Problem 1

(1) Creator: `RatNum(int n)`, `RatNum(int n, int d)`

Observer: `isNaN`, `isNegative`, `isPositive`, `compareTo`, `doubleValue`, `intValue`, `floatValue`, `longValue`, `hashCode`, `equals`, `toString`

Producer: `negate`, `add`, `sub`, `mul`, `div`, `valueOf`

Mutator: none of the methods are mutators

(2) `this != null` doesn't need to be included in the `@requires` clause because `null` can't call any methods; so it's a given that if `this == null`, any method called by it will result in a `NullPointerException`.

(3) `RatNum.valueOf(String)` is a class method because the method acts as a helper method to convert `String`s into `RatNum`s. This doesn't rely on an instance of `RatNum`, making it suitable to be a class method. An alternative to using a class method would be to make another constructor that takes a `String` as an input and creates a `RatNum` from that.

(4) In the specs of `add`, `sub`, `mul`, and `div`, since `@modifies` isn't ever specified, nothing about `this` or the passed in arguments should change. But, `this.numer` and `this.denom` get modified, violating the `@modifies` clause.

(5) Calling `checkRep()` at the beginning and end of every method isn't necessary. Since the fields of `RatNum` are declared to be `final`, they can't be modified. So, if the fields don't cause `checkRep()` to throw an error at the end of a constructor, then the fields will never violate the representation invariant.