Java Competency Test

Friday 17th November 2017

14.00 - -16.00
TWO HOURS

- Please make your swipe card visible on your desk.
- After the planning time log in using your username as **both** your username and password.

The maximum total is 25.

Credit will be awarded throughout for clarity, conciseness, useful commenting, and appropriate use of **assertions**.

Important notes:

- FIVE MARKS will be deducted from solutions that do not compile. You should comment out any code which you cannot get to compile.
- ONE MARK will be deducted from solutions that include any additional print statements. All output should be provided or look like the sample run. Please make sure you remove any of your additional print statements before submitting your final solution.

Big Nats

Integers in Java have a maximum value `Integer.MAX_VALUE`. The aim of this test is to provide a library `BigNatsLibrary` and a test program `TestBigNats` so that programmers can have natural numbers (non-negative integers) that may be much larger. A skeleton file for each of these is provided for you.

One of the main aims of this test, is to test your capabilities with processing arrays, so each *BigNat* must be held in an integer array. Each digit of the number should be held as its own element in the array. Which order you hold the digits and whether the array contains anything else, is up to you.

The expectation is that the methods will be used correctly so methods should not defend against rogue parameters. Rather the expectation is that *assertions* will cause any program that uses this library's methods incorrectly to stop with an assertion error.

Getting Started

In the `BigNats` directory you will find 3 `.java` files. `IOUtil.java` contains the input/output utility methods used in your notes and labs. You should not edit this file. You should edit the remaining two files providing your answers to this test.

- `TestBigNats.java`: contains tests for `BigNatLibrary` methods. As part of this test you will need to write the `main` method to run the tests. You are not to provide any additional tests.
- `BigNatLibrary.java`: contains stubs for all the methods that you will need to provide. You may provide additional helper methods. There are several methods that are fully provided to help you get started, but there is no need to use them. You may replace them with your own methods if you wish. There are no marks associated with doing this replacement. The main reason for replacing the provided methods is to enable you to provide a different implementation of the *BigNat* data type.

You should not change the signatures of any of the provided methods: auto-testing of your solution depends on the methods having exactly their original signatures. You should not provide any additional `.java` files.

What to do

Part I: TestBigNats.java (2 marks)

Implement the main program, which should allow a programmer to run interactive tests multiple times. For your reference, an example run has been added to the end of this document.

You are free to implement this as you wish, however methods are provided in `TestBigNats.java`. If you do not use the methods provided, please ensure that the output is exactly as shown in the above run. Note from the sample run that when a number other than *1,2,3,4* or *5* is entered this input terminates the run (just as *0* would have).

Part II: BigNatLibrary.java (23 marks)

This is the library to be implemented and which your `main` method in `TestBigNats.java` actually tests. It already has stubs for all the methods you need to implement. It is important that you do not change the signatures of any of the methods. You are free to add any helper methods.

You are required to implement your *Big Nat*s as integer arrays where each element of the array is one digit. However, what order the elements are in, where in the array they are held, and whether the array contains anything other than the digits is up to you. The two methods `zero` and `read` fully provide a specific implementation for *Big Nats.* Feel free to change this implementation (together with replacements for `zero` and `read`).

To complete this class you will need to throw away the stubs provided, remembering not to change the signatures, and provide full implementations of the following:

1. Method `void print(int[] a))` – which prints a *Big Nat* on the screen. This is not a `println` method so do not go to the next line. *1 mark*
2. Method `boolean areDigits(char[] cs))` – which takes an array of characters and returns *true* if and only if every character in the array is between *'0'* and *'9'* inclusive. *1 mark*
3. Method `boolean areEqual(int[] a, int[] b)` – which takes two *Big Nats* and returns true if and only if the two numbers have the same value. *2 marks*

4. Method `boolean isGreater(int[] a, int[] b)` – which takes two *Big Nats* and returns true if and only if the first number is larger than the second number. *2 marks*
5. Method `int[] reverse(int[] a)` – which takes a *Big Nat* and reverses the digits. For example, the `reverse` of 12345 is 54321. *2 marks*
6. Method `int[] intToBigNat(int n)` – which takes an `int` and returns it as a *Big Nat*. This is changing the representation of the number. *2 marks*
7. Method `int[] add(int[] a, int[] b)` – which takes two *Big Nats* and returns their sum as a *Big Nat*. Make sure you cover the possibility that the two numbers when added produce a number too large to store in your implementation of a *Big Nat*. This may be done using an assertion or in code. *3 marks*
8. Method `int[] sub(int[] a, int[] b)` – which takes two *Big Nats* and returns their difference as a *Big Nat*. The second number is to be taken from the first number. Make sure you cover the possibility that the second number is larger than the first. This may be done using an assertion or in code. *3 marks*
9. Method `int[] mul(int[] a, int[] b)` – which takes two *Big Nats* and returns their product as a *Big Nat*. Make sure you cover the possibility that the two numbers when multiplied produce a number too large to store in your implementation of a *Big Nat*. This may be done using an assertion or in code. *4 marks*
10. Method `int[] fib(int[] n)` – which takes a *Big Nat* n and *recursively* returns the n^{th} Fibonacci number. where

    0^{th} Fibonacci number is 0
    1^{st} Fibonacci number is 1
    2^{nd} Fibonacci number is 1
    20^{th} Fibonacci number is 6765

    Your solution can be naive. However, as a recursive solution soon slows down (try `fib(40)`) so if you have the time why not make it more efficient, while keeping it recursive? There is an *extra mark* for a more efficient implementation. *3 or 4 marks*

Total across both files: 25 marks (possibly capped to 25)

Sample Run – *java -ea TestBigNats*

```
0: Quit
1: Add two numbers
2: Subtract the second number from the first number
3: Multiply two numbers
4: Calculate a Fibonacci number using big numbers rather than integers
5: Test helper functions
Choice ->  1
Please type in two big numbers ->  12 13
25
0: Quit
1: Add two numbers
2: Subtract the second number from the first number
3: Multiply two numbers
4: Calculate a Fibonacci number using big numbers rather than integers
5: Test helper functions
Choice ->  2
Please type in two big numbers ->  12345 2346
9999
0: Quit
1: Add two numbers
2: Subtract the second number from the first number
3: Multiply two numbers
4: Calculate a Fibonacci number using big numbers rather than integers
5: Test helper functions
Choice ->  3
Please type in two big numbers ->  1000 1000
1000000
0: Quit
1: Add two numbers
2: Subtract the second number from the first number
3: Multiply two numbers
4: Calculate a Fibonacci number using big numbers rather than integers
5.:Test helper functions
Choice ->  4
Which Fibonacci number do you want ->  6
8
0: Quit
1: Add two numbers
2: Subtract the second number from the first number
3: Multiply two numbers
4: Calculate a Fibonacci number using big numbers rather than integers
5: Test helper functions
Choice ->  5
Please type in two big numbers ->  12345 12345
The numbers are equal.
The first number is not greater than the second number.
Reversing the first number gives 54321
Please type in a positive integer to convert to a big natural number ->  123456
123456
0: Quit
1: Add two numbers
2: Subtract the second number from the first number
3: Multiply two numbers
4: Calculate a Fibonacci number using big numbers rather than integers
5: Test helper functions
Choice -> 6
bye
```