1.

a. creator
   public RatNum(int n)
   public RatNum(int n, int d)
   public static RatNum valueOf(String ratStr)
   Observers
   public boolean isNaN()
   public boolean isNegative()
   public boolean isPositive()
   public int compareTo(RatNum rn)
   public double doubleValue()
   public int intValue()
   public float floatValue()
   public long longValue()
   public int hashCode()
   public boolean equals(Object obj)
   public String toString()
   Producers
   public RatNum negate()
   public RatNum add(RatNum arg)
   Public RatNum sub(RatNum arg)
   public RatNum mul(RatNum arg)
   public RatNum div(RatNum arg)
   mutator

b. this != null is absent because this can't be a null in a method. If an instance is happening this is already a valid object meaning it can't be null. Java also already makes sure that instance methods only call valid objects.

c. RatNum.valueOf(String ratStr) is static because it doesn't use an existing RatNum instance (this). Instead, it takes a String parameter to create a new RatNum object. The alternative would be to directly use a constructor like public RatNum(String ratStr), which would allow object creation from a string without needing a separate static method.

d. @return a new RatNum means the method must create and return a new object, not modify the existing one. Since there is no @modifies clause, changing this would break the expected behavior. Making RatNum mutable would also corrupt constants like NaN and ZERO, leading to unintended consequences.

e. "Because methods cannot modify RatNum, it is sufficient to check the representation invariant only at the end of constructors. Since RatNum has no mutators, its state never changes after creation. This means the only time a RatNum object could violate its representation invariant is during its creation, so checkRep() only needs to be called at the end of the constructor.