CS 304 Homework Assignment 2

Due: 11:59pm, Thursday, February 6th

This assignment is scored out of 53. It consists of 2 programming questions. When you submit, you are required to create a folder with your name (Last name first, then First name), CS304, HW2, e.g., LastName_FirstName_CS304_HW2. Put all your Java programs (`*.java`) as well as output files in the same folder. Zip this folder, and submit it as one file to Desire2Learn. Do not hand in any printouts. Triple check your assignment before you submit. **If you submit multiple times, only your latest version will be graded and its timestamp will be used to determine whether a late penalty should be applied.**

Programming Questions

In the folder that is provided with this homework assignment, there are seven files:

`NumberStack.java`: an interface that defines the **`NumberStack`** ADT

`LNode.java`: a class that defines a node in linked lists

`LinkedNumberStack.java`: a linked list based implementation for the **`NumberStack`** ADT

`CharStack.java`: an array based implementation for a **`char`** stack ADT

`StackApps.java`: a class that implements binary conversion and big integer addition using **`CharStack`**

`TestStack.java`: a driver class that tests the implementations of your **`LinkedNumberStack`** and **`StackApps`** classes

`testNumbers.dat`: a binary file that contains 5 test decimal numbers and 15 pairs of big integers

P1 (28pts)

a. Completing the `LinkedStack` class

In the **`LinkedNumberStack`** class, you are required to implement the following methods:

`push(int v)` – This method adds an integer to the top of the stack.

`pop()` – This method removes an element from the top of the stack and returns the element. It throws a **`RuntimeException`** "*pop attempted on an empty stack*" if this operation is attempted on an empty stack.

`size()` – This method returns the number of elements on the stack.

Note that you are only supposed to touch the above three methods. You are NOT allowed to create any other methods, instance variables, or make any changes to methods other than these three methods or files other than "`LinkedNumberStack.java`". Points will be taken off if you fail to follow this rule.

b. Code Testing

You are provided with a test driver implemented by "**`TestStack.java`**" (**Do not make any changes to this file!**) so there is no need to write your own.

Once you have completed the methods, you can run the test. You should create a plain text file named "**`output-P1.txt`**", copy and paste the output corresponding to **Problem 1** (if your code crashes or does not compile, copy and paste the error messages) to this file and save it.

P2 (25pts)

a. Completing the `StackApps` class

In this programming question, you are required to implement two methods, a binary number conversion and a big integer addition.

(1) Binary number is the fundamental data representation in computer science. However, in most mathematical computations, people usually use decimal because it's more intuitive and simpler. In the **`StackApps`** class, the **`decToBin()`** method takes an decimal integer and converts it to a binary number. The digits of the result binary number are stored in a **`char`** stack, **`stackBinary`**. Make sure you have the digits pushed on to the stack in the correct order. For example, the decimal number 123 is converted to binary number 1111011. **Do not create any arrays or use anything from the `Character`, `Integer`, `Double`, and `String` classes in your implementation, including but not limited to `Integer.parseInt`, `Integer.toString`, `Integer.toBinaryString`, `Character.getNumericValue`, `Character.forDigit`. In general, you are only supposed to deal with the digits at the `char` and `int` level and make use of stacks.**

(2) When an integer has more than a certain number of digits, it is represented using scientific notation. The **`int`** type itself cannot be used to store numbers with more than 10 digits. In some cases, this may be inconvenient as people usually expect complete numbers and doing big integer computations are inevitable. The **`addBigIntegers()`** method reads in two numbers as strings from instance variables **`m_num1`** and **`m_num2`**. For example, when adding 181,749 to 314,739,847, they are stored in the two variables as "181749" and "314739847".

In the **`addBigIntegers()`** method, you are provided with three **`char`** stacks, i.e., **`stackNum1`**, **`stackNum2`**, and **`stackResult`**. You will need to use the first two stacks to store the two operands and save the sum of them on the third stack. Each character (digit) is retrieved from the number strings and then pushed onto the corresponding stacks. You should compute the sum of the two big numbers digit by digit. Be sure to take care of carries. **Do not create any arrays or import the `java.math.BigInteger` library or anything from the `Character`, `Integer`, `Double`, and `String`[1] classes in your implementation! Same as above, you are only supposed to deal with the digits at the `char` and `int` level and make use of stacks.**

Note that you are only supposed to touch the above two methods. You are NOT allowed to create any other methods, instance variables, or make any changes to methods other than the two method or files other than "`StackApps.java`". Points will be taken off if you fail to follow this rule.

[1] You may, however, call the **`length()`** and the **`charAt()`** methods on **`String`** parameters and variables.

b. Code Testing

In "**`TestStack.java`**" (**Again, do not make any changes to this file!**), a test driver for **`StackApps`** is provided. You are also given a data file "**`testNumbers.dat`**" that contains 5 decimal numbers that need to be converted to binary and 15 pairs of big integers for addition.

Depending on your programming environment, the data file might need to be placed in different folders so that you test driver can read it. For jGRASP, you can leave the data file in the same folder as your java files. For NetBeans, you should place it in your project folder in which you see directories like **`build`**, **`nbproject`**, and **`src`**, etc.

Once you have completed the **`decToBin`** and **`addBigIntegers`** methods, you can run the test. You should create a plain text file named "**`output-P2.txt`**", copy and paste the output corresponding to **Problem 2** (if your code crashes or does not compile, copy and paste the error messages) to this file and save it.

Grading Rubric:

Code does not compile: -10

P1.
Code compiles but crashes when executed: -5
Changes were made to things other than methods **`push`**, **`pop`**, and **`size`**: -5
Has output file: 5
Code passes 23 test cases: 23 (each test case worth 1 point)

P2.
Code compiles but crashes when executed: -5
Changes were made to things other than the required methods: -5
Arrays were used in the implementation: -15
`Character`, **`Integer`**, **`Double`**, and/or **`String`** were used in the implementation: -15
`java.math.BigInteger` was imported or used in the implementation: -15
Has output file: 5
Code passes 20 test cases: 20 (each test case worth 1 point)

Sample output for P1:

```
================ Problem 1 ================
Test 1: size() ==> [Passed]
 Expected: 0
 Yours: 0

Test 2: pop() ==> [Passed]
 Expected: a RuntimeException
 Yours: RuntimeException: "pop attempted on an empty stack"
```

```
Test 3: push(10) and then isEmpty() ==> [Passed]
 Expected: false
 Yours: false
Test 4: toString() ==> [Passed]
 Expected (from top to bottom): 10
 Yours (from top to bottom): 10
Test 5: top() ==> [Passed]
 Expected: 10
 Yours: 10
Test 6: push(20) and then toString() ==> [Passed]
 Expected (from top to bottom): 20 10
 Yours (from top to bottom): 20 10

...

Test 23: pop() and then size() ==> [Passed]
 Expected: 0
 Yours: 0

Total test cases: 23
Correct: 23
Wrong: 0
================ End of Problem 1 ================
```

Sample output for P2:

```
================ Problem 2 ================
Test 1: decToBin(1) ==> [Passed]
 Expected: 1
 Yours: 1

Test 2: decToBin(2) ==> [Passed]
 Expected: 10
 Yours: 10

...

Test 19: (32475982374590873429573249057322356324596378265837465895236
+ 23748596384726587324658736425 8) ==> [Passed]
 Expected: 32475982374590873429573249057559842288443644139084053259494
 Yours: 32475982374590873429573249057559842288443644139084053259494

Test 20: (15 +
876483276489127648723164978621398476213874619278346982 7136) ==>
[Passed]
 Expected: 876483276489127648723164978621398476213874619278346982 7151
 Yours: 876483276489127648723164978621398476213874619278346982 7151

Total test cases: 20
```

```
Correct: 20
Wrong: 0
================ End of Problem 2 ================
```