CSC1016S 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.

An important aspect of object-orientation is abstraction. When thinking about how you use a type of object, of what it is supposed to do, you don't need to know about how it is implemented i.e. what instance variables it contains, and how methods are coded.

To emphasis this way of thinking, a number of the exercises involve the use of software testing as a means of guiding problem analysis, solution design and evaluation. The tests are based on an 'outside' view of objects.

The first two exercises involve using supplied test suites to guide development of classes of objects. The third involves devising your own tests as an aid to comprehending a given type of object. The final exercise involves using that same type of object to construct a program.

Exercise One [25 marks]

This question concerns (i) writing a `Student` class to model a student's name, composed of a first name, middle name and last name, and (ii) using a supplied test suite to drive the process.

The `Student` 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.

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.

Thinking of the problem in terms of testing, there are three requirements that must be met:

A. The `setNames()` method sets the names stored in the object i.e. the values of `firstName`, `middleName`, `lastName`.
B. The getFullName() method obtains the name of the student with the middle name converted to an initial only.
C. The getFullName() method does not change anything – i.e. it does not modify the values of `firstName`, `middleName`, `lastName`.

Note that, these requirements are interdependent i.e. you can only check `setNames()` works by using `getFullName()`, and you can only check `getFullName()` works by first using `setNames()`.

On the Vula page for this assignment, you will find a program called `TestStudent` that you should add to your JGrasp project for this question.

CONTINUED

The program assumes that you ‘ve constructed a `Student` class, and as its name suggests, it runs tests on Student objects.

It codes test requirements as a set of two tests: the first test evaluates A and B in conjunction, and the second test focuses on C.

We could have written one test covering all three requirements, however, two tests offer a more precise diagnosis of problems with code.

Use the `TestStudent` program to develop your `Student` class:

- If a test fails, look at the TestStudent program to see what the test does.
- If the tests pass, then your Student class will receive full marks from the automatic marker.

HINT: To start your Student class, we suggest using the following instance variables:

Instance variables

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

CONTINUED

Exercise Two [25 marks]

This question concerns Uber, in particular, (i) writing an `UberService` class to model an Uber service from the point of view of price, and (ii) using a supplied test suite to drive the process.

Consider the following specification for an `UberService` type of object.

Class UberService

An UberService object represents an Uber service from the point of view of price. (Services differ in characteristics such as luxury and capacity of the vehicle, and consequently, in price.)

Each service has a base fare, a cost per minute, cost per kilometre, and cancellation fee. The cost of a journey consists of the base fare + cost for minutes + cost for distance.

Methods

```
void setDetails(String name, int costPerMin, int costPerKM, int baseFee, int cancellationFee)
        // Set the details of this service to the given values.

void setName(String name)
        // Set the service name.

String getName()
        // Obtain the service name.

void setCostPerMinute(int cents)
        // Set the cost per minute.

int getCostPerMinute()
        // Set the cost per minute in cents.

void setCostPerKilometre(int cents)
        // Set the cost per kilometre.

int getCostPerKilometre()
        // get the cost per kilometre in cents.

void setBaseFare(int cents)
        // Set the base fare.

int getBaseFare()
        // get the base fare in cents.

void setCancellationFee(int n)
        // Set the cancellation fee.

int getCancellationFee()
        // Obtain the cancellation fee in cents for this service.

double calculateFare(double minutes, double distance)
        // Obtain the fare (in the form of a real number of cents) for a journey of the
        // given time and distance.
```

CONTINUED

Thinking in terms of testing, there are a number of requirements that must be met:

A. `setDetails()` sets the service name, cost per minute, cost per kilometre, base fee and cancellation fee.
B. `setName()` sets the service name.
C. `setCostPerMinute()` sets the cost per minute.
D. `setCostPerKilometre()` sets the cost per kilometre.
E. `setBaseFare()` sets the base fare.
F. `setCancellationFee()` sets the cancellation fee.
G. `getName()` returns the service name.
H. `getCostPerMinute()` returns the cost per minute.
I. `getCostPerKilometre()` returns the cost per kilometre.
J. `getBaseFare()` returns the base fare.
K. `getCancellationFee()` returns the cancellation fee.
L. `getName()` ONLY returns the service name.
M. `getCostPerMinute()` ONLY returns the cost per minute.
N. `getCostPerKilometre()` ONLY returns the cost per kilometre.
O. `getBaseFare()` ONLY returns the base fare.
P. `getCancellationFee()` ONLY returns the cancellation fee.
Q. `calculateFare()` obtains the correct fare for the given trip duration and distance.
R. `setName()` ONLY sets the service name.
S. `setCostPerMinute()` ONLY sets the cost per minute.
T. `setCostPerKilometre()` ONLY sets the cost per kilometre.
U. `setBaseFare()` ONLY sets the base fare.
V. `setCancellationFee()` ONLY sets the cancellation fee.
W. `calculateFare()` ONLY obtains the correct fare for the given trip duration and distance.

On the Vula page for this assignment, you will find a program called `TestUberService` that codes these requirements as a set of 17 tests.

Of particular note are tests 16 and 17, which evaluate requirements Q and W. The `calculateFare()` method returns a real number. To test real numbers for 'equality', we usually seek to determine whether they are within some margin of error of each other. Hence expressions such as:

```
if (Math.abs(service.calculateFare(3.0, 7.0)-28710.0)<EPSILON) {
  …
}
```

Use the TestUberService class to guide development of the UberService class.

CONTINUED

Exercise Three [25 marks]

This question concerns devising test requirements and writing tests that confirm your understanding of the behaviour of a `Collator` type of object as describe in the following specification:

Class Collator

A Collator object has a label and manages information on a sequence of readings. It records 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.

Methods

void label(String label)
    // Change this Collator's label.

void recordReading(int reading)
    // Use the given reading to update the record.

String label()
    // Obtain this Collator's label.

int maximum()
    // Obtain the largest reading taken. (Assumes numberOfReadings()>0.)

int minimum()
    // Obtain the lowest reading taken. (Assumes numberOfReadings()>0.)

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

int numberOfReadings()
    // Obtain the number of readings which have been taken.

On the Vula page for this assignment you will find an implementation of this class. Based on the specification, devise a set of test requirements for this class, and construct a program called `TestCollator` that implements tests that satisfy these requirements. You should draw on the examples provided in questions one and two.

Your program should produce the same form of output as used by `TestStudent` and `TestUberService`.

e.g.

```
…

Test 10

Pass

Test 11

Fail

…
```

CONTINUED

Submit `TestCollator` program to the automatic marker.

- The marker will run it against versions of the Collator class that are faulty in various ways.
- It also checks that a correct `Collator` class passes all the tests. If not, then the marker deducts the full set of marks available for the question.

Exercise Four [25 marks]

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:
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
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
```

CONTINUED

```
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
Minimum pressure: 1020
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:
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
```

Marking and Submission

Submit the *Student.java*, *UberService.java*, *TestCollator.java*, and *Meteorology.java* files contained within a single .ZIP folder to the automatic marker. The zipped folder should have the following naming convention:

yourstudentnumber.zip

END

CONTINUED