CPSC 3200 Object-Oriented Development

Programming Assignment #1: Due Thursday, April 8, 2021 before MIDNIGHT

For an acceptable P1 submission:

1. use **C#** and Visual Studio
2. upload all files (NOT a project) to Canvas
3. use Unit Testing to verify the functionality of class *c*
4. use Programming by Contract to specify contractual design, placing
    a. class and interface invariants at the top of *dubPrime.cs* file
    b. implementation invariant at the end of *dubPrime.cs* file
    c. pre & postconditions (if needed) before each method header
5. write readable code – see codingStd.docx in the PA folder under Files on Canvas
6. employ the OOP tenets of abstraction, encapsulation and information hiding
7. use functional decomposition => NO monolithic drivers
8. document your driver:
    a. ProgrammingByContract NOT used for drivers
    b. DO NOT assume that the reader has access to this assignment specification
    c. *provide an overview of your program*
    d. *explicitly state ALL assumptions*
9. ***do NOT hard code: replace arbitrary literals, such as '42, with constants***

TWO perspectives supported via P1 fulfillment:

-- the class designer (designs and implements the *dubPrime* class)

-- the client (the software that uses *dubPrime* objects; simulated here by the driver *P1.cs*)

Part I: Class Design (dubPrime.cs)

Each *dubPrime* object encapsulates a positive prime number x, which must be at least 2-digits long, and, acts in either *up* or *down* mode. Upon `query(p)`, a *dubPrime* object returns x + e, if in *up* mode, OR returns x - e, if in *down* mode, where e is the smallest number > p that when added to or subtracted from x yields a prime number. For example, active *dubPrime* **`obj1` that encapsulates 71** and active **`obj2` that encapsulates 89** would yield the following

```
p                obj1.query(p)          obj2.query(p)
3                79                     97    // up mode
3                67                     83    // down mode

9                83                     101   // up mode
9                61                     79    // down mode
```

Every *dubPrime* object is initially in *up* but transitions to *down* after some number of queries (a bound that should vary from object to object). The client may reset as well as revive a *dubPrime* object. Any `query(p)` that yields an invalid number causes that object to be permanently deactivated.

Many details are missing. You MUST make and DOCUMENT your design decisions!!

This assignment is an abstract realization of a data sink (store) that yields specific information upon query but can age and become invalid. With the interface described above, your design should encapsulate and control state as well as the release of information. ***Do NOT tie your type definition to the Console.***

Use Unit Testing to verify your class functionality.

Part II: Driver (P1.cs) -- External Perspective of Client – tests your class design

Design a **FUNCTIONALLY DECOMPOSED** driver to demonstrate the program requirements.
Use many distinct objects, initialized appropriately, i.e. random distribution of dubPrimes, etc.
Adequate testing requires varied (random) input sufficient enough to yield objects in different states and the seamless alteration of the state of some objects.
Make your output readable but not exceedingly lengthy.

The rubric below is ONLY a general sample
NOTE: Regardless of rubric, points will be deducted for non-professional coding styles
Consult codingStd.docx in the coding folder under Files

Class Design (70 points)

Contractual Design	20 points
Interface and Implementation invariants	
Pre & Post conditions	
Proper Accessibility (public, private)	5 points
Appropriate state set in constructor (or default)	5 points
Error design	5 points
Definition and control of state	10 points
Appropriately defined and supported functionality	25 points

Unit Testing 10 points

Driver (20 points)

Appropriate functional decomposition & documentation	10 points
PROGRAMMER name, date, revision history, platform, etc.	
Description of process(es) performed by program	
Explanation of user interface (input, meaning of output)	
Comments on use and validity (error processing)	
Statement of assumptions	
Required functionality verified	5 points
Appropriate allocation and manipulation of objects	5 points