimal number to binary number.

Decimal number: The decimal numeral system is the standard system for denoting integer and non-integer numbers. It is also called base-ten positional numeral system.

Binary number: In digital electronics and mathematics, a binary number is a number expressed in the base-2 numeral system or binary numeral system. This system uses only two symbols: typically, 1 (one) and 0 (zero).

Input

Sample Example

a Decimal Number: 5

Output

a Binary Number: 0101

Largest and Smallest Problem Code: P7

Write a program with a loop that lets the user enter a series of integers. The user should enter −99 to signal the end of the series. After all the numbers have been entered, the program should display the largest and smallest numbers entered.

Input

```
Enter an integer, or -99 to quit: 45

Enter an integer, or -99 to quit: 20

Enter an integer, or -99 to quit: 32

Enter an integer, or -99 to quit: 54

Enter an integer, or -99 to quit: 12

Enter an integer, or -99 to quit: -99
```

Expected Output:

```
Largest: 54

Smallest: 12
```

Celsius to Fahrenheit Table Problem Code: P8

Write a program that displays a table of the Celsius temperatures 0 through 20 and their Fahrenheit equivalents. The formula for converting a temperature from Celsius to Fahrenheit is

$$F=\frac{9}{5}C+32$$

where F is the Fahrenheit temperature and C is the Celsius temperature. Your program must use a loop to display the table.

Output

```
Celsius      Fahrenheit
----------------------------------------

 0.0        32.0

 1.0        33.8

 2.0        35.6

 3.0        37.4

 4.0        39.2

 5.0        41.0

 6.0        42.8

 7.0        44.6

 8.0        46.4

 9.0        48.2

 10.0       50.0

 11.0       51.8

 12.0       53.6

 13.0       55.4

 14.0       57.2

 15.0       59.0

 16.0       60.8

 17.0       62.6

 18.0       64.4
```

Random Number Guessing Game Problem Code: P9

Write a program that generates a random number and asks the user to guess what the number is. If the user’s guess is higher than the random number, the program should display “Too high, try again.” If the user’s guess is lower than the random number, the program should display “Too low, try again.” The program should use a loop that repeats until the user correctly guesses the random number.

Input and output expected

```
I'm thinking of a number.

Guess what it is: 12

Sorry, that's too high.

Guess again: 11

Sorry, that's too high.

Guess again: 10

Sorry, that's too high.

Guess again: 7

Sorry, that's too high.

Guess again: 6

Sorry, that's too high.

Guess again: 18

Sorry, that's too high.
```

invoice for an item sold at the store Problem Code: P10

Create a class called Invoice that a hardware store might use to represent an invoice for an item sold at the store.
An Invoice should include four pieces of information as instance variables-a part number (type String), a part description (type String), a quantity of the item being purchased (type int) and a price per item (double). Your class should have a constructor that initializes the four instance variables. Provide a set and a get method for each instance variable. In addition, provide a method named getInvoice Amount that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0. Write a test application named InvoiceTest that demonstrates class Invoice's capabilities.

input

```
Invoice1 ("A5544", "Big Black Book", 500, 250.00)

Invoice2 ("A5542", "Big Pink Book", 300, 50.00);
```

Output

```
Invoice 1: A5544 Big Black Book   500  $250.00

Invoice 2: A5542 Big Pink Book    300  $50.00
```

Dice Game. Problem Code: P11

Write a program that plays a simple dice game between the computer and the user. When the program runs, a loop should repeat 10 times. Each iteration of the loop should do the following:

• Generate a random integer in the range of 1 through 6. This is the value of the computer's die.
• Generate another random integer in the range of 1 through 6. This is the value of the user's die.
• The die with the highest value wins. (In case of a tie, there is no winner for that particular roll of the dice.)
As the loop iterates, the program should keep count of the number of times the computer wins, and the number of times that the user wins. After the loop performs all of its iterations, the program should display who was the grand winner, the computer or the user.

Output

```
Computer....4

User........5

Ties........1

The user is the grand winner!

Computer....5

User........2

Ties........3

The computer is the grand winner!

Computer....4

User........5

Ties........1

The user is the grand winner!

Computer....5
```

```
User........3

Ties........2

The computer is the grand winner!

Computer....6

User........4

Ties........0

The computer is the grand winner!

Computer....5

User........2

Ties........3

The computer is the grand winner!

Computer....4

User........4

Ties........2

The game has ended in a tie!
```

showChar Method. Problem Code: P12

Write a method named `showChar`. The method should accept two arguments: a reference to a `String` object and an integer. The integer argument is a character position within the `String`, with the first character being at position 0. When the method executes, it should display the character at that character position. Here is an example of a call to the method:

```
showChar("New York", 2);
```

In this call, the method will display the character `w` because it is in position 2. Demonstrate the method in a complete program.

Output

```
w
```

Conversion Program. Problem Code: P13

Write a program that asks the user to enter a distance in meters. The program will then present the following menu of selections:
1. Convert to kilometers
2. Convert to inches
3. Convert to feet
4. Quit the program

The program will convert the distance to kilometers, inches, or feet, depending on the user's selection. Here are the specific requirements:

• Write a `void` method named `showKilometers`, which accepts the number of meters as an argument. The method should display the argument converted to kilometers. Convert the meters to kilometers using the following formula:
```
kilometers = meters * 0.001
```
• Write a `void` method named `showInches`, which accepts the number of meters as an argument. The method should display the argument converted to inches. Convert the meters to inches using the following formula:
```
inches = meters * 39.37
```
• Write a `void` method named `showFeet`, which accepts the number of meters as an argument. The method should display the argument converted to feet. Convert the meters to feet using the following formula:
```
feet = meters * 3.281
```
• Write a `void` method named `menu` that displays the menu of selections. This method should not accept any arguments.

• The program should continue to display the menu until the user enters 4 to quit the program.

• The program should not accept negative numbers for the distance in meters.

• If the user selects an invalid choice from the menu, the program should display an error message.

Here is an example session with the program, using console input. The user's **input is shown in bold**.
```
Enter a distance in meters: 500 [Enter]
1. Convert to kilometers
2. Convert to inches
3. Convert to feet
4. Quit the program

Enter your choice: 1 [Enter]
```

```
500 meters is 0.5 kilometers.
1. Convert to kilometers
2. Convert to inches
3. Convert to feet
4. Quit the program
```

```
Enter your choice: **3 [Enter]**
500 meters is 1640.5 feet.
1. Convert to kilometers
2. Convert to inches
3. Convert to feet
4. Quit the program
Enter your choice: **4 [Enter]**
Bye!
```

Output:

```
Enter a distance in meters: 20

1. Convert to kilometers

2. Convert to inches

3. Convert to feet

4. Quit the program


Enter your choice: 1

20.0 meters is 0.02 kilometers.


1. Convert to kilometers

2. Convert to inches

3. Convert to feet

4. Quit the program


Enter your choice: 2

20.0 meters is 787.4 inches.
```

```
1. Convert to kilometers
2. Convert to inches
3. Convert to feet
4. Quit the program

Enter your choice: 3
20.0 meters is 65.62 feet.

1. Convert to kilometers
2. Convert to inches
3. Convert to feet
4. Quit the program

Enter your choice: 4
Bye!
```

Even/Odd Counter. Problem Code: P14

You can use the following logic to determine whether a number is even or odd:

```
if ((number % 2) == 0)
{
// The number is even.
}
else
{
// The number is odd.
}
```

Write a program with a method named `isEven` that accepts an `int` argument. The method should return `true` if the argument is even, or `false` otherwise. The program's `main` method should use a loop to generate 100 random integers. It should use the `isEven` method to determine whether each random number is even, or odd. When the loop is finished, the program should display the number of even numbers that were generated, and the number of odd numbers.

Output

```
Out of 100 randomly generated numbers :46 were even and 55 is
odd
```

ESP Game. Problem Code: P15

Write a program that tests your ESP (extrasensory perception). The program should randomly select the name of a color from the following list of words: *Red, Green, Blue, Orange, Yellow*

To select a word, the program can generate a random number. For example, if the number is 0, the selected word is *Red*; if the number is 1, the selected word is *Green*; and so forth.

Next, the program should ask the user to enter the color that the computer has selected.

After the user has entered his or her guess, the program should display the name of the randomly selected color. The program should repeat these 10 times and then display the number of times the user correctly guessed the selected color. Be sure to modularize the program into methods that perform each major task.

Input

```
I'm thinking of acolor.
is red,green,blue,orange,or yellow?
Red
```

Output

```
randomly yellow
```

<u>**Temperature Class.** Problem Code: P16</u>

Write a `Temperature` class that will hold a temperature in Fahrenheit, and provide methods to get the temperature in Fahrenheit, Celsius, and Kelvin. The class should have the following field:

• `ftemp` – A `double` that holds a Fahrenheit temperature.

The class should have the following methods:

• Constructor – The constructor accepts a Fahrenheit temperature (as a `double`) and stores it in the `ftemp` field.

• `setFahrenheit` – The `setFahrenheit` method accepts a Fahrenheit temperature (as a `double`) and stores it in the `ftemp` field.

• `getFahrenheit` – Returns the value of the `ftemp` field, as a Fahrenheit temperature (no conversion required).

• `getCelsius` – Returns the value of the `ftemp` field converted to Celsius.

• `getKelvin` – Returns the value of the `ftemp` field converted to Kelvin

Use the following formula to convert the Fahrenheit temperature to Celsius:

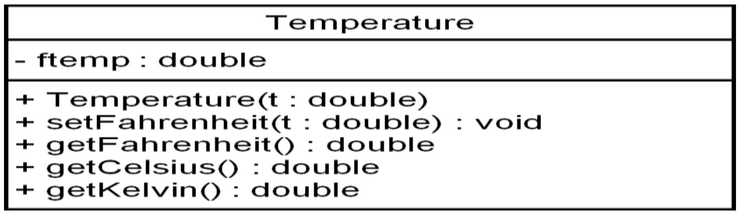
$Celsius = (5/9) \times (Fahrenheit - 32)$

Use the following formula to convert the Fahrenheit temperature to Kelvin:

$Kelvin = ((5/9) \times (Fahrenheit - 32)) + 273$

Demonstrate the `Temperature` class by writing a separate program that asks the user for a Fahrenheit temperature. The program should create an instance of the `Temperature` class, with the value entered by the user passed to the constructor. The program should then call the object's methods to display the temperature in Celsius and Kelvin.

<u>**Temperature Class UML**</u>



```
Temperature
- ftemp : double
+ Temperature(t : double)
+ setFahrenheit(t : double) : void
+ getFahrenheit() : double
+ getCelsius() : double
+ getKelvin() : double
```

<u>***Input***</u>

Enter the Fahrenheit temperature: 74

<u>***output***</u>

Celsius: 23.33

Kelvin: 296.33

Charge Account Validation. Problem Code: P17

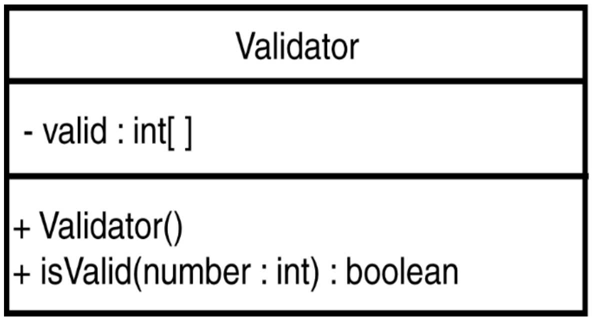
Create a class with a method that accepts a charge account number as its argument. The method should determine whether the number is valid by comparing it to the following list of valid charge account numbers:

```
5658845 4520125 7895122 8777541 8451277 1302850
8080152 4562555 5552012 5050552 7825877 1250255
1005231 6545231 3852085 7576651 7881200 4581002
```

These numbers should be stored in an array or an `ArrayList` object. Use a sequential search to locate the number passed as an argument. If the number is in the array, the method should return `true`, indicating the number is valid. If the number is not in the array, the method should return `false`, indicating the number is invalid.

Write a program that tests the class by asking the user to enter a charge account number. The program should display a message indicating whether the number is valid or invalid.

Validator Class UML



```
Input
Enter your charge account number:145678912
Enter your charge account number: 5658845
Output
That's an INVALID account number.
That's a valid account number.
```

Array Operations. Problem Code: P18

Write a program with an array that is initialized with test data. Use any primitive data type of your choice. The program should also have the following methods:

• `getTotal.` This method should accept a one-dimensional array as its argument and return the total of the values in the array.

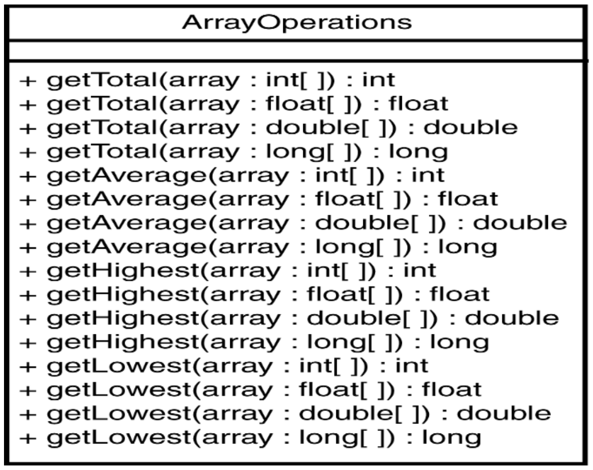
• `getAverage.` This method should accept a one-dimensional array as its argument and return the average of the values in the array.

• `getHighest.` This method should accept a one-dimensional array as its argument and return the highest value in the array.

• `getLowest.` This method should accept a one-dimensional array as its argument and return the lowest value in the array.

Demonstrate each of the methods in the program.

UML DIAGRAM for ARRAY OPERATIONS



```
ArrayOperations

+ getTotal(array : int[ ]) : int
+ getTotal(array : float[ ]) : float
+ getTotal(array : double[ ]) : double
+ getTotal(array : long[ ]) : long
+ getAverage(array : int[ ]) : int
+ getAverage(array : float[ ]) : float
+ getAverage(array : double[ ]) : double
+ getAverage(array : long[ ]) : long
+ getHighest(array : int[ ]) : int
+ getHighest(array : float[ ]) : float
+ getHighest(array : double[ ]) : double
+ getHighest(array : long[ ]) : long
+ getLowest(array : int[ ]) : int
+ getLowest(array : float[ ]) : float
+ getLowest(array : double[ ]) : double
+ getLowest(array : long[ ]) : long
```

Input

```
int[] iarray = { 2, 1, 9, 7, 3 };
float[] farray = { 3.5F, 4.6F, 1.7F, 8.9F, 2.1F };
double[] darray = { 98.7, 89.2, 55.1, 77.6, 99.9 };
long[] larray = {100, 500, 200, 300, 400 };
```

Output

```
Processing the long array.
Total: 1500
Average: 300
Highest value: 500
Lowest value: 100
```

Geometry Calculator. Problem Code: P19

Design a `Geometry` class with the following methods:

• A static method that accepts the radius of a circle and returns the area of the circle.
Use the following formula:
Area = πr^2
Use `Math.PI` for π and the radius of the circle for *r*.

• A static method that accepts the length and width of a rectangle and returns the area of the rectangle. Use the following formula:
Area = *Length* × *Width*

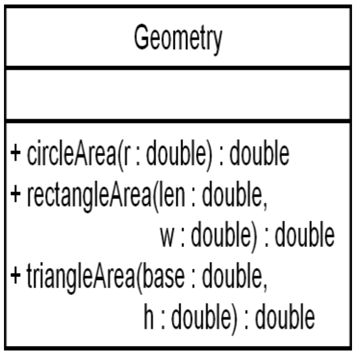
• A static method that accepts the length of a triangle's base and the triangle's height.
The method should return the area of the triangle. Use the following formula:
Area = *Base* × *Height* × 0.5

The methods should display an error message if negative values are used for the circle's radius, the rectangle's length or width, or the triangle's base or height.
Next, write a program to test the class, which displays the following menu and responds to the user's selection:

```
Geometry Calculator
1. Calculate the Area of a Circle
2. Calculate the Area of a Rectangle
3. Calculate the Area of a Triangle
4. Quit
Enter your choice (1-4):
```

Display an error message if the user enters a number outside the range of 1 through 4 when selecting an item from the menu.

UML DIAGRAM for Geometry


Geometry
+ circleArea(r : double) : double
+ rectangleArea(len : double, w : double) : double
+ triangleArea(base : double, h : double) : double

input

```
Geometry Calculator
1. Calculate the Area of a Circle
2. Calculate the Area of a Rectangle
3. Calculate the Area of a Triangle
4. Quit
Enter your choice (1-4) :
```

Output

```
Enter your choice (1-4) : 1
What is the circle's radius?
The circle's area is 12.566370614359172
```

BankAccount and SavingsAccount Classes. Problem Code: P20

Design an abstract class named `BankAccount` to hold the following data for a bank account:

- Balance
- Number of deposits this month
- Number of withdrawals
- Annual interest rate
- Monthly service charges

The class should have the following methods:

Constructor: The constructor should accept arguments for the balance and annual interest rate.

`deposit`: A method that accepts an argument for the amount of the deposit. The method should add the argument to the account balance. It should also increment the variable holding the number of deposits.

`withdraw`: A method that accepts an argument for the amount of the withdrawal. The method should subtract the argument from the balance. It should also increment the variable holding the number of withdrawals.

`calcInterest`: A method that updates the balance by calculating the monthly interest earned by the account and adding this interest to the balance. This is performed by the following formulas:

Monthly Interest Rate = (*Annual Interest Rate* / 12)

*Monthly Interes*t = *Balance* * *Monthly Interest Rate*

Balance = *Balance* + *Monthly Interest*

`monthlyProcess`: A method that subtracts the monthly service charges from the balance, calls the `calcInterest` method, and then sets the variables that hold the number of withdrawals, number of deposits, and monthly service charges to zero.

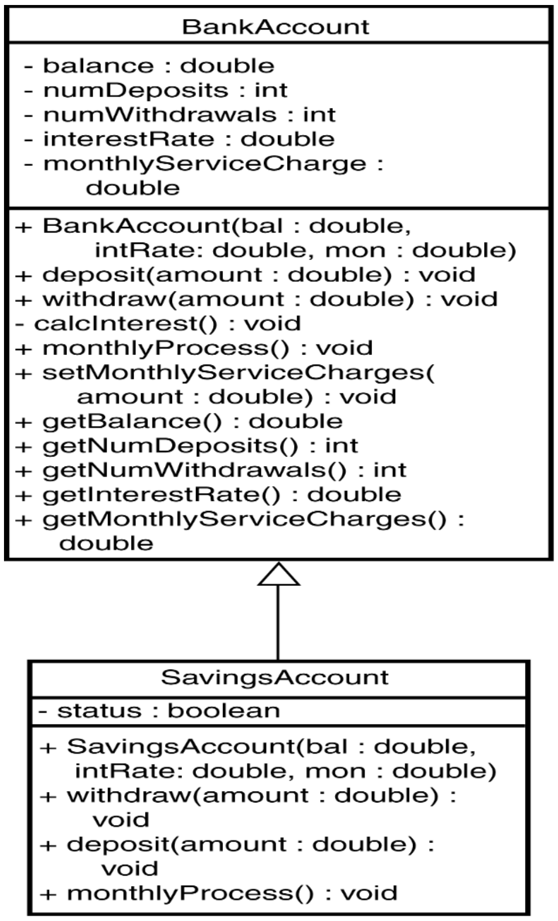
`withdraw`: A method that determines whether the account is inactive before a withdrawal is made. (No withdrawal will be allowed if the account is not active.) A withdrawal is then made by calling the superclass version of the method.

`deposit`: A method that determines whether the account is inactive before a deposit is made. If the account is inactive and the deposit brings the balance above $25, the account becomes active again. A deposit is then made by calling the superclass version of the method.

`monthlyProcess`: Before the superclass method is called, this method checks the number of withdrawals. If the number of withdrawals for the month is more than 4, a service charge of $1 for each withdrawal above 4

is added to the superclass field that holds the monthly service charges.
(Don't forget to check the account balance after the service charge is taken. If the balance falls below $25, the account becomes inactive.)
Next, design a `SavingsAccount` class that extends the `BankAccount` class. The `SavingsAccount` class should have a status field to represent an active or inactive account. If the balance of a savings account falls below $25, it becomes inactive. (The `status` field could be a `Boolean` variable.) No more withdrawals may be made until the balance is raised above $25, at which time the account becomes active again. The savings account class should have the following methods:
`withdraw`: A method that determines whether the account is inactive before a withdrawal is made. (No withdrawal will be allowed if the account
is not active.) A withdrawal is then made by calling the superclass
version of the method.
`deposit`: A method that determines whether the account is inactive before a deposit is made. If the account is inactive and the deposit brings the balance above $25, the account becomes active again. A deposit is then made by calling the superclass version of the method.
`monthlyProcess`: Before the superclass method is called, this method checks the number of withdrawals. If the number of withdrawals for the month is more than 4, a service charge of $1 for each withdrawal above 4
is added to the superclass field that holds the monthly service charges.
(Don't forget to check the account balance after the service charge is taken. If the balance falls below $25, the account becomes inactive.)

UML Diagram for problem P20


BankAccount
- balance : double
- numDeposits : int
- numWithdrawals : int
- interestRate : double
- monthlyServiceCharge : double
+ BankAccount(bal : double, intRate: double, mon : double)
+ deposit(amount : double) : void
+ withdraw(amount : double) : void
- calcInterest() : void
+ monthlyProcess() : void
+ setMonthlyServiceCharges(amount : double) : void
+ getBalance() : double
+ getNumDeposits() : int
+ getNumWithdrawals() : int
+ getInterestRate() : double
+ getMonthlyServiceCharges() : double
SavingsAccount
- status : boolean
+ SavingsAccount(bal : double, intRate: double, mon : double)
+ withdraw(amount : double) : void
+ deposit(amount : double) : void
+ monthlyProcess() : void

Input and output

```
Balance: $100.00
Number of deposits: 0
Number of withdrawals: 0
Balance: $170.00
Number of deposits: 3
Number of withdrawals: 0
Balance: $20.00
Number of deposits: 3
Number of withdrawals: 2
Balance: $17.54
Number of deposits: 0
Number of withdrawals: 0
```