Assignment #3: Classes & Fundamental Data Types

*Due: Sunday, October 14th @ 11:59PM*

*Total Possible Points: 15*

How to Submit

* Moodle assignment (no emails or hardcopies accepted)
* Submit IDE project in ZIP or RAR format as *Assignment2.zip* or *Assignment2.rar*

Goals

* To implement a Java class based on requirements
* To choose appropriate data types based on the data used in a program.
* To test the capabilities of Java's fundamental data types using different inputs
* To use casting and Java's Math class to solve problems.
* To document your classes effectively and generate Javadocs for your classes

|  |  |
| --- | --- |
| Forbidden | If you have trouble getting your Javadoc tool to work, look at this help page:  *Generating Javadocs:*  <https://jpgrady28.azurewebsites.net/Home/Docs/235> |
|  |  |

## Your Task

One way hackers try to infiltrate a target that uses encryption is to discover - or "break" – the *key* target uses to encrypt and decrypt data. In cryptography, a *key* is simply a sequence of bits, usually between 40 and 2056 bits long.

One method of breaking the key is to perform a *brute-force attack*: try all possible combinations of keys until a match is found. In theory, the longer the key, the longer the time required to break it, on average. On the other hand, if you have a fast machine that can test millions of keys per second, you may be able to break the key much faster…maybe.

**Your task is to create a** KeyBreaker **class that calculates the time needed to break a key of a given length of bits, and a** KeyTester **class that tests the** KeyBreaker **class using two key sizes**. Your classes must meet the requirements laid out in the grading rubric.

## Class Design

These are the methods your KeyBreaker must implement. All methods are public unless otherwise noted. *See the Grading Rubric for the class's instance variables*.

|  |  |  |  |
| --- | --- | --- | --- |
| Method name | What it does: | What info it needs: | What info it returns: |
| KeyBreaker | Constructs a KeyBreaker object | * Length of the key to break, in bits * Number of keys that can be broken per second | Nothing |
| getKeyLength | Returns the length of the key to break | Nothing | Key length, in bits |
| setKeyLength | Sets the length of the key to break | Key length, in bits | Nothing |
| countKeys  (private) | Calculates the total number of possible keys based on the key's length | Nothing | Number of possible keys |
| getTimeInHours | Returns the time to break the key, in hours | Nothing | Number of hours needed to break the key |
| getTimeInDays | Returns the time to break the key, in days | Nothing | Number of days needed to break the key |

## Program Output

Your test program should print this as its output. Note the discrepancy between what Java calculated and the expected (real) answer for the 128-bit key…why is that the case?

It takes 3 hours to break 40-bit keys at 100,000,000 keys per second

Expected: 3 hours

It takes 2922 years to break 128-bit keys at 100000000 keys per second

Expected: 1.08 x 10^23 years

## Requirements/Grading Rubric

* YES = Full credit
* Partial = Half credit
* NO = No credit

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Implemented**  **Successfully?** | | |  |  |
| **Requirement** | **YES** | **Partial** | **NO** | **Max Points** | **Comments** |
| ***Program Setup (1.5 points)*** | | | | | |
| Has a package named keys |  |  |  | **0.5** |  |
| Has a class named KeyBreaker |  |  |  | **0.5** |  |
| Has a class named KeyTester with a main() method |  |  |  | **0.5** |  |
| ***KeyBreaker class (7 points)*** | | | | | |
| Has two (2) private instance variables with appropriate data types:   * keyLength: stores the key length * keyRate: stores the number of keys that can be broken per second |  |  |  | **1** |  |
| Has one (1) constructor that initializes the key length and key rate with the given parameter values |  |  |  | **1** |  |
| Implements four (4) public methods and one (1) private method with correct logic, parameters and return types  See **HINTS** on next page for help. |  |  |  | **5** |  |
| ***KeyTester test program class (5 points)*** | | | | | |
| Declares a constant for the number of keys to break per second (100 million) |  |  |  | **0.5** |  |
| Constructs a KeyBreaker object   * key length: 40-bit key * key rate: 100 million |  |  |  | **0.5** |  |
| Calls the getTimeInHours() method to get the time to break a 40-bit key, storing the result |  |  |  | **0.5** |  |
| Prints the 40-bit key output correctly using KeyBreaker's "getter" methods |  |  |  | **1** |  |
| Calls the setKeyLength() method to change your KeyBreaker object's key length to 128 bits |  |  |  | **0.5** |  |
| Calls the getTimeInDays() method to get the time to break a 128-bit key |  |  |  | **0.5** |  |
| Prints the 128-bit key output correctly |  |  |  | **0.5** |  |
| Uses sensible, descriptive variable names |  |  |  | **1** |  |
| ***Documentation (1.5 points)*** | | | | | |
| Javadoc-style code comments for the KeyBreaker class and all public methods, including constructors |  |  |  | **1** |  |
| Code comment in the KeyTester class briefly explaining why the program printed 2922 years as the 128-bit key-breaking time (which is wrong.) |  |  |  | **0.5** |  |

|  |  |
| --- | --- |
| Lightbulb | **HINTS on a few of** KeyBreaker **class's methods**:  countKeys() : a key is a string of bits. To calculate the number of keys, think powers of two! What Math class method helps you out here? Choose your return type carefully…you're going to have a lot of possible keys!  getTimeInHours() : to do this, first call KeyBreaker's own method that counts the number of keys for you. Take the KeyBreaker's keyRate – the number of keys that it can break in one second – and calculate the number of keys it can break in one hour. Finally, calculate the time it will take to go through all keys… a simple speed problem.  getTimeInYears() : again, re-use a method from the KeyBreaker class. Convert the number that comes back from that method to years and return your answer. |