CSE 2221 - Project 8

Task

Become competent with **NaturalNumber** objects and their associated methods. Gain familiarity with JUnit for testing methods. Gain exposure to some of the different computations performed in cryptography.

Original Project Instructions

Project 8 Instructions from CSE2221 Project Site

Program Requirements

- Implement the recursive reduceToGCD method using the algorithm written in pseudocode in the code template
- Implement the isEven method according to the method contract
- Implement the recursive powerMod method according to the method contract
- Implement the isWitnessToCompositeness method according to the method contract
- Implement the isPrime2 method according to the method contract
- Implement the generateNextLikelyPrime method according to the method contract
-

Tips, Rules, & Things to Note

- reduceToGCD should be nothing more than a single if-statement with 3 lines of code in the if-statement
- For reduceToGCD: note the parameter modes of n and m
- For is Even, use kernel methods and it will make everything insanely easy, also remember how we do simplified boolean returns (refer to slides from week 1 or 2 if you forgot, should be near the end of some slide deck)
- In isEven: note the parameter mode n (or rather, the lack of a specified parameter mode, what does this mean?)
- For powerMod, refer to the fastPower algorithm we have talk about in class and implemented in lab as an instance method, note that this new method is a static method. Make the necessary modifications from the instance method
- For powerMod, you will only mod (%) after any time you multiply n (i.e. n.multiply(something))
- Your powerMod method should follow the basic structure of a recursive function, refer to last project's instructions of my webpage if you forgot
- You can do the powerMod method with 3 NaturalNumber objects and 1 constant NaturalNumber object (so 4 in total) besides the formal parameters
- isWitnessToCompositeness should test the first two facts of Fermat's thing (note: the facts tell us if a number is prime, so we need to think of the opposite of these facts as proof for compositeness)
- Again, watch how many NaturalNumber objects you are creating, there shouldn't be more than 5 in total (there could be less though)
- For isPrime2, copy the method body of isPrime1 and then just change what is inside the else block
- For isPrime2, use a while loop that is the MOST efficient possible (refer to what we have done in the past for efficient solutions if you are confused (i.e. getChildElement in project 4 and 5))

- For isPrime2, when generating your random number, the randomNumber function generates a number [0, n], but we want a random number [2, n-2], so how can we transform n before the randomNumber call then shift our number that is returned from randomNumber to where it fits our necessary distribution?
- Have your number of iterations equal to 50 for simplicity
- Don't overthink generateNextLikelyPrime, just go one by one (sometimes more than one ***WINK WINK***) incrementing n until you get an n that isn't prime according to isPrime2, think 6 lines of code max
- Use your intuition you have during the testing lecture for creating unit tests, also make sure you check the parameter modes in your unit tests (restores, clears, updates, etc.), remember small boys and big boys

Steps

1. Copy and paste the *ProjectTemplate* project to create a new project folder for this project
2. Name the project *CryptoUtilities*
3. Open the *src* folder, then open *(default package)*
4. Rename any ONE file to *CryptoUtilities.java*
5. Delete the other files
6. Open *CryptoUtilities.java*
7. Go to this page and copy and paste the source code there into *CryptoUtilities.java*
8. Create a new JUnit test fixture in the *test* folder, name is *CryptoUtilitiesTest.java*
9. Once completed and convinced your code looks good: Zip it up with the naming scheme I recommend and submit to Carmen
10. **If you do not put your name in the author tag and/or do not provide sufficient comments I, myself, will penalize you (unless you aren't in my section). It is unacceptable to skip over these two crucial steps** – To provide further context, in my QA internship after my sophomore year, I didn't put my name on a few of the files I made and my manager (jokingly) wrote his name in the author spot during my code review then said to me "It looks like I made these files... so what have you done all summer? Without your name in the author tag is looks like nothing." – Thankfully I have always been a hefty commenter, but I do not want my students going to internships and jobs and being the jerks in the office who don't write comments. DO IT. YOU WILL LOVE YOURSELF LATER.