CPSC 3200 Object-Oriented Development

Programming Assignment #2: Due Sunday, September 27, 2020 before MIDNIGHT

P2 exercises your understanding of composition, copy semantics, and smart pointers

For an acceptable P2 submission:

1. **use C++ --** the g++ compiler (C++17) on cs1
   a. programs developed in Visual Studio often do NOT compile on cs1
   b. HIGHLY recommended to use C++ tools -- g++, CLion, Xcode etc.
   c. Submissions that do not compile on cs1 will NOT receive credit
2. upload all .h and .cpp files to cs1 AND to Canvas (do not use .hpp)
   use the submission script to upload and compile:
   /home/fac/dingle/submit/20fq3200/p2_runme
3. design using **composition** and **move semantics**
   a. `jumpPrime` should be reused from P1 but must be rewritten in C++
4. **use smart pointers in your driver**
5. Fulfill requirements as specified in steps 4-9 from P1

Part I: Class Design

Each *duelingJP* object encapsulates some number of distinct *jumpPrime* objects; the cardinality of *jumpPrime* subobjects varies across *duelingJP* objects. The key functionality of a *duelingJP* object is to manage the *jumpPrime* subobjects and answer client queries as to the number of 'collisions' and 'inversions'.

Two *jumpPrime* objects are said to 'collide' if they produce the same prime number upon an up (or down) request. For example, if active *jumpPrime* object `j1` encapsulates 2222, and *jumpPrime* object `j2` encapsulates 2233, they both will produce 2221 for `down()` and 2237 for `up()`.

Two *jumpPrime* objects are said to be 'inversions' if they produce the same prime number for complementary requests. For example, if active *jumpPrime* object `j3` encapsulates 3519, and *jumpPrime* object `j4` encapsulates 3528, `j4` produces 3527 for `down()` and `j3` produces 3527 for `up()`.

Deep copying must be supported, and should be explicitly tested in the driver.
Move semantics must be supported, and should be explicitly tested in the driver by the use of smart pointers and an STL container.

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

Part II: Driver (P2.cpp) -- External Perspective of Client – tests your class design

Fulfill the testing requirements as specified in P1. Unit testing is not required or expected.
Additionally:

1) Do NOT use raw pointers
2) Demonstrate your understanding of smart pointers by using both `unique_ptr` and `shared_ptr` instead of raw pointers to reference heap-allocated *`duelingJP`* objects
3) Use a container from the STL
   Add and remove heap-allocated *`duelingJP`* objects to this container
4) Demonstrate copying of *`duelingJP`* objects via call by value