CSCI 2600 — Principles of Software
Homework 3: Problem 1 Answers

Jordan Ye

March 13, 2025

Problem 1: RatNum

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

Creators:

- `RatNum(int n)`
- `RatNum(int n, int d)`
- `valueOf(String ratStr)`

Observers:

- `isNaN()`
- `isNegative()`
- `isPositive()`
- `compareTo(RatNum rn)`
- `doubleValue()`
- `intValue()`
- `floatValue()`
- `longValue()`
- `hashCode()`
- `equals(Object obj)`
- `toString()`

Producers:

- `negate()`
- `add(RatNum arg)`
- `sub(RatNum arg)`
- `mul(RatNum arg)`
- `div(RatNum arg)`

Mutators:

- None (RatNum is immutable)

(2) **Why is `this != null` absent from the requires clause?**

The requires clause doesn't need to state that `this != null` because it's impossible for "this" to be null in a method invocation. When a method is called on an object, the object reference must exist for the method to be called in the first place. In contrast, "arg" is a parameter passed to the method, which could be null if not properly checked.

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

`RatNum.valueOf(String)` is a class method (static) because it's a factory method used to create new RatNum instances from a String representation. This allows for constructing RatNum objects without using the constructor directly. An alternative would be to use an instance method in a separate factory class (like `RatNumFactory`) that would handle the creation of RatNum instances from strings.

(4) **How would the alternative implementation fail to meet specifications?**

The alternative implementation would fail to meet specifications in several ways:

(a) It would modify the state of the existing RatNum object (`this`), violating the immutability requirement specified in the class documentation.

(b) It would violate the `@modifies` clause (or lack thereof) as these methods don't specify they modify anything.

(c) It would produce incorrect behavior when operations are chained (e.g., `a.add(b).mul(c)`) since the first operation would modify 'a' directly rather than returning a new object.

(d) It breaks the representation invariant since it would not run `checkRep()` on the modified object.

(5) **Why is it sufficient to call `checkRep()` only at the end of constructors?**

It's sufficient to call `checkRep()` only at the end of constructors because RatNum is an immutable class. The fields are declared as `final`, which means they cannot be modified after the object is constructed. Since no method can modify a RatNum instance after creation, the representation invariant, once established by the constructor, cannot be violated. This makes additional `checkRep()` calls in other methods unnecessary, as the object's state never changes after construction.