1.

a.	RatNum(int n)	creator
b.	RatNum(int n, int d)	creator
c.	isNaN()	observer
d.	isNegative()	observer
e.	isPositive()	observer
f.	compareTo(RatNum rn)	observer
g.	doubleValue()	observer
h.	intValue()	observer
i.	floatValue()	observer
j.	longValue()	observer
k.	negate()	producer
l.	add(RatNum arg)	producer
m.	sub(RatNum arg)	producer
n.	mul(RatNum arg)	producer
o.	div(RatNum arg)	producer
p.	hashCode()	observer
q.	equals(/*@Nullable*/ Object obj)	observer
r.	toString()	observer
s.	valueOf(String ratStr)	producer

2. If an object reference equals null, it is not referring to any object. Being able to use a method that is part of an object's implementation directly implies that the current object (this) exists, so checking for "this != null" is unnecessary.
3.
    a. RatNum.valueOf(String) is a static method because it exists independently of any instance of their class; it does not reference any instances of RatNum and only creates instances via "new" and RatNum constructors.
    b. A RatNum constructor method with one argument of the String type would also allow one to generate a RatNum from an input string. Instead of creating a RatNum instance with "new" keyword, the private variables (numer, denom) can be directly accessed to create the instance; this is typical constructor behavior.

4. These changes mean the current object is modified ("modifies: this"), but none of the functions have @modifies statements, which implies "modifies: none." In terms of the RatNum class, these changes are allowed if RatNum is a mutable datatype, but the class's overview states that RatNum is an immutable datatype.
5. Because RatNum is an immutable datatype, no method can modify a RatNum in any way, so checks against its representation invariant are only needed after executing the code involved in its creation. Thus, checkRep is only called at the end of constructors. Also, there is no chance of representation exposure since its functions only return new instances of RatNum or primitives.