Timestamp: 2020/10/30 15:41

CSC1016S Make-up Assignment 2

Introduction to objects and classes

Assignment Instructions

This assignment involves constructing programs in Java using class declarations to model simple types of object.

Exercise One [25 marks]

Write a `Student` class to model a student's name, composed of a first name, middle name and last name. The class should meet the following specification:

Class Student

A Student object represents a student. A student has a first name, middle name and last name.

Instance variables

private String firstName;
private String middleName;
private String lastName;

Methods

public void setNames(String first, String middle, String last)
    // Set the first, middle and last names of this Student object.

public String getFullName()
    // Obtain the full name of this Student with the middle name converted to an initial only.

Write a driver class called '`TestStudent`' containing a main method. The main method should

1. Ask the user to input a student's first, middle and last names,
2. Create an instance of the Student class.
3. Call setNames using the data input by the user.
4. Call the method to compute the full name and output it.

Sample I/O

```
Enter first name:
Tito
Enter middle name:
GimmeYourMoney
Enter last name:
Mboweni
The full name is: Tito G. Mboweni
```

Continued

Exercise Two [25 marks]

This question concerns developing a pseudo random number generator (PRNG). Java provides a perfectly serviceable PRNG (`java.util.Random`), however, building one is a useful OO exercise and offers an insight into these things are done.

A pseudo random number generator is a device that can produce or output a sequence of numbers that appear to be randomly selected.

One approach to developing a PRNG is based on the following general formula:

$$X_{k+1} = (gX_k + c) \; mod \; n$$

(Note that the word "mod" denotes the modulo operation. For example, 5 mod 3 is 2.)

The formula describes a sequence of pseudo random values. The term X_k denotes one value in the sequence while X_{k+1} denotes the next. Given a value in the sequence, we calculate the next by applying the formula. Customarily, a seed or start value, X_0 is provided with which to kick the whole thing off.

According to this Wikipedia page, good values for the constants are:

$c = 0$
$g = 48{,}271$
$n = 2^{31} - 1 = 2{,}147{,}483{,}647$

Here's a specification for a RandomNumber generator class:

Class RandomGenerator

A RandomGenerator is a Pseudo Random Number Generator (PRNG). Its implementation is the Lehmer variant of linear congruential generator.

Instance variables

private long x;

// This field holds the most recently calculated value i.e. X_k.

Constructors

public RandomGenerator(long seed)

// Create a RandomGenerator object that uses the given seed as the value for X_0.

Methods

public int nextInt()

// Return a pseudorandom integer value.

public int nextInt(int i)

// Return a pseudorandom integer value between 0 (inclusive) and the specified value (exclusive).

public double nextDouble()

// Return a pseudorandom real number value between 0.0 (inclusive) and 1.0 (exclusive).

Continued

Construct a `RandomGenerator` class of object that meets the given specification and is implemented using the given formula and constants.

- Note the use of the data type `long` for the instance variable `x`. The numbers that the formula generates are very large integers hence this is the best choice.
- Each 'next…' method should obtain a value for X_{k+1} and then use it to obtain the required result.
    - The `nextInt()` method should obtain the result of ***casting*** X_{k+1} as an integer.
    - The `nextDouble()` method should obtain the result of dividing $X_{k+1}$by n.
      (You may wish to investigate the difference between, say, $3/4$ and $3/4.0$.)
    - The `nextInt(int i)` method should obtain `nextDouble()`, scale it and cast it.

On the Vula page for this assignment you will find a test harness class called `TestPRNG` that may be used to check your work. Here is an example of expected output:

```
Enter a seed value
7
Enter a parameter value for nextInt(int i):
10
Enter the number of calls required of each method.
5
nextInt(): 337897
nextDouble(): 0.5952271440044172
nextInt(10): 2
nextInt(): 518142577
nextDouble(): 0.7756899137868033
nextInt(10): 3
nextInt(): 1298864186
nextDouble(): 0.786058717307662
nextInt(10): 8
nextInt(): 439347582
nextDouble(): 0.6268343877172258
nextInt(10): 9
nextInt(): 162366641
nextDouble(): 0.6669665727144883
nextInt(10): 1
```

Note: you can comment out lines in the test harness, enabling incremental development of `RandomGenerator` methods.

Also Note: The automaker will test methods individually, meaning that you can obtain partial marks for an incomplete solution.

Continued

Exercise Three [25 marks]

Consider the following specification for a `Collator` type of object:

Class Collator

A Collator object manages information on a sequence of readings. It records a label (for the readings), the number of readings, the maximum, the minimum and the average.

Constructors

Collator(String label)

Create a Collator object. The number of readings is set to zero. The given string is used as the readings label.

Methods

void recordReading(double reading)

// Use the given reading to update the record.

String label()

// Obtain the label for these readings.

double maximum()

// Obtain the largest reading taken. Requires numberOfReadings()>0.

double minimum()

// Obtain the lowest reading taken. Requires numberOfReadings()>0.

double average()

// Obtain the average of readings taken, rounded to the nearest integer. Requires numberOfReadings()>0.

int numberOfReadings()

// Obtain the number of readings which have been taken. Requires numberOfReadings()>0.

NOTE:

- Individual readings do not need to be stored.
  Only average, number of readings, maximum and minimum are required.
- Average must be stored as a real number.

Construct a class declaration for `Collator`, and then build a program called `Meteorology` that uses 3 `Collator` objects, one for temperature, one for pressure, one for humidity, and that behaves as follows:

```
Meteorology Program
Make a selection and press return:
1. Record a temperature reading.
2. Record a pressure reading.
3. Record a humidity reading.
4. Print maximum values.
5. Print minimum values.
6. Print average values.
7. Quit.
1
Enter value:
17
Make a selection and press return:
```

Continued

```
1. Record a temperature reading.
2. Record a pressure reading.
3. Record a humidity reading.
4. Print maximum values.
5. Print minimum values.
6. Print average values.
7. Quit.
4
Maximum temperature: 17.00
Maximum pressure: -
Maximum humidity: -
Make a selection and press return:
1. Record a temperature reading.
2. Record a pressure reading.
3. Record a humidity reading.
4. Print maximum values.
5. Print minimum values.
6. Print average values.
7. Quit.
2
Enter value:
1020
Make a selection and press return:
1. Record a temperature reading.
2. Record a pressure reading.
3. Record a humidity reading.
4. Print maximum values.
5. Print minimum values.
6. Print average values.
7. Quit.
1
Enter value:
11
Make a selection and press return:
1. Record a temperature reading.
2. Record a pressure reading.
3. Record a humidity reading.
4. Print maximum values.
5. Print minimum values.
6. Print average values.
7. Quit.
5
Minimum temperature: 11.00
Minimum pressure: 1020.00
Minimum humidity: -
Make a selection and press return:
1. Record a temperature reading.
2. Record a pressure reading.
3. Record a humidity reading.
4. Print maximum values.
5. Print minimum values.
6. Print average values.
7. Quit.
3
Enter value:
87
Make a selection and press return:
```

Continued

```
1. Record a temperature reading.
2. Record a pressure reading.
3. Record a humidity reading.
4. Print maximum values.
5. Print minimum values.
6. Print average values.
7. Quit.
6
Average temperature: 14.00
Average pressure: 1020.00
Average humidity: 87.00
Make a selection and press return:
1. Record a temperature reading.
2. Record a pressure reading.
3. Record a humidity reading.
4. Print maximum values.
5. Print minimum values.
6. Print average values.
7. Quit.
7
```

Exercise Four [25 marks]

The Acme cash register is a simple device that is used in commerce to calculate sales and change due. (Actually, strictly speaking, it's not a register at all since it does not record sales. It just assists with the calculations.)

The device supports customer transactions in the following way:

- Ready to ring up a customer's purchases, the running total is initially zero.
- As the operative enters the price of each item the running total is updated.
- Once the final item has been added, requesting payment, the operator may enter the amount tendered. This causes the cash drawer to open along with the calculation of the change that is due.
- Having received payment and obtained change as required, closing the cash drawer serves to complete the transaction, setting the running total, amount tendered and change due back to zero.
- The running total may be examined at any point.
- A transaction can be cancelled at any point. All values stored by the register revert to zero.

An Acme cash register simulator is required. Here is sample I/O for the program:

```
Cash Register
Running total: R0.00
1. Ring up item.
2. Enter amount tendered.
3. New transaction.
4. Quit.
1
Enter amount:
34.50
Running total: R34.50
1. Ring up item.
2. Enter amount tendered.
3. New transaction.
```

Continued

```
4. Quit.
2
```

Continued

```
Enter amount:
50
Running total: R34.50
Amount tendered: R50.00
Change due: R15.50
1. Ring up item.
2. Enter amount tendered.
3. New transaction.
4. Quit.
3
Running total: R0.00
1. Ring up item.
2. Enter amount tendered.
3. New transaction.
4. Quit.
4
```

Design and implement a `CashRegister` class that models Acme cash register function, and then write a program called `RegisterUI` that uses a `CashRegister` object, providing I/O.

Use the design of the meterology program of question 3 to guide you:

- The `CashRegister` class should contain methods for each register function.
- The `RegisterUI` class should be responsible for creating a `CashRegister` object, obtaining user input, calling the `CashRegister` object's methods, and printing results.
- The `CashRegister` class should NOT contain any print statements.

Note that five of the marks available for this question will be for good design practices – meaningful names, adequate quantity of methods, adequate choice of data types etc. To obtain these marks you must present your work to a tutor.

Marking and Submission

Submit the *Student.java*, *TestStudent.java*, *RandomGenerator.java*, *Collator.java*, *Meteorology.java*, *CashRegister.java* and *RegisterUI.java* files contained within a single .ZIP folder to the automatic marker. The zipped folder should have the following naming convention:

yourstudentnumber.zip

END