CPSC 3200 Object-Oriented Development

Programming Assignment #5: Due Thursday, June 3, 2022 before MIDNIGHT

P5 exercises your understanding of multiple inheritance and interfaces

For an acceptable P5 submission:

1. Reuse two inheritance hierarchies to define 'cross-product' functionality
2. Use the C# interface construct
3. Fulfill requirements as specified in steps 1-9 from P1

Part I: Class Design

Reuse the *dataExtractor* class hierarchy from P3 -- *dataExtractor*, *dataHalf*, and *dataPlus*, where all types are as previously defined EXCEPT that each *dataExtractor's* minimum length is dependent (in some manner) on the value of the last odd number of its encapsulated sequence.

Define a second class hierarchy of *guard*s, where each *guard* object:

1) Encapsulates its own stable array s of numbers
2) May be in *up* or *down* mode
3) If in *up* mode, and x is a valid integer, *value(x)*
    a. returns the smallest prime in s that is larger than x
4) If in *down* mode, and x is a valid integer, *value(x)*
    a. returns the largest prime in s that is smaller than x

skipGuard is-a *guard* that skips k numbers in s when computing *value(x)* where k is an unstable number that varies from object to object.

quirkyGuard is-a *guard* that replaces numbers arbitrarily, and concatenates a prime number when in *up* mode, and a non-prime number when in *down* mode, upon each *value* query. *quirkyGuard*s can switch modes only a limited number of times (variable across type but stable for an individual object).

Define and implement the classes need to mimic several 'multiply inherited types' and thus reflect the 'cross-product' of both inheritance hierarchies. Composite type functionality must be represented. Sample types would be *dataHalfGuard, dataHalfSkipGuard, dataHalfQuirkyGuard, dataPlusGuard,…*

Many details are missing. You MUST make and DOCUMENT your design decisions!!
Do NOT tie your type definition to the Console.

Use Unit Testing to verify functionality of each new class (no need to retest the classes from P3).

Part II: Driver (P5.cs) -- External Perspective of Client – tests inheritance hierarchy design

The P5 driver must test the use of the 'multiply-inherited' types together.
Thus, it will differ from the unit tests which test each type separately.
Additionally:

1) Use at least one heterogeneous collection for testing functionality
2) Instantiate a variety of objects
3) Trigger a variety of mode changes