COSC 1046 – Fall 2016 – Assignment 4

This assignment is designed to give you some practice and experience writing Java applications using methods. Each question specifies the name that each file/application should have. Please follow this naming convention. When you have completed the assignment compress all of your files into a single archive using Windows (Right Click folder -> Send To -> Compressed Folder) or OS X (Right Click folder -> Compress). Using a 3rd party compression utility such as WinRAR or 7zip may render your files unreadable and un-markable. Submit a single compressed file to D2L. You can resubmit your files as many times as you would like up to the due date and time.

Be sure to include your name and a brief description of your program (as comments) at the top of each file. Pay attention to using good variable names. If you have any questions, please check or post to the forum.

Submit solutions for these questions. Wherever applicable, do your best to reproduce my output exactly. In my sample **bold** indicates user-input.

If you work in partners, submit only one solution and make sure that both partners' names are in ALL files. If only one name appears, only one person will get the grade. No exceptions.

A word in general on assignment grading (for future assignments): If your program produces the displayed output and meets the criteria specified in the question you should expect to receive full marks. Deductions are taken when there are deviations in the output – small deductions for small deviations (calculation error, improper formatting, etc.) while larger deductions are taken for larger deviations (missing output, substantially incorrect values, etc.). Express your creativity in your code, not in your output. Also, deductions will be taken for:

- lack of comments
- poor variable names
- programs that don't run at all (large deduction), so make sure your program runs, even if it is not complete.

If you have any questions or concerns about the grading scheme before you submit an assignment, please post to the forum and ask for clarification. If you have concerns about the grading you've received on an assignment, please email me and I will review the grading form.

Do all of the questions to ensure that you are practicing all concepts but submit solutions to questions 1 and 2 ONLY for grading. It is assumed that you are doing all of the questions and some of them may be referenced in future assignments. Solutions will be provided only for required questions.

Question 1. (`TemperatureC.java` and `TemperatureTester.java, 20 marks`) Write a Java class to represent a temperature. The class has a single data field:

- temperatureC

The class has a single constructor that accepts an initial temperature provided as a double argument. If this argument is < -273.15, set it to -273.15.

The class has the following methods:

- setC – this method takes a single double argument and changes the value of the data field accordingly. Like the constructor his method should prevent the temperature from being less than -273.15.
- getC – this method returns the current value of the data field.
- getF – this method computes and returns the value of temperatureC in Fahrenheit.
  - $C \times 1.8 + 32 = F$
- getK – this method computes and returns the value of temperatureC in Kelvin.
  - $C + 273.15 = K$
- ***Write JavaDoc comments for this class.***

Write a tester class that performs the following.

- Prompt the user and collect a double value. Use this value to create a Temperature object.
- Show the initial temperature in C, F and K.
- Prompt the user for another value and use this value to change the temperature of your existing object. Again, show the new values of C, F and K.

```
Please enter the initial temperature: 20
The current temperature in Celsius is: 20.0
The current temperature in Fahrenheit is: 68.0
The current temperature in Kelvin is: 293.15

Please enter a new temperature: 4000

The current temperature in Celsius is: 4000.0
The current temperature in Fahrenheit is: 7232.0
The current temperature in Kelvin is: 4273.15
```

(Extra – 5 marks) In your tester create two objects instead of one (this means prompting for two initial values). Do the exact same work as listed above but show that the two object behave independently of each other.

```
Please enter the initial temperatures: 20 6000
1) The current temperature in Celsius is: 20.0
1) The current temperature in Fahrenheit is: 68.0
1) The current temperature in Kelvin is: 293.15
-----------------------------------------
2) The current temperature in Celsius is: 6000.0
2) The current temperature in Fahrenheit is: 10832.0
2) The current temperature in Kelvin is: 6273.15

Please enter a new set of temperatures: 4000 0

1) The current temperature in Celsius is: 4000.0
1) The current temperature in Fahrenheit is: 7232.0
1) The current temperature in Kelvin is: 4273.15
-----------------------------------------
2) The current temperature in Celsius is: 0.0
2) The current temperature in Fahrenheit is: 32.0
2) The current temperature in Kelvin is: 273.15
```

Question 2. (`Die.java` and `DiceTester.java, 20 marks`) Implement a class used to simulate a die (singular of dice). Your class should have two data fields, one to store the number of sides on your die and another to store which side (value) is currently facing up. Your class should have two constructors – one that takes zero arguments and creates a standard 6-sided die and a second constructor that takes a single argument. The argument should allow the user to create a die that has 4, 6, 8, 10, 12, 20 or 100 sides; if the user enters any other value then the die should default to 6 sides. Both constructors should leave the side-up data field with a random (but valid) value. Your class should have a single accessor (getValue()) that returns the current value facing up. Your class should have a single void mutator method (roll()) that randomly rolls the die – make use of your number of sides data field here. Finally, add a toString() (see chapter 8!) method to make it easy to print out the state of your die objects.

Write a complete tester program to show that your die class is fully functional. In particular you should create at least two dice, one with each constructor (get user input for the constructor that takes an argument, don't do any input checking). Then roll them independently singly and using a loop. Also, since many games use more than one die show the sum of all your dice objects. ***Write JavaDoc comments for this class.***

```
Creating two dice - one with 6 sides, enter number of sides for the second one: 20
Die[6 sides, value=3]
Die[20 sides, value=11]
Rolling each separately and printing the values:
Die 1 showing:6
Die 2 showing:3
Sum of the dice:9

for loop to roll the two dice 10 times and show the values and the sum:
Die 1Die 2Sum
4          13         17
1          17         18
5          10         15
3          19         22
4          12         16
1          2          3
1          4          5
3          8          11
2          1          3
2          6          8
3          1          4
```

(Extra – 5 marks) Add the following to your tester class. Create five more six-sided dice and use nested loops (for loop and while loop?) to determine the average number of rolls it take to get a Yahtzee (all 5 dice with the same value). Keep rolling the dice until you achieve a Yahtzee – keep track of the number of rolls it took to get it. Once you have done this, wrap it in a loop that repeats this process 1000 times. Display the maximum, minimum and average number of rolls it takes to get a Yahtzee. The output below should be in addition to the output shown above.

```
Creating five dice.
Die[6 sides, value=5]
Die[6 sides, value=4]
Die[6 sides, value=4]
Die[6 sides, value=2]
Die[6 sides, value=3]
Rolling until I find 1000 Yahtzees.  Counting the number of rolls it takes to get each one.
Found 1000 Yahtzees.  Max rolls to find one: 12057 min rolls: 1 avg rolls: 1297.22
```

Question 3. (`Car.java` and `CarTester.java`) Write a class that represents a car object. A Car has the following data fields:

- `yearModel – The model year of the car`
- `make – The manufacturer of the car`
- `speed – The speed the car is currently travelling`

A Car should have the following as well:

- A Constructor that takes as arguments the car's year and make. The speed of the care should be set to 0.
- Accessors – there should be an appropriate accessor method for each of the data fields.
- Mutators – there should be an accelerate method and a brake method. Each of these methods adds or subtracts 5 from the speed of the car respectively. Note: The speed of the car should never be less than 0 and never more than 210 (assume km/h). Your methods should make sure these limits are respected.
- *****Write JavaDoc comments for this class.*****

Write a tester class that:

- Prompts the user for the make and model of the Car
- Constructs a Car with the input given
- Accelerate once and show the speed of the Car
- Use a loop to accelerate 5 (or better yet a random number of) times and show the speed of the Car.
- Use a loop to decelerate until the Car is stopped. Display the number of times the brakes were applied.

```
Please enter the make and model of the car: Tiberon 2006
New car created! Tiberon, 2006
The current speed is 5.
The current speed is 30.
The current speed is 0, and the brakes were applied 6 times.
```

Question 4. (`SlotMachine.java` and `SlotMachineTester.java`) A slot machine has three integer data fields – one for each of the values displayed to the user. Think of them as item1, item2 and item3. It also has a single double data field called money that keeps track of how much money the player has left.

In addition it should have the following:

- A constructor that sets the value of each of the items to an appropriate random value. Remember how the slot machine works, each item has 7 possible values (this can be done by calling the spin method described below). The constructor takes a single argument an integer that sets the initial amount of money in the machine.
- A mutator called cashOut that takes no arguments and reduces the money in the machine to 0 and returns whatever the previous balance was.
- A mutator method called spin that changes the three data fields to random values.
- An accessor method called displayResults that shows the result of the spin to the user. This should return a single string with the word value of the results, not the integer value (eg. "seven, seven, cherry").
    - This is a great place for a "helper" method that is designed to take in an integer as an argument and return the appropriate word value for the item.
- A mutator method called adjustMoney that takes no arguments. Instead it is called after the spin method to either add or remove the appropriate amount of money to the player's total. This method determines if there was a win or loss. If there are two matching items the player wins $2, if all three matched they win $10, if none match they lose $1.
- ***Write JavaDoc comments for this class.***

Write a tester program that does the following:

- Prompt the user for an initial amount of cash to put into the machine. If the amount is less than 0 or greater than 100 display an error message and exit. Otherwise, use the amount to create the slot machine object and display the initial results (this is not a win or loss, just a display).
- Ask the user if they want to spin or cash out.
    - If they cash out, tell them their total and end the program.
    - If they spin proceed accordingly – have the machine spin and adjust the player's total.
        - Display the results and the new player total (money) after each spin.
- Continue to prompt the user to cash out or spin until they choose to leave or their money reaches $0.

Sample output is shown below.