NAME: _____

Learning Objectives

- Learn how to implement classes (user defined data types).
- Understand how to implement constructors.
- Understand how to implement and use member functions.
- Understand the concept of encapsulation (hiding internal details to make usage of a class more obvious).
- Learn how to write test code for classes.

Instructions

Work out the answers to these problems manually without the use of a computer. This will help you develop the ability to read and analyze code. It will also provide you with practice solving the type of problems that will appear on Quiz 2 and the final exam. During these exams, you will not have access to a computer to solve these problems, so you need to develop the ability to read code, analyze it and think critically about it. After coming up with answers manually, you can optionally write and run test code to check whether you have solved the problems correctly.

Submit a single document with your answers either by email or through your remote Git repository.

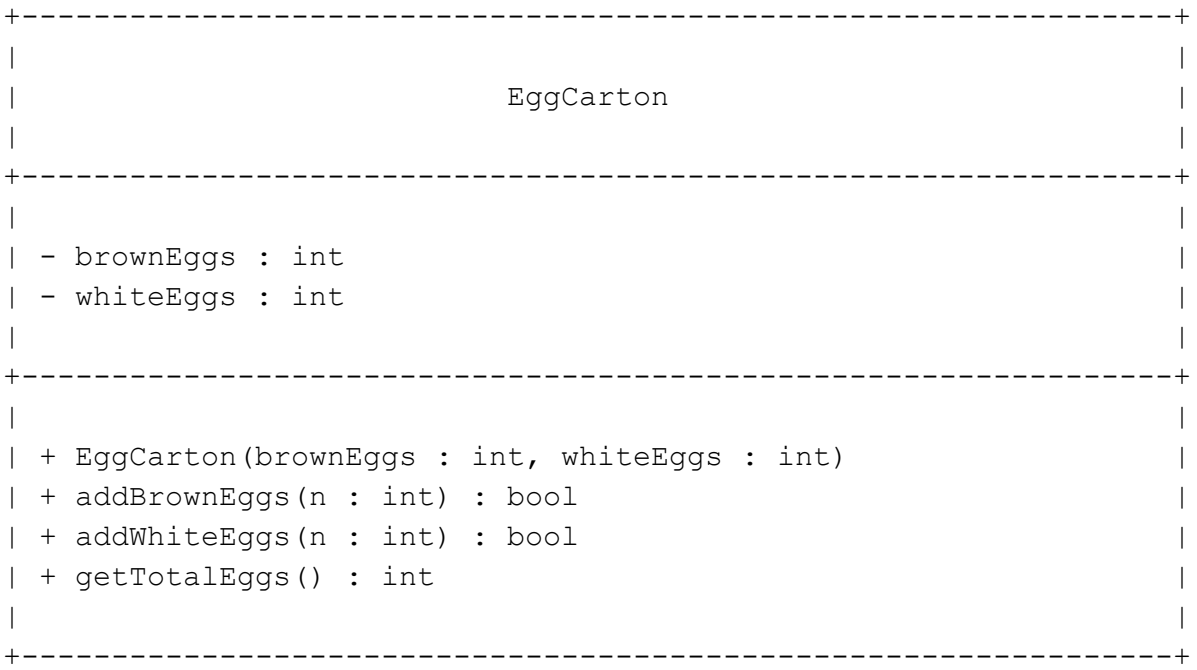

Figure 1

The UML class diagram for the *EggCarton* class is shown in Figure 1. Instances of this class represent egg cartons that can hold up to 12 eggs. The class provides functions to add white and brown eggs. All of the functions that modify the number of eggs in the carton return false if the operation can not be done exactly as requested. For example, if there are 3 white eggs and 0 brown eggs, then *addBrownEggs(10)* returns false and the number of white eggs remains at 3 and brown eggs at 0. There is a member function called *getTotalEggs*, which returns the total number of white and brown eggs together. For example, if the carton has 1 brown and 2 white eggs, then *getTotalEggs* returns 3.

1) Provide an implementation of the constructor for the *EggCarton* class. (5 points)

2) Provide an implementation of the *getTotalEggs* function in the *EggCarton* class. (5 points)

3) Provide an implementation of the *addBrownEggs* function for the *EggCarton* class.
(5 points)

4) Provide an implementation of the *addWhiteEggs* function for the *EggCarton* class. (5 points)

5) Provide test code for the *addBrownEggs* function. Use assert statements for this purpose. Make sure that your test code executes every line of code in the function. (5 points)

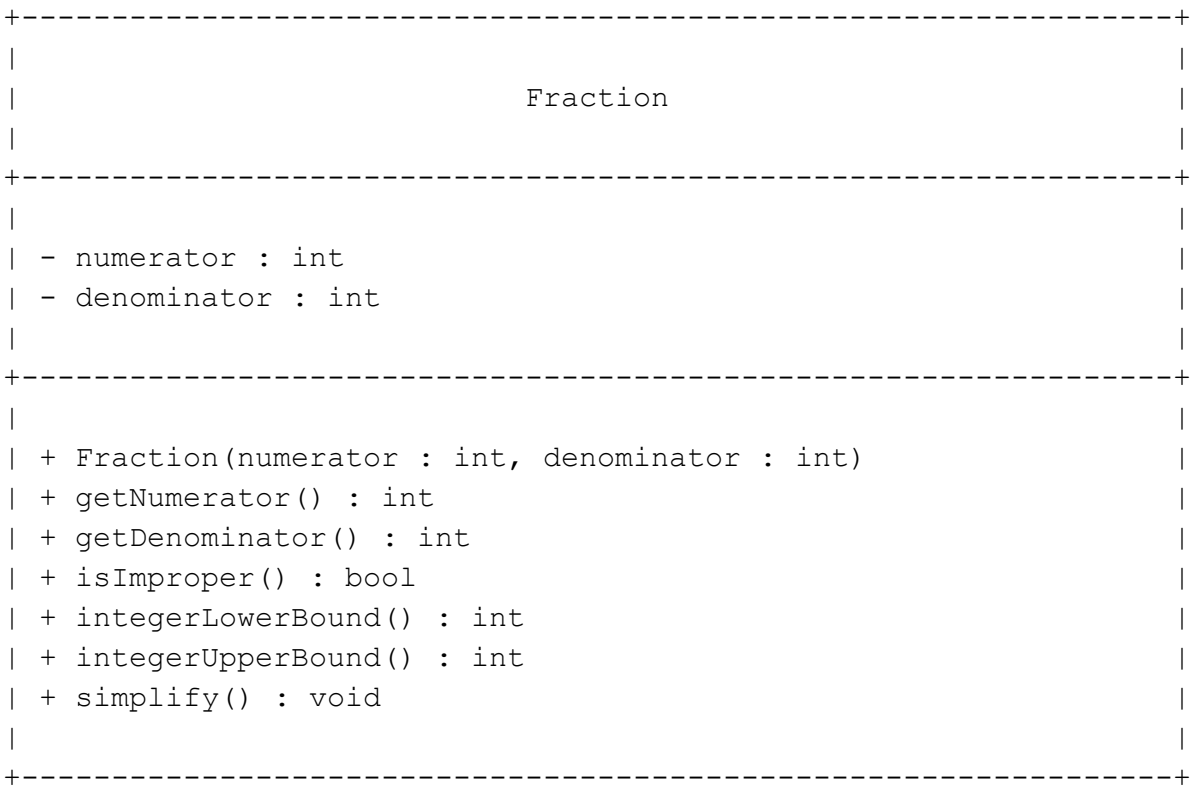

Figure 2

```
class Fraction
{
public:
    Fraction(int numerator, int denominator);
    int getNumerator() const;
    int getDenominator() const;
    bool isImproper() const;
    int integerLowerBound() const;
    int integerUpperBound() const;
    void simplify();

private:
    int numerator;
    int denominator;
};
```

Figure 3

Figure 2 contains the UML class diagram for a class called *Fraction* and Figure 2 contains the code for declaring the class.

Instances of the *Fraction* class represent positive fractions. The function *getNumerator* returns the numerator. The function *getDenominator* returns the denominator. The function *isImproper* returns true if the numerator is larger than the denominator, otherwise it returns false. For example, *isImproper* returns true for 8/6. The function *integerLowerBound* returns the greatest integer that is less than or equal to the fraction. For example, the integer lower bound of 17/5 is 3. The function *integerUpperBound* returns the smallest integer that is greater than or equal to the fraction. For example, the integer lower bound of 17/5 is 4. The function *simplify* reduces the numerator and denominator to their smallest possible values without changing the rational number the fraction represents. For example, 8/6 simplifies to 4/3.

For this problem, assume that the numerator and denominator within instances of the *Fraction* class are positive integers.

6) Provide an implementation of the *Fraction* constructor. (5 points)

7) Provide an implementation of the *isImproper* function. (5 points)

8) Provide an implementation of the *integerLowerBound* function. Hint: rely on integer division. (5 points)

9) Provide an implementation of the *integerUpperBound* function. Hint: use the modulo operator (%). (5 points)

10) Provide test code for the *integerLowerBound* and *integerUpperBound* functions. Use *assert* statements for this purpose. Make sure the test code executes all lines of code in the functions. Use the fraction 8/4 for one of your test cases. (5 points)

11) Provide an implementation of the *simplify* function. (5 points)