CSCI 1700 Programming Assignment # 3 ( 100 pts)

**Due Date: 10/27/2015**

1. (10 pts) Chap 5 #1

**A sample dialogue:**

Creat three HotDogStands:

Creat HotDogStand s1 with default constructor: Id: 0 numberSold: 0

Creat HotDogStand s2 with Id: 2 numberSold: 0

Creat HotDogStand s3 with Id: 3 numberSold: 0

Reset s1 ID: 1

Invoke s1.justSold()

Invoke s2.justSold()

Invoke s1.justSold()

Stand 1 sold 2

Stand 2 sold 1

Stand 3 sold 0

Total sold = 3

Invoke s3.justSold()

Invoke s1.justSold()

Stand 1 sold 3

Stand 2 sold 1

Stand 3 sold 1

Total sold = 5

Process completed

1. (30 pts) Chap 5 #7 (complex number: all three parts)

**Part 1**

**A sample dialogue**:

Enter Complex number a+bi:

0.0+0.0i

You input 0.0+0.0i

complex1 is: 0.0 + 0.0i

Enter Complex number a+bi:

5.0+0.0i

You input 5.0+0.0i

complex2 is: 5.0 + 0.0i

Enter Complex number a+bi:

1.0+4.0i

You input 1.0+4.0i

complex3 is: 1.0 + 4.0i

Complex1's real part is: 0.0

Complex1's imaginary part is: 0.0

Changing complex1's real part.

Complex1 is : 3.0 + 0.0i

Changing complex1's imaginary part.

Complex1 is: 3.0 + 4.0i

complex1 is equal to complex2? false

Test copy constructor: a copy of complex2 is equal to complex2? true

Adding 1+4i and 3+4i: 4.0 + 8.0i

Subtracting 3+4i - 1+4i: 2.0 + 0.0i

Multiplying 1+4i \* 3+4i: -13.0 + 16.0i

**Part2 : A sample dialogue:**

Enter Complex number a+bi:

0.0+0.0i

You input 0.0+0.0i

complex1 is: 0.0 + 0.0i

Enter Complex number a+bi:

5.0+0.0i

You input 5.0+0.0i

complex2 is: 5.0 + 0.0i

Enter Complex number a+bi:

1.0+4.0i

You input 1.0+4.0i

complex3 is: 1.0 + 4.0i

Complex1's real part is: 0.0

Complex1's imaginary part is: 0.0

Changing complex1's real part.

Complex1 is : 3.0 + 0.0i

Changing complex1's imaginary part.

Complex1 is: 3.0 + 4.0i

complex1 is equal to complex2? false

Test copy constructor: a copy of complex2 is equal to complex2? true

Adding 1+4i and 3+4i: 4.0 + 8.0i

Subtracting 3+4i - 1+4i: 2.0 + 0.0i

Multiplying 1+4i \* 3+4i: -13.0 + 16.0i

Adding 1+4i and 3+4i (test overloaded add): 4.0 + 8.0i

Subtracting 3+4i - 1+4i (test overloaded subtract): 2.0 + 0.0i

Multiplying 1+4i \* 3+4i (test overloaded multiply): -13.0 + 16.0i

**Part 3: A sample dialogue:**

Enter Complex number a+bi:

0.0+0i

You input 0.0+0.0i

complex1 is: 0.0 + 0.0\* i

Enter Complex number a+bi:

5+0.0i

You input 5.0+0.0i

complex2 is: 5.0 + 0.0\* i

Enter Complex number a+bi:

1.0+4.0i

You input 1.0+4.0i

complex3 is: 1.0 + 4.0\* i

Complex1's real part is: 0.0

Complex1's imaginary part is: 0.0

Changing complex1's real part.

Complex1 is : 3.0 + 0.0\* i

Changing complex1's imaginary part.

Complex1 is: 3.0 + 4.0\* i

complex1 is equal to complex2: false

Test copy constructor: a copy of complex2 is equal to complex2? true

Adding 1+4i and 3+4i: 4.0 + 8.0\* i

Subtracting 3+4i - 1+4i: 2.0 + 0.0\* i

Multiplying 1+4i \* 3+4i: -13.0 + 16.0\* i

Adding 1+4i and 3+4i (test overloaded void add): 4.0 + 8.0\* i

Subtracting 3+4i - 1+4i (test overloaded void subtract): 2.0 + 0.0\* i

Multiplying 1+4i \* 3+4i (test overloaded void multiply): -13.0 + 16.0\* i

1. (Additional 10 pts) Write a complete Java program that prompts the user for a series of numbers to determine the smallest value entered and display the smallest value. Use a **do** loop to control the program repetition for finding the smallest value of a new series of numbers.

Note: You can read user input as a whole line and use class **StringTokenizer** to get individual tokens. You will need to use method from **Wrapper class** **Double** to parse the user’s input number as a **String** to a **double**.

**A sample dialogue:**

This program finds the smallest number in a series of numbers.

Please enter a list of numbers separated by comma and space:

1, 2, 5, 9.1, -10, 8, -20, -13, -19, 5, 7, 9, 200, -500, 900, -1000

The smallest number is -1000.0.

Do it again? Type y for yes, n for no.

y

This program finds the smallest number in a series of numbers.

Please enter a list of numbers separated by comma and space:

-5, 10, 30, -2000, 3000, 50, 40.9, -100, 9000, 3000, -5000, -9999, 1000.999

The smallest number is -9999.0.

Do it again? Type y for yes, n for no.

n

Process completed.

1. (Additional 5 pts) Write a Java **program** that will read in one or more lines of text and then determine the following:

* The number of uppercase letters in the text.
* The number of lowercase letters in the text.
* The number of digits in the text.
* The number of whitespace characters (i.e., blanks or newlines) in the text.
* The number of other, non-letter, non-digit, non-whitespace characters in the text.

After reading the text, the program should display the text followed by a display of all of the above statistics.

**Suggestions and hints:**

* Use a String variable to read the text into (since you do not know in advance how much text will be input).
* The nextLine() method from **Scanner** class can be used to read in more than one line into a String variable. To do this, inform user to type multiple lines of text before hitting the “Enter” key on the computer.
* You will need to use methods from wrapper class **Character.**

**a sample dialogue** (user input is one line in **bold**):

**A string is a joy forever! Enter 3 or 4 words: Am I a palindrome? Hello friends# Is this a valid**

**English sentence**

A string is a joy forever! Enter 3 or 4 words: Am I a palindrome? Hello friends# Is this a valid

English sentence

There were 113 total characters.

There were 7 upper case letters.

There were 78 lower case letters.

There were 2 digits.

There were 22 white space characters.

There were 4 other characters.

Process completed.

1. (5 pts) Chapter 6 #5 (standard deviation)

**Sample dialogue:**

--------------------Configuration: <Default>--------------------

Standard deviation of {2.0,2.0,2.0,2.0} = 0.0

Standard deviation of {2.0,2.0,2.0,4.0} = 0.8660254037844386

Standard deviation of {2.0,2.0,2.0} = 0.0

Standard deviation of {2.0,0.0,2.0,4.0} = 1.4142135623730951

Standard deviation of {2.0,0.0,2.0} = 0.9428090415820634

Process completed.

1. (20 pts) Roll Die program

When tossing one die, results of 1 through 6 are possible results. So, when tossing a pair of dice, sums of 2 through 12 are possible results. If the dice are fair, it can be shown that the probability of tossing each of these sums is:

2 or 12: ; 3 or 11: ; 4 or 10: ;

5 or 9: ; 6 or 8:; 7: 

Write a **Java program** that will repeatedly simulate tossing a fair pair of dice a number of times specified by the user (any number between 1 and 100,000), and calculate and display the total number of each sum produced along with its percentage of the total tosses (the relative frequency approximation of probability).

Your program must contain these methods:

* rollDie – this method will receive an array and the number of tosses and will simulate tossing one die the specified number of times, storing the toss results in the array.
* findSum – this method will receive the number of tosses, two arrays holding the toss results for two dice, and a third array, and will calculate the toss sums, storing these in the third array.
* tossCount – this method will receive the number of tosses, an array of toss sums, and an array of counters for the possible sums, and will examine the toss sums array and count the toss sum totals.
* display – this method will receive the number of tosses and an array of counters for the possible sums, and display the total number of tosses, and the total number of each of the possible sums, along with the probability of each possible sum.

Your method main should repeatedly obtain a number of tosses from the user and then call the other methods to handle the simulations, until the user elects to exit the program.

Notes and suggestions:

* In rollDie, use the Math.random() method to simulate each die toss. Remember Math.random() return a random number greater than or equal to 0.0 and less than 1.0. You can use (int) (Math.random() \* 6 + 1) to get an integer from 1 to 6.
* Methods tossCount and display both receive “an array of counters for the possible sums”. As noted above, the toss sums will be values 2 through 12. So, use an integer array of size 13, with all elements initialized to 0, for the toss sums counter array. Then, to count a sum of value *k* (*k* between 2 and 12), increment element *k* of the toss sums counter array.
* In display, the probability of a given sum is the percentage of tosses that produced that sum.
* In display, use the tab escape sequence \t to produce evenly spaced columns of numbers.

**a sample dialogue** (user input in **bold**):

Enter number of tosses:

**5000**

Total number of tosses = 5000

Toss Count Probability

2 149 2.98

3 274 5.48

4 416 8.32

5 539 10.78

6 704 14.08

7 823 16.46

8 718 14.36

9 576 11.52

10 390 7.8

11 279 5.58

12 132 2.64

Do another simulation? (yes or no):

**yes**

Enter number of tosses:

**25000**

Total number of tosses = 25000

Toss Count Probability

2 734 2.936

3 1371 5.484

4 2162 8.648

5 2763 11.052

6 3481 13.924

7 4113 16.452

8 3506 14.024

9 2795 11.18

10 2031 8.124

11 1382 5.528

12 662 2.648

Do another simulation? (yes or no):

**no**

Process completed.

1. (20 pts) Create a **class** **Elevator** that will simulate the operation of an elevator. Here are the details:

* Assume that for a given elevator, it must keep track of its current floor position.
* Assume that an elevator will initially start on the first floor.
* An elevator should be able to service a request to move to a specified floor. In servicing a request, the elevator should display a summary of its floor movement.
* An elevator should also be able to return its current floor position to other program modules.

Using your Elevator class, write a **Java program** that will use the Elevator class to simulate the operation of **three elevators (use an array to store the three elevators)**  traveling between the 1st and 10th floors of a building, from the perspective of a person waiting for an elevator on the first floor. The program should repeatedly do the following tasks:

* Display the current positions of each of the three elevators.
* Ask the user to choose one of the elevators to use (i.e., like pushing the “up” button for one of the elevators – we assume here that each elevator has its own “up” button).
* If the chosen elevator is not on the first floor, the driver should give the elevator a request to come to the first floor.
* Ask the user which floor she wants (i.e., like pushing the “floor number” button inside the elevator).
* Send the chosen elevator a request to move to the selected floor.

**a sample dialogue** (user input in **bold**):

Elevator Status

A B C

1 1 1

Which elevator do you want (1=A, 2=B, 3=C, or other to exit)? **1**

Which floor do you want? **3**

Starting at floor 1

Going up - now at floor 2

Going up - now at floor 3

Stopping at floor 3

Elevator Status

A B C

3 1 1

Which elevator do you want (1=A, 2=B, 3=C, or other to exit)? **3**

Which floor do you want? **5**

Starting at floor 1

Going up - now at floor 2

Going up - now at floor 3

Going up - now at floor 4

Going up - now at floor 5

Stopping at floor 5

Elevator Status

A B C

3 1 5

Which elevator do you want (1=A, 2=B, 3=C, or other to exit)? **2**

Which floor do you want? **4**

Starting at floor 1

Going up - now at floor 2

Going up - now at floor 3

Going up - now at floor 4

Stopping at floor 4

Elevator Status

A B C

3 4 5

Which elevator do you want (1=A, 2=B, 3=C, or other to exit)? **3**

Starting at floor 5

Going down - now at floor 4

Going down - now at floor 3

Going down - now at floor 2

Going down - now at floor 1

Stopping at floor 1

Which floor do you want? **6**

Starting at floor 1

Going up - now at floor 2

Going up - now at floor 3

Going up - now at floor 4

Going up - now at floor 5

Going up - now at floor 6

Stopping at floor 6

Elevator Status

A B C

3 4 6

Which elevator do you want (1=A, 2=B, 3=C, or other to exit)? **9**

Process completed.

**What you need to turn in:**

1. **Source code listing:** A printed copy of the source code for each problem. Remember to include your name in a comment at the top of your source code. Be sure to follow the “Code Style Guidelines” specified in class.
2. **Source code files:** E-mail me a copy of your source code. Send individual.java files as attachments.
3. **Working in Pairs:** If you want to work with one other student in our class on this assignment, this is acceptable provided that both members of the pair make a contribution to the solution. If you decide to work in a pair, turn in only one copy of the solution – clearly identify the name of each pair member on everything that you turn in.
4. **Late Assignments:** Please note this information given in the “Assignment Information and Guidelines” handout regarding assignments that are turned in late:

“Assignments are due by the end of the day on the specified due date (both the paper copies and the e-mail copies). If you wish to turn in the paper copy of an assignment after class, place them under my office door. **Assignments turned in late will be assessed a 20% penalty per class day late.”**