CS 284: Homework Assignment 1
Due: Thursday, February 3rd, 11:59pm

1 Assignment Policies

Collaboration Policy. Homework will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. Use of the Internet is allowed, but should not include searching for existing solutions. **Under absolutely no circumstances can code be exchanged between students.** Excerpts of code presented in class can be used.

Your code must include a comment with your name and the Stevens honor pledge.

2 Assignment

Define a class `BinaryNumber` in the package `cs284` that represents binary numbers and a few simple operations on them, as indicated below. An example of a binary number is

1011

Its *length* is 4. Note that its leftmost digit is the most significant one: it represents the decimal number $1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = 11$. You should represent things in the array in reverse order, i.e., the 1s place is the 0th index, the 2s place is the 1st index, the 4s place is the 2nd index, etc. Put another way, if the digits are in an array a of length n, then:

$$\mathsf{decimal}(a) = \sum_{i=0}^{n} 2^i \cdot a_i$$

We ask you to support a number of operations be supported. We divide them into two groups. The first is a set of basic operations; the second is slightly more challenging and addresses addition of binary numbers.

You may be tempted to simply convert binary numbers to native Java `int`s, do the work there, and then convert back. *Do not do this.* We will take off points: the goal of the assignment is to get practice with Java concepts, not to find clever implementations of arbitrary width integers. (For that, see `java.math.BigInteger`.)

2.1 Basic operations

The following operations should be supported:

- A constructor `BinaryNumber(int length)` for creating a binary number of length `length` and consisting only of zeros.
- A constructor `BinaryNumber(String str)` for creating a binary number given a string. For example, given the string `"1011"`, the corresponding binary number should be created. Anything other than 0s and 1s is an error. For this exercise you will have to use some standard String operations. These are listed in the "Hints" section below.
- An operation `int length()` for determining the length of a binary number.
- An operation `int getDigit(int index)` for obtaining a digit of a binary number given an index. The starting index is 0. If the index is out of bounds, then you should throw an `IndexOutOfBoundsException`.
- An operation `int toInt()` for transforming a binary number to its decimal notation (cf. the example given above).
- An operation `void bitShift(int direction, int amount)` for shifting all digits in a binary number any number of places to the left or right. The `direction` parameter indicates a left shift when the value is -1. When `direction` is given the value 1, the shift should be to the right. Any other value for `direction` should be seen as invalid. The `amount` parameter specifies how many digits the BinaryNumber will be shifted, and is only valid when it is nonnegative.

  For example, `'1011'` shifted right by 2 is `'10'`. `'1011'` shifted left by 2 is `'101100'`. Notice that shifting changes the length of the string. You should throw an exception if `amount` is too large to have a number left.

  Notice that shifting right decreases the number by factors of 2, while shifting left increases the number by factors of 2. These operations are equivalent to the ">>" and "<<" operators in Java.
- An operation `static BinaryNumber bwor(BinaryNumber bn1, BinaryNumber bn2)` that computes the bitwise or of the two numbers. Note that both argument BinaryNumbers must be of the same length for the input to be considered valid. The bitwise or of `'1010'` and `'1100'` is `'1110'`.
- An operation `static BinaryNumber bwand(BinaryNumber bn1, BinaryNumber bn2)` that computes the bitwise and of the two numbers. Note that both argument BinaryNumbers must be of the same length for the input to be considered valid. The bitwise and of `'1010'` and `'1100'` is `'1000'`.
- An operation `String toString()` that returns the BinaryNumber as the corresponding encoded string.

2.2 Addition of Binary Numbers

Here is an example of how two binary numbers of the same length are added[1].

	1			1			(carried digits)
	0	1	1	0	1		
+	0	1	0	0	1		
=	1	0	1	1	0	= 22	

Note that it is possible for the addition of two numbers to yield a result which has a larger length than the summands. In that case, room should be made for the extra digit—meaning the array should be copied over to a new one that is one greater in length. The number of digits should *not* increase unnecessarily.

	1	1	1				(carried digits)
		1	0	1	1	0	
+		1	1	1	0	1	
=	1	1	0	0	1	1	= 51

The `int[]` field `data` should be added to the data fields of `BinaryNumber`.

Implement the following operations:

- `void add(BinaryNumber aBinaryNumber)` for adding two binary numbers, one is the binary number that receives the message and the other is given as a parameter. If the lengths of the two BinaryNumbers do not coincide, then the smaller one should have 0's prepended to it in order to prevent errors. Note how `'101' + '1'` is the same as `'101' + '001'`. The `BinaryNumber` which receives `aBinaryNumber` should be modified with the result of addition.

2.3 Hints and Notes

- Use `IllegalArgumentException` to indicate bad arguments.
- For the `BinaryNumber(String str)` constructor, the following operations might come in handy:
  - `char java.lang.String.charAt(int index)`, which returns the char value at the specified index. An index ranges from 0 to `length()` − 1. The first char value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.
  - `int java.lang.Character.getNumericValue(char ch)`, which returns the int value that the specified Unicode character represents.
- For methods where allocating more space is necessary, it may be useful to define a `void extend(int amount)` method, that puts `amount` 0's at the end of a BinaryNumber, or a `void ensureLength(int minimumLength)`, that conditionally extends (check out `Arrays.copyOf`).

[1] Source: `https://en.wikipedia.org/wiki/Binary_number`

- We've asked you to implement operations in two styles: as static methods that return new objects (`bwand`, `bwor`) and as instance methods that modify the existing object (`bitShift`, `add`). In a real system, you'd probably want to use only one style... but this way you get more practice!

3 Submission instructions

Please submit a zipfile that contains the entire `src/` directory of your code, i.e., with the structure:

```
src/
  cs284/
    BinaryNumber.java
```

You may submit as many times as you like. *But...* if I find that people are submitting lots and lots of times, I may start to limit the number of submissions. The tests here give you instant feedback, but they're no substitute for understanding.

Your grade will be determined as follows:

- You will get 0 if your code does not compile.
- We will try to feed erroneous and inconsistent inputs to all methods. All arguments should be checked.
- Partial credit may be given for style, comments and readability.
- Your code should match the UML diagram below. In order to compile, you must have (at least) the same public interface. You can add other methods or constructors, but these have to be present and unchanged. Compilation errors may indicate that (a) your zipfile is incorrectly arranged or (b) your code's interface is incorrect.

BinaryNumber
private int data[]
public BinaryNumber(int length) public BinaryNumber(String str) public int length() public int getDigit(int index) public static BinaryNumber bwor(BinaryNumber bn1, BinaryNumber bn2) public static BinaryNumber bwand(BinaryNumber bn1, BinaryNumber bn2) public void bitShift(int direction, int amount) public void add(BinaryNumber aBinaryNumber) public String toString() public int toInt()