ROMAN NUMBERS CLASS

We now have everything we need to write a complete Roman Numbers data type.

Yes, it looks intimidating, but you have already written most of the code. All that really remains is to do all of the operator overloading, and the only data type you have to worry about is int so that is easy. Many of the operator overloads just require one line of C++ code.

We will accompany this with an InvalidRomanNumberException type, which will be thrown as necessary.

I have given the UML as "private" and "public" blocks so it follows the flow of your C++ code.

```
RomanNumber
// private data and functions

theValue: int                 // the integer value of the RomanNumber
```

```
// public constant and functions

// class constant
// const static means one value is shared by all instances of the class
// in java we would say static final

maxValue: const static int // value of the largest RomanNumber
                           // should be at least 4999, go higher if you can

// public helper functions, all from the earlier Roman Number assignment

bool isValidDecimalNumber(string)    // used by a constructor
bool isValidRomanNumber(string)      // used by a constructor
string convertRomanToDecimal(string) // used by a constructor

// constructors

RomanNumber          // default RomanNumber
                     // value is 1 (one) as we cannot have zero

RomanNumber (int)    // creates a RomanNumber using the given int;
                     // throw an exception if zero or negative
                     // throw an exception if greater than maxValue

RomanNumber(double)  // cast to int and invoke the previous constructor

RomanNumber(string)  // create a RomanNumber based on the given string
                     // that string could be decimal or Roman
                     // throw an exception if invalid
                     // this is where you need those helper functions

RomanNumber(RomanNumber) // copy constructor, just copy theValue
```

```
// type casts

(int)       // casts a RomanNumber to int (just return theValue)
(double)    // casts a RomanNumber to double (again, just theValue)
(string)    // casts a RomanNumber to string (just call toString)

// I/O functions

toString: string  // provides the RomanNumber as a string; e.g. MCMLXXVII
                  // your DecimalToRoman function from the previous
                  // assignment is the code you need here

// since operator<< and operator>> aren't invoked by a RomanNumber,
// we need to give them full access to our class; thus they are "friends"

friend operator<< // output stream, uses the toString code

friend operator>> // input stream, accepts anything
                  // (int, double, string, RomanNumber)
                  // that a RomanNumber constructor can take

// math operations; note that this is all integer arithmetic
// all should throw an exception if result <= 0 or result > maxValue

operator+=        // write these first, then build the next five from them
operator-=
operator*=
operator/=
operator%=

operator+         // build these from the above as in the previous demo
operator-
operator*
operator/
operator%

// relational operators

operator==        // these are all one line functions
operator!=
operator<
operator<=
operator>
operator>=
```

Run the provided RomanNumberDemo code against your class.
DO NOT ALTER THE PROVIDED CODE IN ANY WAY.
Submit your code and a screen shot of the running demo code with all output shown.