<Homework Programming Assignment 2>

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

General requirements

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

2. There are 3 subproblems

3. Starting programming with your identity information is very important to protect copyrights. The first line of all your codes must contain your student ID, student name, and file name. For example:

```
// 20xxxxxxx, Jean Martin, COMP217, Homework 2-1

// B20xxxxxxx, Lea Michel, COMP217, Homework 2-2
```

4. Grading scheme
Correct identification 10%
Compliance to the given requirements 30%
Coding style 10%
Correct results for unknown inputs 50%
Cheating -100%

[2-1] (30%) Set implementation

Purpose
Set is very popular mathematical concept and used in many research areas. In this programming assignment you will write a program that manage sets with integer numbers using arrays.

Assumption
The key characteristics of mathematical sets are uniqueness, unlimited number of elements, etc. Therefore, we would like to make several assumptions to make set implementation feasible with arrays.

A1. Assume that we only have 5 sets: A, B, C, D, U, where U is a Universal set, and the others are subsets of U.

A2. There is a limit on the number of elements in a single set (including U). The computer memory is finite.

A3. Maintain the values in array “sorted in an ascending order” so that we can efficiently search a specific value.

A4. If you make any further assumption, write it in the source code as comments.

Set operations
O1. Finding the number of elements in a set (Cardinality)
O2. Determine if an element belongs to a given set
O3. Add an element to an existing set
O4. Remove a specific element from an existing set
O5. Union of two sets
O6. Intersection of two sets
O7. Difference of two sets
O8. Complement of a set (on a given Universe)
O9. Copy the set

Implementation
Define set structure using arrays to satisfy A1-A3, and write member function for O1-O7.
Here is an example skeleton that you can start from:

```
class MySetInt {
   private static final int MAX_ELEMENTS = 10;      // maximum number of elements
   private int[] elements = new int[MAX_ELEMENTS];
   private int num_elements = 0;     // actual #elements is not element.length

   public int numelms() { return num_elements; }    // O1
   public boolean isIn(int x) { /* your implementation */ } // O2
   public boolean add(int x) {       // O3, returns false if overflowed, true otherwise
       // check if num_elements is less than MAX_ELEMENTS
       // add x so that A3 is sorted as well after the addition
   }
   public void remove(int x) { /* ... */ }   // O4
   public boolean union(MySetInt X, MySetInt Y) { /* returns false if overflowed */ }
        // O5: this = X U Y
   public void intersection(MySetInt X, MySetInt Y) { /* ... */ }  // O6: this = X and Y
   public void difference(MySetInt X, MySetInt Y) { /* ... */ }    // O7: this = X - Y

   public void complement(MySetInt X, MySetInt U) { /* ... */ }
   // O8: this = X^c where U is universal set

   public void print() { /* prints the set as '{1 2 3}' */ }
   public void copy(MySetInt X) { /* ... */ }       // O9: this = X

   /* add your own methods or variable if necessary */
}

public class TestMySetInt {
   public static void main(String args[])
```

```
  {
    // deleted on 6/13
    // the code in the below is from my OLD implementation and does not use input files
    // so please DO NOT USE
    // write your own test code so that
    // 1. Creates sets U, A, B, C, D
    // 2. Open and read the input file
    // 3. Decoder the input (for example, "add U 1"
    // 4. Perform set operations
    /*
    Scanner sc = new Scanner(System.in);

    // create
    // 0: U, Universal set
    // 1, 2, 3, 4: set A, B, C, D
    MySetInt[] mysets = new MySetInt[SetNames.length];

    for (int i=0; i<mysets.length; i++) {
      mysets[i] = new MySetInt();
    }
    */

    /* your implementation */
  }
}
```

Requirements

1. "Set" is already defined in Java, so use different name such as "MySetInt".
2. For every set operation that increases the size of the set, always check if the number of elements exceeds MAX_ELEMENTS
3. If the value is already in the set, do not add, to satisfy uniqueness.
4. Maintain the values in array "sorted in an ascending order" always so that we can efficiently search a specific value. Whenever there are changes on the elements, the old values should move to make a room for the new values, or to suppress the empty spot(s).
5. (Z = X (op) Y) For Union, Intersection, and Difference, two sorted sets are merged. One way to Union sorted list is, find the smaller one from the 2 sets, and add it to the end of Z, which should be initially empty. Also the results of intersection and difference should be ordered as well.
6. For union and add, returns false if overflowed (exceeded the MAX_ELEMENTS), and display 'overflowed' if it returns false.
7. After every command print the set, in the form of '{1 2 3}' if it has 1, 2, 3, for example (no space between the last end closing brace)
8. If 'exit' is the command, exit the program.

Input

```
(command) (set_name)
(command) (set_name) (element(s))
(command) (set_name) (set_name)
(command) (element(s))
( exit  )

Command = numel isin add remove copy union intersection difference complement
Set_name = A B C D U
```

Sample Output

`$ java TestMySetInt input_sample_1_hw2-1.txt`
`{1}`
`{1 2}`
`{1 2 3}`
`{1 2 3 4}`
`{1 2 3 4 5}`
`{1 2 3 4 5 6}`
`{1 2 3 4 5 6 7}`
	`[input_sample_1_hw2-1.txt]`
	`add U 1`
	`add U 2`
	`add U 3`
	`add U 4`
	`add U 5`
	`add U 6`
	`add U 7`

{1 2 3 4 5 6 7 8} {1 2 3 4 5 6 7 8 9} {1 2 3 4 5 6 7 8 9 10} {1 2 3 4 5 6 7 8 9 10} {2 3 4 5 6 7 8 9 10} {2 4 5 6 7 8 9 10} {2 4 6 7 8 9 10} {2 4 6 8 9 10} {2 4 6 8 10} {1 3 5 7 9} {1 3 5 7 9} {1 3 4 5 7 9} {1 2 3 4 5 6 7 8 9 10} {4} {1 3 5 7 9} {2 4 6 8 10}	add U 8 add U 9 add U 10 copy A U remove A 1 remove A 3 remove A 5 remove A 7 remove A 9 complement B A add B 1 add B 4 union C A B intersection D B A difference C B A copy C A exit
$ java TestMySetInt input_sample_2_hw2-1.txt {1} {1 4} {1 2 4} {1 2 4 5} {1 2 3 4 5} {16} {16 17} {3 16 17} {3 12 16 17} {3 12 16 17} {3 12 16 17} {1 2 3 4 5 12 16 17} {1 2 3 4 5 12 16 17} {1 2 3 4 5 12 16 17 22} {1 2 3 4 5 12 16 17 22 25} overflow overflow {3 12 16 17} {22 25}	[input_sample_1_hw2-2.txt] add A 1 add A 4 add A 2 add A 5 add A 3 add B 16 add B 17 add B 3 add B 12 add B 16 add B 3 union C A B union D C A add C 22 add C 25 add C 20 add C 19 intersection A C B difference B C D exit

Submission: TestMySetInt.java

(30%) Homework 2_2 – class Complex with interface Comparable<Complex>

Complete the class "**Complex**" for complex numbers, *a+bi*. The class has two members for real and imaginary parts, and should extend abstract class Number and implement interface Comparable<Complex>.

The skeleton code can be downloaded from the website, "Complex_template.java". Hints are given in the code skeleton. Implement all the empty method bodies;

```
public class Complex extends Number implements Comparable<Complex> {
  private double real;
  private double imag;

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

  public double re() { return real; }
  public double im() { return imag; }
  public double abs() { /* FILL */ }
  // absolute value of the complex value, real^2 + imag^2

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

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

  @Override
  public String toString()
    // should return a String in the form of "(3.1+6.4i)" or "(-3.1-6.4i)"
    // according to the signs. Double values are represented by
    // 1 digit after the decimal point
    { /* FILL */ }

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

  @Override
  // returns integer-casted absolute value
  public int intValue() { return (int)doubleValue(); }

  @Override
  // Implement the abstract floatValue method in Number
  public float floatValue() { return (float)doubleValue(); }

  @Override
  // returns absolute value
  public double doubleValue() { /* FILL */ }

  @Override
  // Implement the abstract longValue method in Number
  public long longValue() { return (long)doubleValue(); }

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

The test code, hw2_2.java, is provided.

```
/* file hw2_2.java */
mport java.util.Scanner;

// Test class for Complex
```

```
public class hw2_1 {
  ////////////////////////////////
  // main function for testing
  ////////////////////////////////
  public static void main( String[] args )
  {
    Complex c1 = new Complex(5.2,3.5);
    Complex c2 = new Complex(2.3,-7.2);
    Complex c3 = new Complex(c1);
    Complex c4 = new Complex();
    c4.set(c2);
    Complex c5 = new Complex();

    // 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);
  }
}
```

The expected output is

```
$ javac hw2_2.java
$ java hw2_2
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)
```

[Submission] Complex.java, hw2_2.java

(40%) Homework 2-3 – class GenericMatrix

[**Update on 6/11**: Some java versions have problems in returning "E[ ][ ]" in **addMatrix** and **multiplyMatrix**. The return type is changed to "**void**" and a matrix to store output results are given as arguments. Test class hw2_3 is updated accordingly (use "**hw2_3_B.java**").
Complete a generic class "**GenericMatrix**" for matrix arithmetic. This class implements matrix addition and multiplication common for all element types.

UML

GenericMatrix<E extends Number>	
#add(o1: E, o2: E): E	➢ Abstract method for adding two elements of the matrices, required by addMatrix and MultiplyMatrix
#multiply(o1:E, o2: E): E	➢ Multiplies two elements of the matrices
#zero(): E	➢ Defines zero for the matrix element. It may be useful in method mutlipleMatrix() --- 2019/6/13.
#str(o: E): String	➢ Abstract method for element string, required for methods toString
+addMatrix(result: E[][], matrix1: E[][], matrix2: E[][]): void	➢ Adds two matrices (updated). The first argument is to store the addition results, should be allocated in the calling method
+multiplyMatrix(result: E[][]. matrix1: E[][], matrix2: E[][]): void	➢ Multiplies two matrices (updated). Same as the above
+toString(matrix: E[][]): String	➢ Generates a string to display the matrix

You will be provided 2 complete code files: ~~*hw2_3.java*~~ **hw2_3_B.java** (updated) for testing your codes, and IntegerMatrix.java for the example concrete implementation to guide you.

You will also be provided 3 template codes to complete: GenericMatrix_template.java, DoubleMatrix_template.java, and ComplexMatrix_template.java.

1) Rename GenericMatrix_template.java to GenericMatrix.java, with filling out the missing methods: addMatrix() and multiplyMatrix(), to handle all types extended from Number.
2) Rename DoubleMatrix_template.java to DoubleMatrix.java, and fill in the bodies of methods add(), multiply(), zero(), and str().
3) Rename ComplexMatrix_template.java to ComplexMatrix.java, and fill in the bodies of methods add(), multiply(), zero(), and str(). It uses class Complex, so put your Complex.java in the same folder, and reuse the methods as much as possible.
4) Note that in testing your code, unknown input files will be used. So ensure all your implementation is correct according to the definition.

Example output

```
$ javac hw2_3.java
$ java hw2_3
m1 + m2 is
 2 3 4
 6 7 8
 1 1 1

m1 * m2 is
 5 5 5
 14 14 14
 3 3 3

m1 + m2 is
 0.0 -0.1 1.2
 1.6 2.7 0.6
 1.1 2.1 0.9

m1 * m2 is
 -0.1 1.1 1.1
 0.2 1.3 1.1
 0.1 1.3 1.1

m1 + m2 is
 (2.0+1.0i) (3.0+2.0i) (4.0+3.0i)
 (3.0+2.0i) (4.0+3.0i) (5.0+4.0i)

m1 * m3 is
 (-5.8+9.0i) (-2.8+13.5i)
 (-2.8+13.5i) (3.3+18.0i)
```

[Submission] hw2_3.java, GenericMatrix.java, IntegerMatrix.java, DoubleMatrix.java, ComplexMatrix.java