CSc 127B — Introduction to Computer Science II
Fall 2015 (McCann)

http://www.cs.arizona.edu/classes/cs127b/fall15/

Program #2: Two's Complement

Due Date: September 10th, 2015, at the beginning of class

Overview: The real point of this assignment is to give you a chance to work with exceptions. But you can't do anything with exceptions unless there's something that can go horribly wrong, so we need something with plenty of potential for error . . .

Most computer systems and languages store integer values using *two's complement* representation. In two's complement, positive values are represented as ordinary binary values (ex: $6_{10} = 0110_2$; I'll use a simple four–bit representation for these examples). Representing a negative value, such as that of -6_{10}, is more involved. Begin with the binary representation of the absolute value of the number (0110). Invert the bit values (1001). Finally, add 1 (1010). The two's complement representation of -6_{10} is 1010_2.

Why bother with this special representation of negative values? With it, we can perform subtraction using addition. In decimal, you know that $6 - 2 = 6 + (-2) = 4$. Using the two's complement version of -2 (which is 1110, because $0010 \rightarrow 1101 + 0001 = 1110$), we can compute $6_{10} + (-2_{10}) = 0110_2 + 1110_2 = (1)0100_2 = 4_{10}$. That leading 1–bit is lost, because we have only the four bits available to store values, leaving just 0100 as the result.

Observe that whenever the leftmost bit is a 0, the value is positive or is zero. When that bit is a 1, the value is always negative. Way back when, computers sometimes used representations that permitted the "value" -0 to exist. You can imagine the confusion that caused!

Assignment: Write a Java class (completely documented according to the class documentation guidelines, of course) named `BinaryNumber` that represents integer values in two's complement form. Here are the constructors and methods that you are to implement:

- `BinaryNumber()` – sets the new object's value to 0, using an 8–bit capacity as the default.
- `BinaryNumber(String)` – sets the new object's value to the binary value provided as a Java `String`. Ex: An argument of `"1011010"` would force the creation of the 7–bit (not 8!) binary value 1011010_2 Similarly, `"00000"` is a 5–bit value. Yes, leading zeroes count when determining the capacity for this constructor.

  The object's capacity is determined by the number of binary digits in the string, except that the smallest acceptable representation is four bits. If the constructor is given a string shorter than four digits, say `"10"`, append leading 0's to pad the size to four (0010, in this case).

  If the string contains characters other than 0 and 1, the Java exception `IllegalArgumentException` is to be thrown.
- `setValue(String)` – Changes the value held by the current object to that of the given `String`. It works as does the second constructor, except that the representation size doesn't change to match the length of the argument.

  If the length of the argument exceeds that of the current object, throw the exception `IllegalArgumentException`.
- `toString()` – Return the value of the object as a string containing the characters '0' and/or '1'. The length of the string is to match the number of bits stored by the object.
- `equal(BinaryNumber)` – returns `true` if the value of this object matches that of the argument. Note that the sizes need not match for the values to match. For example, 01101 = 0001101.

(Continued...)

- `incrementBy(BinaryNumber)` – Add the argument's value to the value currently held by this object. The result is to replace this object's previous value.

  If this object's storage can't hold the result of the addition (e.g., a 4-bit value is asked to add a big 6-bit value to itself), throw a `BufferOverflowException` and leave the value of this object unchanged.

  If both objects are holding positive values and the result would be interpreted to be negative (because of a leftmost bit of 1), throw an `ArithmeticException` and leave the value of this object unchanged. Other combinations of values (+/-, -/+, and -/-) are to be added and stored with no exceptions thrown. (I'm not saying that this is necessarily the best way to handle these situations; I'm just saying that I think there's already enough in this assignment for you to worry about.)

- `decrementBy(BinaryNumber)` – Subtract the argument's value from the value held by this object. If the argument is positive, then it should be converted to a negative two's complement value and added to the value of this object. If the given argument is already a negative value, the effect should be that of adding the corresponding positive value.

  As with `incrementBy()`, the result may be a `BufferOverflowException` or an `ArithmeticException`, and an unchanged value for this object.

Data: For this assignment, you need to create your own testing program, named `Program2.java`, that you will submit for grading along with your `BinaryNumber()` class. We will not be providing a testing program; you can use the testing programs from Program #1 and the Section #1 activity as guides for writing your own. Be sure that your `BinaryNumber` class is in a separate file (`BinaryNumber.java`) from your testing program, so as not to interfere with our testing program.

Because we will be grading your testing programs, sharing testing programs between students is not allowed. You may, however, share test cases. That is, you can post to Piazza, "Remember to test your `decrementBy()` method with an argument of zero!", but you can't post code that does that test.

By the way, be sure that your testing program checks for the exceptions that the `BinaryNumber` methods should be throwing when things go wrong. You can bet that our testing program will be looking for them.

Output: As with Program #1, the class you're creating should not generate any output of its own. Because your testing program and its output will be unique, there is no required output or output format for this assignment.

Turn In: Use 'turnin' to electronically submit your well-tested, completely-documented, and well-structured (`BinaryNumber.java` file and your equally-well tested, documented and structured `Program2.java` testing program to the `cs127bsXp02` directory (replace the 'X' with your section's letter) at any time before the stated due date and time.

Want to Learn More?

- There are other complement methods that can be used with number systems other than binary. See `http://mathforum.org/library/drmath/view/55949.html` for examples.

Other Requirements and Hints:

- We aren't requiring that you use a particular internal representation for your objects' current values. You'll hear me say this a lot this semester: *Choose the representation that best supports the operations.* (Why? It makes the code easier to write!) You can use arrays, or you can use strings. Think about what your methods need to do, and then which of those two representations will make those actions easiest to code.

- Put your external block comment for this assignment at the top of the `Program2.java` file. Of course, your `BinaryNumber` class is expected to have a class block comment ahead of it, and all other rules for documentation and coding style are to be followed.

- My usual final bit of advice: Start early!