Homework 3

Problem 1: RatNum (5 pts)

Answer the following questions, writing your answers in the file answers/problem1.pdf. Two or three sentences should be enough to answer each question. For full credit, your answers should be short and to the point. Points will be deducted for answers that are excessively long or contain irrelevant information.

1. Classify each public method of RatNum as either a creator, observer, producer, or mutator.

public RatNum(int n):	**Creator**
public RatNum(int n, int d):	**Creator**
public boolean isNaN():	**Observer**
public boolean isNegative():	**Observer**
public boolean isPositive():	**Observer**
public int compareTo(RatNum rn):	**Observer**
public double doubleValue():	**Observer**
public int intValue():	**Observer**
public float floatValue():	**Observer**
public long longValue():	**Observer**
public RatNum negate(): producer	**Producer**
public RatNum add(RatNum arg):	**Producer**
public RatNum sub(RatNum arg):	**Producer**
public RatNum mul(RatNum arg):	**Producer**
public RatNum div(RatNum arg):	**Producer**
public int hashCode():	**Observer**
public boolean equals(/*@Nullable*/ Object obj):	**Observer**
public String toString():	**Observer**
public static RatNum valueOf(String ratStr):	**Producer**

2. add, sub, mul, and div all require that arg != null. This is because all of these methods access fields of arg without checking if arg is null first. But these methods also access fields of this without checking for null; why is this != null absent from the requires clause for these methods?

This represents a RatNum object. When a RatNum object is initialized, a value must be assigned to this object. This means we are safe to build our methods without having to check if this != null.

3. Why is RatNum.valueOf(String) a class method (has static modifier)? What alternative to class methods would allow someone to accomplish the same goal of generating a RatNum from an input String?

Valueof is created as a static modifier. This means this method only modifies the parameters given, which would be the string. It does not modify "this," only the parameters given.

Since the Valueof method does not need access to the variables within the class, the method could be changed to a helper function outside of the RatNum class.

4. add, sub, mul, and div all end with a statement of the form return new RatNum (numerExpr, denomExpr);. Imagine an implementation of the same function except the last statement is:

```
this.numer = numerExpr;
this.denom = denomExpr;
return this;
```

For this question, pretend that the this.numer and this.denom fields are not declared as final so that these assignments compile properly. How would the above changes fail to meet the specifications of the function (hint: take a look at the @requires and @modifies clauses, or lack thereof) and fail to meet the specifications of the RatNum class?

Add, sub, mul, and div are all producer methods. They all lack @modify statements, so they should not be able to alter class variables via "this." Not only this but, the class as a whole is immutable, so we are unable to change object variables.

5. Calls to checkRep() are supposed to catch violations in the classes' invariants. In general, it is recommended to call checkRep() at the beginning and end of every method. In the case of RatNum, why is it sufficient to call checkRep() only at the end of constructors? (Hint: could a method ever modify a RatNum such that it violates its representation invariant? Could a method change a RatNum at all? How are changes to instances of RatNum prevented?)

From the start, RatNum as a class is Immutable. Once initialized with the constructor, the RatNum object is unable to be changed. This means we don't need checkRep to check elements of RatNum with the exception being the constructor itself.