CS 211, All Sections
Project 1
due Sunday, September 13 at midnight

The objective of this project is to implement a set of number manuplation methods, exercising skills in writing programs that utilize conditions, loops, and arrays

---

Overview:

1. Create the Java class `NumberProcessor` using the provided template.
2. Implement all the methods described below.
3. Test your code for correctness. You may use the provided tester module, or create a `main` method to perform your own tests.
4. Prepare the assignment for submission and submit it.

The ten digits or numbers that we learn early in life influence our daily life in far more ways we could ever imagine. These digits and the infinite array of other numbers they create represent age, height, license plate number, PIN numbers, bank account numbers, sequence/series of numbers, factors, squares, Fibonacci numbers, perfect numbers, and the list goes on. Over time, many of the infinite arrays, or patterns, of numbers derivable from the basic ten digits have been classified as a variety of number types. Below are some basic definitions of terms. We will create a class (`NumberProcessor`) with static methods, which can be called to compute the result of some of the definitions; these methods are described below.

Rules

1. This project is an individual effort; the Honor Code applies.
2. You may not import any extra functionality besides the default. For example, `System` and `Math` are imported by default and thus may be used, whereas something like `ArrayList` must be explicitly imported so it is disallowed.
3. The `main` method will not be tested; you may use it any way you want.

Instructions: Download the template source for `NumberProcessor.java`:

- https://cs.gmu.edu/~tmengis/courses/FA20/NumberProcessor.java

This file contains a template code for the project. Our task will be to fill in each of the methods as described. Whenever you implement a method, be sure to delete the line which throws an exception. The exceptions are placeholders which ensure that the unit tester runs correctly even when a method hasn't been implemented yet.

Number Methods:

- *(10 pts)* In mathematics, a divisor of a number divides the number with out a remainder. For example, the positive divisors of 8 are 8, 4, 2, 1, the positive divisors of 1184 are 1184, 592, 296, 148, 74, 37,32, 16,8, 4, 2, 1. There are numbers (let's call them `Excess` numbers) where the sum of all **positive divisors** of the number (including itself) is greater than twice the number. For example, 12 is such a number as the sum of all its positive divisors, 1 + 2 + 3 + 4 + 6 + 12 = 28 > 24 (2 * 12). Whereas 10 is not `Excessive` as 1 + 2 + 5 + 10 = 18 < 20 (2 *10). Write a method named `isExcessive` that returns true if a number is `Excessive`, false otherwise. Use the method signature:

  ```
  public static boolean isExcessive(int input)
  ```

input	return
0	false
11	false
24	true
33	false
44	false
56	true
66	true
95	false
101	false
102	true
945	true

- *(10 pts)* In mathematics there are special numbers whose value is the sum of $x^y + y^x$, where x and y are integers greater than 1. Let's call these numbers `Power` numbers. Write a method that accepts an integer and returns true if the number is `Power`. Use the method signature:

```
public static boolean isPower(long num)
```

input	return
6	false
8	true ($2^2 + 2^2$)
10	false
17	true ($2^3 + 3^2$)
34	false
100	true ($6^2 + 2^6$)
150	false
593	true($9^2 + 2^9$)
1125	false

- *(10 pts)* Consider an even digit integer n. If we can factor the number using two integers (a and b), whose product gives the number n and with the following characteristics:
  - Both a and b contain half the number of digits in the integer n. For example, if the number is 2568, a and b should be two digits numbers.
  - n contains the digits from both a and b. For example for n= 1530, a = 30 and b= 51. a * b = n and n contains all the digits in a and b (3, 0, 5 and 1).
  - Both a and b cannot have trailing 0 at the same time, i.e., at most one of the numbers can have trailing 0.

Let's call such numbers `Squad` numbers. Define a method named `isSquad` that accepts an integer and returns true if the number is `Squad`, false otherwise. Use the method signature:

```
public static boolean isSquad(long num)
```

input	return
1530	true (30 * 15)
109	false(odd number of digits)
1002	false
1395	true (93 * 15)
2187	true (27 * 81)
126000	false(does not meet the 3rd condition)
150300	true (300 * 501)

Note that n can have multiple distinct pairs of a and b.

- *(10 pts)* Consider the following sequence that contains a series of positive numbers.

  ```
  1, 6, 15, 28, 45, 66, 91, 120, 153, 190, 231, ...
  ```

  Let's call the sequence `MaSequence`. For n=1, `MaSequence` = 1, for n=2, it is 6, for n=7, `MaSequence` = 91. Write a method that returns the n^{th} `MaSequence` number. The method returns 0, if n is ≤ 0. Use the method signature:

  ```
  public static int maSequence(int num)
  ```

- *(10 pts)* Consider the following pattern of a number to create a sequence.

  Consider 7: $7^2 = 49$; $4^2 + 9^2 = 97$; $9^2 + 7^2 = 130$; $1^2 + 3^2 + 0^2 = 10$; $1^2 + 0^2 = 1$.

  Consider 392: $3^2 + 9^2 + 2^2 = 94$; $9^2 + 4^2 = 97$; $9^2 + 7^2 = 130$; $1^2 + 3^2 + 0^2 = 10$; $1^2 + 0^2 = 1$

  Let's call such numbers `OneSummative`. Define a method named `isOneSummative` that accepts an integer and returns true if the number is `OneSummative`, otherwise it returns false. *(Hint: in order to avoid infinite loop, calculate the values of the method one step and two steps ahead. If the two values are the same, that is an indication of repetition.).*Use the method signature:

  ```
  public static boolean isOneSummative(int num)
  ```

Array Methods:

- *(10 pts)* An array is called `EvenDual`if it has the following properties:
  - The value of the first element equals the sum of the next two elements, which equals to the next three elements, which equals to the sum of the next four elements, etc.
  - It has a size of `x*(x+1)/2` for some positive integer `x` .

  Write a method named `isEvenDual()` that returns true if the array is `EvenDual` false otherwise. Use the following signature for your method.

```
public static boolean isEvenDual(int array[])
```

input array	output
{6, 2, 4, 2, 2, 2, 1, 5, 0, 0}	true (6=2+4=2+2+2=1+5+0+0)
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, -2, -1}	true
{2, 1, 2, 3, 5, 6}	false (2 !=1+2 != 3+5+6)
{}	false
{0,0}	false (does not meet the second criterion)
{1,0,1,0,1,0,1,0}	false

- *(10 pts)* An array is called `IncrementalArray` if for the given positive integer n, it produces an array with incremental pattern.

```
if n = 1, it will produce {1}

if n= 2. it produces {1, 1, 2}

if n= 4. it produces {1, 1, 2, 1, 2, 3, 1, 2, 3, 4}

if n= 6. it produces {1, 1, 2, 1, 2, 3, 1, 2, 3, 4,1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6}
```

  Define a method `incrementalArray` that accepts a positive n and returns corresponding `IncrementalArray`. Use the following signature for your method.

```
public static int[] incrementalArray(int n)
```

  Note that the size of the array is 1 + 2 + 3 +....+ n.

- *(10 pts)* An array is called `Divisible` if it can be divided into two non-empty sub arrays, where the sum of elements of the first sub array equals the sum of elements of the second sub array. The order of elements in the sub arrays should be the same as the original array. Define a method `isDivisible` that returns true if the array is `Divisible`, false otherwise. Use the following signature for your method.

```
public static boolean isDivisible(int array[])
```

input array	output
{6, 2, 4, 2, 2, 2, 1, 5, 0, 0}	true ({6,2,4} and {2,2,2,1,5,0,0})
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, -2, -1}	true ({0,0,0,0,0,0,0,0,0,0} and {1,1,1,-2,-1})
{2, 1, 2, 3, 5, 6}	false
{}	false
{0,0}	true ({0} and {0})
{1,0,1,0,1,0,0,0}	false

- *(10 pts)* An array is called `ConsecutiveDual` if it contains consecutive elements (greater than 1 element) of same value. For example, the array `{1, 2, 3, 3, 4, 5}` and `{ 4, 4 , 4 , 4, 4 }` are `ConsecutiveDual` arrays whereas `{10, 9, 8, 7, 8, 9}` and `{0,1,0,1,0,1}` are not. Define a method `isConsecutiveDual` that returns true if the array is ConsecutiveDual, false otherwise. Use the following signature for your method.

  ```
  public static boolean isConsecutiveDual(int array[])
  ```

- *(10 pts)* An array is called `PairArray` if exactly one pair of its elements sum to 10. For example, {4,10,14, 0} is `PairArray` as only 10 and 0 sum to 10. The array {10,3,0,15,7} is not `PairArray` as more than one pair (10,0) and (3,7) sum to 10. {4,1,11} is not also `PairArray` as no pair sums to 10. Define a method `isPairArray` that returns true if the array is `PairArray`, false otherwise. Use the following signature for your method.

  ```
  public static boolean isPairArray(int array[])
  ```

HONORS SECTION ONLY (+20%):

This portion is required for honors section students; it is not worth any credit for the other sections but you're welcome to try it if you have extra time. Honors section students need to score 120 points to get a perfect score on this assignment. You will see a number score on Blackboard when grades are posted, but ignore the denominator (/100), which has to be set once for all students.

- *(10 pts)* Given an array of integers, find the consecutive elements with the largest sum. For example, if the array is `{-2, 11, -4, 13, -5, 2}` the maximum sum is 20 which is the sum of the subarray that contains `11, -4, 13`. Take a look at the following examples:

  `{1, -3, 4, -2, -1, 6}` - max sum is 7 (the bolden elements are the sub-array elements that give the max sum).

  `{-1, -3, 4, -1, -1, 2, 6, -4}` - max sum is 10.

  `{ -5, -7, -8, -4, -3, -2}` - max sum is 0. This is becuase the empty array is a sub array of any array and the max sum for an empty array is 0.

  Define a method `maxSum` that returns the maximum sum of a subarray in an array. Use the following signature for your method.

  ```
  public static int maxSum(int array[])
  ```

- *(10 pts)* Based on the question above, define a method named `maxSubArray` that returns the sub array with the maximum sum. Use the method signature:

  ```
  public static int[ ] maxSubArray(int array[])
  ```

  TESTING:

  Download the following (the `jar` file is not necessary if you are testing from within Dr Java):

  - https://cs.gmu.edu/~tmengis/courses/FA20/junit-cs211.jar
  - https://cs.gmu.edu/~tmengis/courses/FA20/P1Tester.java

  These include sample test cases. For grading, we may modify the set of test cases.

When you think your code is ready (you can run it even without the tester if you write a main method), do the following from the command prompt to run the tests.
On Windows:

```
javac -cp .;junit-cs211.jar *.java
java -cp .;junit-cs211.jar P1Tester
```

On Mac or Linux:

```
javac -cp .:junit-cs211.jar *.java
java -cp .:junit-cs211.jar P1Tester
```

In Dr Java:

- open the files, including `P1Tester.java`, and click the *Test* button.

In Eclipse:

- create a new JUnitTest Case (File --> New --> JUnit TestCase)
- Name it P1Tester
- Copy/paste the content of P1Tester.java in the new file
- Save the file
- Right click on the P1Tester.java on the Package explorer
- Chose Run As -->JUnit Test

GRADING:

The implementation of each section described above is worth the points specified at the begining of each question. Of those points, half will be based on automatic testing (the percent of test cases which passed). 1 point will be granted for style (i.e. commenting code, not writing unreadable code). The remainder will be awarded based on manual inspection. Thus, partial credit is possible, but hard-coding to pass a certain set of test cases will receive an automatic zero on the manual inspection points.

Programs which do not compile without modification will receive a zero in most cases.

SUBMISSION:

Submission instructions are as follows.

1. Let *xxx* be your lab section number, and let *yyyyyyyy* be your GMU userid. Create the directory `xxx_yyyyyyyy_P1/`
2. Place the file `NumberProcessor.java` in the directory you've just created. Do not alter the name of the class or the file, and be sure that you submit the `.java` source file rather than a `.class` file.
3. Create the file `ID.txt` in the format shown below, containing your name, userid, G#, lecture section and lab section, and add it to the directory.

   Full Name: Donald Knuth
   userID: dknuth
   G#: 00123456
   Lecture section: 004
   Lab section: 213

4. compress the folder and its contents into a .zip file, and upload the file to Blackboard.