CSC260 Homework 7 (CH10)

- 50 points in total
  - 10 points from UML diagrams (No MPL in HW7)
  - 20 points from class activities
  - 20 points from Programming questions.
- Submit your homework at Canvas/Assignments/HW7
- If you have any questions about the homework, please send me an email (start with CSC260 with a section number) or open a Discussion on Canvas.

1. UML diagrams (10 points) - No MPL in HW7

You should refer to Chapter 9 class activities for the classes.

1. (3 pt) Draw a class diagram of a `Circle class`. Instantiate two objects (name c1 and c2) with different radius and draw object diagrams. (2 points for class diagram and 1 point for object diagram)
2. (3 pt) Draw a class diagram of a `CircleWithStaticMembers class`. Instantiate two objects (name c1 and c2) with different radius and draw object diagrams. (2 points for class diagram and 1 point for object diagram)
3. (4 pt) Draw a class diagram of a `Stack class`. Instantiate two objects (name s1 and s2) of a stack class and draw object diagrams. (2 points for class diagram and 2 points for object diagram)

You can use UML diagram tools such as StarUML, or you can draw the diagrams on a paper to take pictures of them to submit. Copy your UML diagrams in the UML directory.

3. Programming (20 points)

1. You should make the program that returns correct answer.
2. You should print out the results and paste them as a comment.
3. For getting outputs, use the inputs from the sample run. Some programs don't need an input, then print out and copy the results.

4. Copy only Java files for submission; copy only the Java files in the `programming` directory.
5. Students earn 100% when they get correct answers and copied results, 60% when they get wrong answers, 0% when they can't compile the Java source or no answers copied.

3.1 LargeFactorial.java (5 points)

To get 5! (5 factorial), you need to compute $5 \times 4 \times 3 \times 2 \times 1$. In general N! (N factorial) means $N \times (N-1) \times ...1$. Note that the factorial of an integer value can be very large, so you need to use BigInteger to compute the factoral of any integer.

1. In this program, you compute 50! and prints the result.
2. You implement factorial method that returns the factorial value (BigInteger type value) from a given input (long type).
3. You need to import `import java.math.*;`.

Hint 1

The value 1 is described as follows using BigInteger.

```
BigInteger result = BigInteger.ONE; // 1 in BigInteger
```

Hint 2

You need to use BigInteger object for dealing with large values without overflow.

```
BigInteger result = BigInteger.ONE; // 1 in BigInteger
for (int i = 1; i <= n; i++)
    result = result.multiply(new BigInteger(i + "")); // You need a string. Concatenation ar

return ???;
```

3.2 MyInteger.java (5 points)

Design a class named MyInteger. The class contains the following:

- An int type member field (variable) named `value` that stores the int value represented by this object.
- A constructor that creates a MyInteger object from a int type argument.
- A getter method (getValue()) that returns the `value`.
- The methods isEven(), isOdd(), and isPrime() that return true if the value in this object is even, odd, or prime, respectively.

- The **static** methods isEven(int), isOdd(int), and isPrime(int) that return true if the specified value is even, odd, or prime, respectively.
- The **static** methods isEven(MyInteger), isOdd(MyInteger), and isPrime(MyInteger) that return true if the specified value is even, odd, or prime, respectively.
- The methods equals(int) and equals(MyInteger) that return true if the value in this object is equal to the specified value.
- A static method parseInt(char[]) that converts an array of numeric characters to an int value.
- A static method parseInt(String) that converts a string into an int value.

```
n1 is even? false
n1 is prime? true
15 is prime? false
3539
3539
n2 is odd? false
45 is odd? true
n1 is equal to n2? false
n1 is equal to 5? true
```

Hint

```
    // member field (variable)
    private int value;

    public int getValue() {
        return ???;
    }

    // constructor
    public MyInteger(int value) {
        this.value = ???;
    }

    // isPrime() method uses the equivalent static method
    public boolean isPrime() {
        return isPrime(value);
    }

    // this is the static method
    public static boolean isPrime(int num) {
        // You can use the algorithm that we discussed in the class.
        ???
    }
```

```
// isPrime method with a parameter of MyInteger
// 1. You can use getValue() method to get the integer value
// 2. You can reuse the static isPrime method
public static boolean isPrime(MyInteger o) {
    return isPrime(o.getValue());
}

// same idea
public boolean isEven() {
    return isEven(value);
}

public boolean isOdd() {
    return isOdd(value);
}

public static boolean isEven(int n) {
    ???
}

public static boolean isOdd(int n) {
    ???
}

public static boolean isEven(MyInteger n) {
    return isEven(n.getValue());
}

// you can define equals for an object
public boolean equals(int anotherNum) {
    return value == anotherNum;
}

public boolean equals(MyInteger o) {
    return value == o.getValue();
}

public static int parseInt(char[] numbers) {
    // numbers consists of digit characters.
    // For example, if numbers is {'1', '2', '5'}, the return value
    //  should be 125. Please note that
    // numbers[0] is '1'
    // numbers[1] is '2'
    // numbers[2] is '5'
```

```
        int result = 0;
        for (int i = 0; i < numbers.length; i++) {
            result = result * 10 + (numbers[i] - '0');
        }

        return result;
    }

    // You may mention this when you covered Ch8
    public static int parseInt(String s) {
        // s consists of digit characters.
        // For example, if s is "125", the return value
        //  should be 125.
        // You should use charAt() to access each element in a string

        ???

        return result;
    }
```

3.3 AproximateE.java (10 points)

(Approximate e) Programming Exercise 5.26 approximates e using the following series:

$$e = 1 + 1/1! + 1/2! + 1/3! + ... + 1/i!$$

In order to get better precision, use BigDecimal (this is similar to BigInteger, google it if necessary) with 25 digits of precision in the computation. Write a program that displays the e value for i = 100, 200, . . . , and 1000.

```
The e is 2.718281828459045235360296O for i = 100
The e is 2.7182818284590452353603060 for i = 200
The e is 2.7182818284590452353603160 for i = 300
The e is 2.7182818284590452353603260 for i = 400
The e is 2.7182818284590452353603360 for i = 500
The e is 2.7182818284590452353603460 for i = 600
The e is 2.7182818284590452353603560 for i = 700
The e is 2.7182818284590452353603660 for i = 800
The e is 2.7182818284590452353603760 for i = 900
The e is 2.7182818284590452353603860 for i = 1000
```

Hint 1

The value 1 is described as follows using BigDecimal.

```
BigDecimal e = BigDecimal.ONE; // 1
```

Hint 2

The usage of BigDecimal is similar to BigInteger, but you should use divide and add methods for this homework assignment. Google these methods if necessary.

```
for (int i = 1; i <= 1000; i++) {
    item = item.divide(new BigDecimal(i + ""), 25, BigDecimal.ROUND_UP);
    e = e.add(item);
}
```

Hint 3

- I added the programming answer (ApproximateE-answer.java), but use this answer only when you spend too much time on this homework. I ask you to solve this question on your own or with your friends.