<Homework 1 – Updated on 5/11>

COMP217 Java Programming, spring 2019. Instructor: Gil-Jin Jang

Total 5 problems, 100 pts

※ Note: Your code should satisfy all the given conditions. If you have made any additional assumptions, specify them in your submitted file as comments.

1. (20%) For odd numbers r = 1, 3, 7, 9, it has a special property of reordering numbers 0 to 9 as follows

x	0	1	2	3	4	5	6	7	8	9
r	y = (r*x)%10									
1	0	1	2	3	4	5	6	7	8	9
3	0	3	6	9	2	5	8	1	4	7
7	0	7	4	1	8	5	2	9	6	3
9	0	9	8	7	6	5	4	3	2	1

As shown in the table above, numbers 0-9 are show in a different order, and we can call this mapping as an **ENCRYPTION**. Using the above table, we can define inverse mapping from the encrypted number to the original one, called **DECRYPTION**. A few examples are listed as follows:

Decrypt(6,3) = 2, Decrypt(2,3) = 4, Decrypt(1,3) = 7 (the 2^{nd} integer is the encryption **KEY**)

Decrypt(6,7) = 8, Decrypt(2,7) = 6, Decrypt(1,7) = 3

Decrypt(6,9) = 4, Decrypt(2,9) = 8, Decrypt(1,9) = 9

The purpose of problem 1 is to implement multiplication of two encrypted number. For example,

```
34(9) * 21(9) = 76 * 89 = 6764 = 4346(9)
```

Note that the conversion should be digit-by-digit.

TODO: Write a Java program that does the following:

1) Take 3 integer numbers: conversion key in {3,7,9}, and two numbers in that key
2) Convert two numbers to corresponding original numbers (DECRYPTION)
    A. You may need to build the conversion table first
    B. Write a method for decryption by table mapping
    C. Other ideas all welcomed.
3) Multiply two original numbers
4) Convert the multiplication result using the same key (ENCRYPTION)

A. For every digit, perform (key*x)%10

5) **Submission: `EncryptMult.java`** (no other names, no other files)

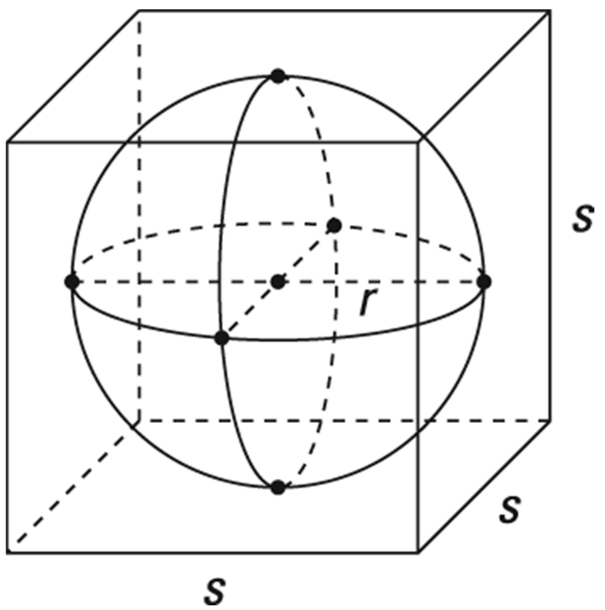
6) Clear indication of the author is very important to keep your property. **Add your name and student ID in the very first line of the source code**.

Test your program with the following examples (doesn't have to be identical, except the last number)

```
$ java EncryptMult
Encryption key r and numbers y1, y2 in that key: 3 12 23
12(3) * 23(3) = 74 * 41 = 3034 = 9092(3)
Encryption key r and numbers y1, y2 in that key: 7 12 23
12(7) * 23(7) = 36 * 69 = 2484 = 4868(7)
Encryption key r and numbers y1, y2 in that key: 9 12 23
12(9) * 23(9) = 98 * 87 = 8526 = 2584(9)
Encryption key r and numbers y1, y2 in that key: 3 342 4543
342(3) * 4543(3) = 184 * 8581 = 1578904 = 3514702(3)
Encryption key r and numbers y1, y2 in that key: 7 342 4543
342(7) * 4543(7) = 926 * 2529 = 2341854 = 4187658(7)
Encryption key r and numbers y1, y2 in that key: 9 342 4543
342(9) * 4543(9) = 768 * 6567 = 5043456 = 5067654(9)
```

---

2. (20%) The value of π can be approximated by Monte Carlo method (simulation). Monte Carlo method uses repeated random sampling to obtain numerical results. We can use volume equations of a cube and a sphere. Suppose that both are centered at the origin in a 3-dimensional space, (0,0,0), and that the radius of the sphere is r, and the side length of the cube is 2r. In mathematics, the volume of the sphere is $\frac{4}{3}\pi r^3$, and the volume of the cube is $(2r)^3 = 8r^3$. We can approximate the value of π using the following algorithm:



A. Take an integer number for the total number of samples to generate (N_a)

B. (1 sample) Generate 3 random double numbers, x, y, z in (-1, 1), and make it 3-dimensional vector

C. Compute Euclidean distance from (0, 0, 0) to (x, y, z)

$$\sqrt{x^2 + y^2 + z^2}$$

D. If distance is less than or equal to 1.0, the points belongs to the sphere; otherwise, it belongs non-sphere region.

E. Repeat N_a times, and count the

number of the points that belong to the unit sphere (N_s)

F. From the volume equations, we can derive an equation for finding the value of π.

$$\frac{4}{3}\pi r^3 : 8r^3 = N_s : N_a \quad \leftrightarrow \quad \pi = 8\frac{N_s}{N_a}\frac{3}{4} = 6\frac{N_s}{N_a}$$

TODO: Write a Java program for the above.

Submission: `ApproximatePISphere.java`

Requirements:

(1) clear indication of the author is very important to keep your property. **Add your name and student ID in the very first line** of the source code.

(2) take the user input for the number of samples to be generated (N_a), N_a > 0.

(3) use N_a for random seed (java.util.Random)

(4) random number should be in (-1, 1), not (0, 1)

(5) because the numbers are to be generated randomly, your output may be different from the given example. The evaluation will be done mostly on your code, not the output.

[Execution Examples]

```
How many samples to generate? 20
pi from 20 samples = 3.9000000, error = 2.414e-01
How many samples to generate? 100
pi from 100 samples = 3.4200000, error = 8.862e-02
How many samples to generate? 1000
pi from 1000 samples = 3.1020000, error = 1.260e-02
How many samples to generate? 10000
pi from 10000 samples = 3.1320000, error = 3.053e-03
How many samples to generate? 100000
pi from 100000 samples = 3.1336800, error = 2.519e-03
How many samples to generate? 1000000
pi from 1000000 samples = 3.1373640, error = 1.346e-03
How many samples to generate? 10000000
pi from 10000000 samples = 3.1421076, error = 1.639e-04
How many samples to generate? 20000000
pi from 20000000 samples = 3.1418052, error = 6.766e-05
How many samples to generate? 100000000
pi from 100000000 samples = 3.1416656, error = 2.323e-05
How many samples to generate? 200000000
pi from 200000000 samples = 3.1416145, error = 6.951e-06
```

3. (20%) Complete the class "**Complex**" for complex numbers, $a+bi$. The class has two members for real and imaginary parts. The skeleton code is given in the following figure.

```java
public class SimpleComplex {
   private double real;
   private double imag;

   public SimpleComplex() { set(0,0); }
   public SimpleComplex(double re, double im) { set(re,im); }
   public SimpleComplex(SimpleComplex other) { /* --- FILL --- */ }

   public double re() { return real; }
   public double im() { return imag; }
   public double abs() { /* --- FILL --- */ }

   public void set(double re, double im) { /* --- FILL --- */ }
   public void set(SimpleComplex a) { /* --- FILL --- */ }

   // arithmetic operators
   public void add(SimpleComplex a, SimpleComplex b) // this = a+b
   { /* --- FILL --- */ }
   public void sub(SimpleComplex a, SimpleComplex b) // this = a-b
   { /* --- FILL --- */ }
   public void mul(SimpleComplex a, SimpleComplex b) // this = a*b
   { /* --- FILL --- */ }

   public String toString() { /* --- FILL --- */ */

   // Override the equals method in the Object class
   public boolean equals(SimpleComplex other) { /* --- FILL --- */ }

   // Implement the compareTo method in SimpleComparable
   // returns 1 if |this| > |o|; -1 if |this| < |o|; 0 if |this| == |o|
   public int compareTo(SimpleComplex o) { /* --- FILL --- */ }
}
```

(**update**: public boolean equals(**Object** other) ➔ public boolean equals(**SimpleComplex** other)

TODO: Implement all the empty method bodies in the above skeleton. Hints are given as comments.

- **Submission: SimpleComplex.java TestSimpleComplex.java**

Requirements:

(1) Clear indication of the author is very important to keep your property. **Add your name and student ID in the very first line** of the source code.

(2) do not use "Comparable<Complex>" and "@Override". We haven't learned them yet. If you use them in your code, penalty will be given.

(3) the complete test code, "TestSimpleComplex.java" is already given in the figure below. Make your output exactly same as the example output.

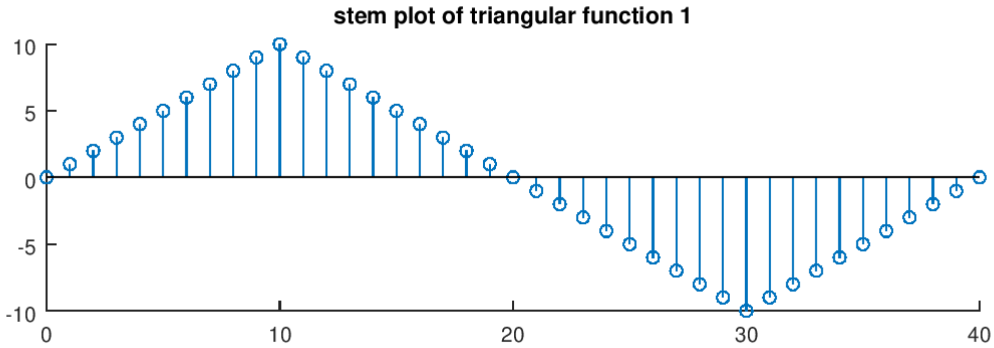
[Execution Example]

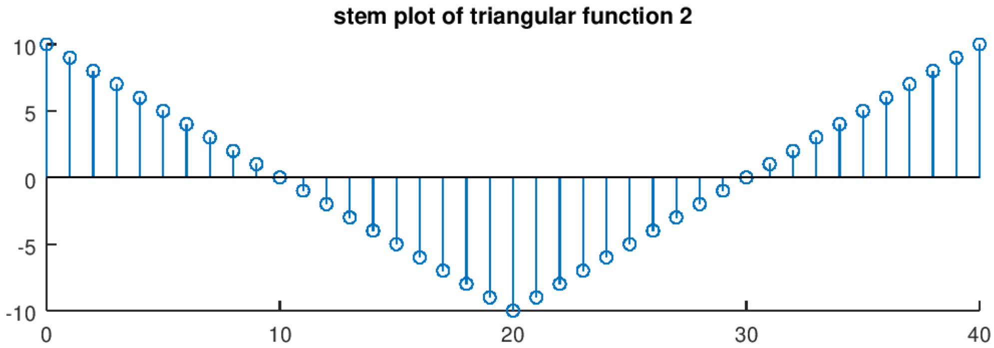
```
$ java TestSimpleComplex
c3 = (5.2+3.5i)
c3 = (2.3-7.2i)
|(5.2+3.5i)| = 6.27 < |(2.3-7.2i)| = 7.56
(5.2+3.5i) == (5.2+3.5i) = true
(5.2+3.5i) == (2.3-7.2i) = false
(5.2+3.5i) + (2.3-7.2i) = (7.5-3.7i)
(2.3-7.2i) - (5.2+3.5i) = (-2.9-10.7i)
(5.2+3.5i) * (2.3-7.2i) = (37.2-29.4i)
```

[Complete test code]

```java
import java.util.Scanner;
// Test class for SimpleComplex
public class TestSimpleComplex {
  public static void main( String[] args )
  {
    SimpleComplex c1 = new SimpleComplex(5.2,3.5);
    SimpleComplex c2 = new SimpleComplex(2.3,-7.2);
    SimpleComplex c3 = new SimpleComplex(c1);
    SimpleComplex c4 = new SimpleComplex();
    c4.set(c2);
    SimpleComplex c5 = new SimpleComplex();
    // toString()
    System.out.println("c3 = " + c3);
    System.out.println("c3 = " + c4);
    // compareTo() and method abs()
    int r = c1.compareTo(c2);
    String op;
    switch ( r ) {
      case 1: op = ">"; break;
      case -1: op = "<"; break;
      case 0: op = "=="; break;
      default: op = "?";
    }
    System.out.format("|%s| = %.2f %s |%s| = %.2f\n",
    c1.toString(),c1.abs(),op,c2.toString(),c2.abs());
    // equality
    System.out.println(c1 + " == " + c3 + " = " + c3.equals(c1));
    System.out.println(c1 + " == " + c4 + " = " + c1.equals(c4));
    // arithmetic operators
    c5.add(c1,c2);
    System.out.println(c1 + " + " + c2 + " = " + c5);
    c5.sub(c2,c3);
    System.out.println(c2 + " - " + c3 + " = " + c5);
    c5.mul(c3,c4);
    System.out.println(c3 + " * " + c4 + " = " + c5);
  }
}
```

4. (20%) Stem plot is one of the drawing methods for representing time-varying, analog signal in discrete time domain. It represents the signal value by a vertical line from 0. For example:





<u>TODO</u>: Write a program that draws the stem plots for the above triangularly shaped functions.

- **<u>Submission: `StemTriangle1.java, StemTriangle2.java`</u>**

Requirements:

(1) Clear indication of the author is very important to keep your property. **<u>Add your name and student ID in the very first line</u>** of the source code.

(2) Your program first takes integer N, which defines scale of the plot. Y-range is N to –N, total 2N+1 lines ([N, N-1, …, 1, 0, -1, …, -N+1, -N]), and x-range is 0 to 4N.

(3) Use 'O' (uppercase alphabet), '|' (vertical bar, can enter with the key above Enter with Shift key pressed), and '-' (minus sign or subtraction symbol).

(4) Display lines numbers with signs (use format "%+3d")

[Execution Example - 'java StemTriangle1']

<table>
<tr>
<td><pre>Scale? (int) 5
 +5   0
 +4   0|0
 +3   0||0</pre></td>
<td><pre>Scale? (int) 7
 +7   0
 +6   0|0
 +5   0||0</pre></td>
</tr>
</table>

```
 +2   0|||||0
 +1  0|||||||0
  0 0---------0---------0
 -1            0|||||||0
 -2             0|||||0
 -3              0|||0
 -4               0|0
 -5                0
```

```
 +4     0|||||0
 +3    0|||||||0
 +2   0|||||||||0
 +1  0|||||||||||0
  0 0-------------0-------------0
 -1                0|||||||||||0
 -2                 0|||||||||0
 -3                  0|||||||0
 -4                   0|||||0
 -5                    0|||0
 -6                     0|0
 -7                      0
```

```
Scale? (int) 10
+10           0
 +9          0|0
 +8         0|||0
 +7        0|||||0
 +6       0|||||||0
 +5      0|||||||||0
 +4     0|||||||||||0
 +3    0|||||||||||||0
 +2   0|||||||||||||||0
 +1  0|||||||||||||||||0
  0 0-------------------0-------------------0
 -1                      0|||||||||||||||||0
 -2                       0|||||||||||||||0
 -3                        0|||||||||||||0
 -4                         0|||||||||||0
 -5                          0|||||||||0
 -6                           0|||||||0
 -7                            0|||||0
 -8                             0|||0
 -9                              0|0
-10                               0
```

[Execution Example - ‘java StemTriangle2’]

```
Scale? (int) 5
 +5 0                    0
 +4 |0                  0|
 +3 ||0                0||
 +2 |||0              0|||
 +1 ||||0            0||||
  0 -----0----------0-----
 -1       0|||||||0
 -2        0|||||0
 -3         0|||0
 -4          0|0
 -5           0
```

```
Scale? (int) 7
 +7 0                            0
 +6 |0                          0|
 +5 ||0                        0||
 +4 |||0                      0|||
 +3 ||||0                    0||||
 +2 |||||0                  0|||||
 +1 ||||||0                0||||||
  0 -------0--------------0-------
 -1         0|||||||||||0
 -2          0|||||||||0
 -3           0|||||||0
 -4            0|||||0
 -5             0|||0
 -6              0|0
 -7               0
```

```
Scale? (int) 10
+10  0                                        0
 +9  |0                                      0|
 +8  ||0                                    0||
 +7  |||0                                  0|||
 +6  ||||0                                0||||
 +5  |||||0                              0|||||
 +4  ||||||0                            0||||||
 +3  |||||||0                          0|||||||
 +2  ||||||||0                        0||||||||
 +1  |||||||||0                      0|||||||||
  0  ----------0--------------------0----------
 -1             0|||||||||||||||||0
 -2              0|||||||||||||||0
 -3               0|||||||||||||0
 -4                0|||||||||||0
 -5                 0|||||||||0
 -6                  0|||||||0
 -7                   0|||||0
 -8                    0|||0
 -9                     0|0
-10                      0
```

---

5. (20%) Repeat the previous one to draw Stem plots of sine and cosine functions.

TODO: Write a program that draws the stem plots for sine and cosine functions.

➢ **Submission: `StemSine.java, StemCosine.java`**

Requirements:

(1) Clear indication of the author is very important to keep your property. **Add your name and student ID in the very first line** of the source code.

(2) Your program first takes integer N, which defines scale of the plot. Y-range is N to –N, total 2N+1 lines ([N, N-1, …, 1, 0, -1, …, -N+1, -N]), and x-range is 0 to ~~ceiling(2*π*N)~~ **round**(2*π*N). The ~~ceiling~~ **round** function ensures x-range should include 2π.

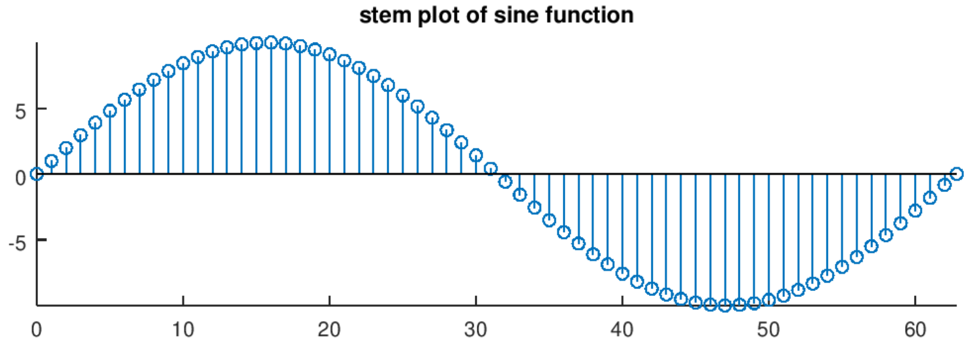
   A. *Note: x-range may become different depending on your implementation method. I just want to you a basic idea. Try yourself to find the correct range.*

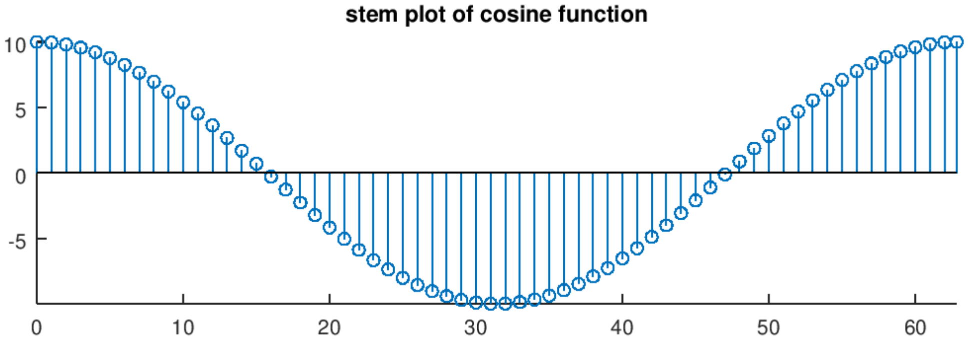
(3) Use round functions for finding integer values for sine and cosine functions (the closest integer).

(4) Use 'O' (uppercase alphabet), '|' (vertical bar, can enter with the key above Enter with Shift key pressed), and '-' (minus sign or subtraction symbol).

(5) Note: there are more than one 'O' in a single line.

(6) Display lines numbers with signs (use format "%+3d")





[Execution Example - ‘java StemSine’]

```
Scale? (int) 5
 +5        00000
 +4      00|||||0
 +3     0||||||||00
 +2    0|||||||||||0
 +1   0|||||||||||||0
 +0  0---------------0--------------0
 -1                   0||||||||||||0
 -2                    0||||||||||0
 -3                     0||||||||0
 -4                      00|||00
 -5                        0000
```

```
Scale? (int) 7
 +7           00000
 +6         00|||||00
 +5       00|||||||||00
 +4      0|||||||||||||0
 +3     0|||||||||||||||0
 +2    0|||||||||||||||||0
 +1   0|||||||||||||||||||0
 +0  0---------------------0--------------------0
 -1                         0|||||||||||||||||||0
 -2                          0|||||||||||||||||0
 -3                           0|||||||||||||||0
 -4                            0|||||||||||||0
 -5                             00|||||||||00
 -6                               00|||||00
 -7                                 00000
```

```
Scale? (int) 10
+10               000000
 +9             00||||||000
 +8           00|||||||||||0
 +7          0||||||||||||||00
 +6        00|||||||||||||||||0
 +5       0||||||||||||||||||||0
 +4      0||||||||||||||||||||||0
 +3     0||||||||||||||||||||||||0
 +2    0||||||||||||||||||||||||||0
 +1   0||||||||||||||||||||||||||||0
```

```
 +0  0------------------------------0-------------------------------0
 -1                                  0|||||||||||||||||||||||||||||0
 -2                                   0|||||||||||||||||||||||||||0
 -3                                    0|||||||||||||||||||||||||0
 -4                                     00||||||||||||||||||||||0
 -5                                       0||||||||||||||||||||0
 -6                                        0|||||||||||||||||00
 -7                                         0|||||||||||||||0
 -8                                          00|||||||||||00
 -9                                            00|||||||00
-10                                              0000000
```

[Execution Example - ‘java StemCosine’]

```
Scale? (int) 5
 +5  000                           00
 +4  |||0                        00||
 +3  ||||00                     0||||
 +2  ||||||0                   0|||||
 +1  |||||||0                 0||||||
 +0  --------0---------------0-------
 -1           0|||||||||||||0
 -2            0||||||||||00
 -3             0||||||||0
 -4              00||||00
 -5                0000
```

```
Scale? (int) 7
 +7  000                                       000
 +6  |||00                                   00|||
 +5  |||||00                               00|||||
 +4  |||||||0                             0|||||||
 +3  ||||||||0                           0||||||||
 +2  |||||||||0                         0|||||||||
 +1  ||||||||||0                       0||||||||||
 +0  -----------0---------------------0-----------
 -1              0|||||||||||||||||||0
 -2               0|||||||||||||||||0
 -3                0|||||||||||||||0
 -4                 0|||||||||||||0
 -5                  00|||||||||00
 -6                    00|||||00
 -7                      00000
```

```
Scale? (int) 10
+10  0000                                                        0000
 +9  ||||00                                                    00||||
 +8  ||||||00                                                00||||||
 +7  ||||||||0                                              0||||||||
 +6  |||||||||0                                           00|||||||||
 +5  ||||||||||00                                        0|||||||||||
 +4  ||||||||||||0                                      0||||||||||||
 +3  |||||||||||||0                                    0|||||||||||||
 +2  ||||||||||||||0                                  0||||||||||||||
 +1  |||||||||||||||0                                0|||||||||||||||
 +0  ----------------0------------------------------0----------------
 -1                   0||||||||||||||||||||||||||||0
 -2                    0||||||||||||||||||||||||||0
 -3                     0||||||||||||||||||||||||0
 -4                      0||||||||||||||||||||||0
 -5                       0||||||||||||||||||||0
 -6                        0||||||||||||||||||0
 -7                         00||||||||||||||00
 -8                           0|||||||||||00
 -9                            000||||||00
-10                               000000
```

(last page of homework 1)