```java
import java.math.BigInteger;

/**
 * Represents an arbitrary-precision rational number (fraction) with a numerator and
denominator.
 * Provides functionality for basic arithmetic operations, comparison, and formatting.
 */
public class Rational {

    private final BigInteger numerator;
    private final BigInteger denominator;

    /**
     * Constructs a Rational number with the specified numerator and denominator.
     * The fraction is automatically simplified to its lowest terms.
     *
     * @param numerator the numerator of the rational number
     * @param denominator the denominator of the rational number; must not be zero
     * @throws IllegalArgumentException if the denominator is zero
     */
    public Rational(BigInteger numerator, BigInteger denominator) throws
IllegalArgumentException {
        if (denominator.equals(BigInteger.ZERO)) {
            throw new IllegalArgumentException("Denominator cannot be zero");
        }
        BigInteger gcd = numerator.gcd(denominator);
        this.numerator = numerator.divide(gcd);
        this.denominator = denominator.divide(gcd);
    }

    /**
     * Constructs a Rational number representing an integer value.
     *
     * @param numerator the integer value as a rational number (denominator = 1)
     */
    public Rational(BigInteger numerator) {
        this(numerator, BigInteger.ONE);
    }

    /**
     * Adds this rational number to another rational number and returns the result as
a new Rational object.
     *
     * @param other the Rational number to add
     * @return a new Rational representing the sum of this and the other rational
number
     * @throws NullPointerException if the other Rational is null
     */
```

```java
    public Rational add(Rational other) throws NullPointerException {
        return null;
    }

    /**
     * Subtracts another rational number from this rational number and returns the
result as a new Rational object.
     *
     * @param other the Rational number to subtract
     * @return a new Rational representing the difference between this and the other
rational number
     * @throws NullPointerException if the other Rational is null
     */
    public Rational subtract(Rational other) throws NullPointerException {
        // Subtracts another Rational from this and returns the result
        return null;
    }

    /**
     * Multiplies this rational number by another rational number and returns the
result as a new Rational object.
     *
     * @param other the Rational number to multiply by
     * @return a new Rational representing the product of this and the other rational
number
     * @throws NullPointerException if the other Rational is null
     */
    public Rational multiply(Rational other) throws NullPointerException {
        // Multiplies this Rational by another and returns the result
        return null;
    }

    /**
     * Divides this rational number by another rational number and returns the result
as a new Rational object.
     *
     * @param other the Rational number to divide by
     * @return a new Rational representing the quotient of this and the other rational
number
     * @throws NullPointerException if the other Rational is null
     * @throws ArithmeticException if the other Rational is zero (division by zero)
     */
    public Rational divide(Rational other) throws NullPointerException,
ArithmeticException {
        // Divides this Rational by another and returns the result
        return null;
    }
```

```java
    /**
     * Compares this rational number to another rational number.
     *
     * @param other the Rational number to compare with
     * @return a negative integer if this is less than the other, zero if equal, or a
positive integer if greater
     * @throws NullPointerException if the other Rational is null
     */
    public int compareTo(Rational other) throws NullPointerException {
        // Compares two Rational numbers
        return 0;
    }

    /**
     * Checks if this rational number is equal to another object.
     * Two Rational numbers are considered equal if they have the same numerator and
denominator
     * in their reduced form. Since the constructor automatically reduces the
fraction,
     * equality can be checked directly by comparing the numerator and denominator.
     *
     * @param obj the object to compare with
     * @return true if the other object is a Rational with the same value as this,
false otherwise
     */
    @Override
    public boolean equals(Object obj) {
        // Checks equality between this Rational and another object
        return false;
    }

    /**
     * Returns a hash code for this rational number.
     * The hash code is computed based on both the numerator and denominator in their
reduced form.
     * This ensures that two Rational numbers with the same value (and thus equal by
equals())
     * will have the same hash code.
     *
     * @return an integer hash code for this rational number
     */
    @Override
    public int hashCode() {
        // Provides a hash code for the Rational
        return 0;
    }

    /**
```

```
     * Converts this rational number to a string in the format "numerator/denominator"
or "numerator" if the denominator is 1.
     *
     * @return a string representation of this rational number
     */
    @Override
    public String toString() {
        // Converts the Rational to a string
        return "";
    }

    /**
     * Returns the numerator of this rational number.
     *
     * @return the numerator as a BigInteger
     */
    public BigInteger getNumerator() {
        // Returns the numerator
        return null;
    }

    /**
     * Returns the denominator of this rational number.
     *
     * @return the denominator as a BigInteger
     */
    public BigInteger getDenominator() {
        // Returns the denominator
        return null;
    }

    /**
     * Returns the decimal value of this rational number as a double, with possible
precision loss.
     *
     * @return the decimal approximation of this rational number
     */
    public double toDouble() {
        // Converts the Rational to a double approximation
        return 0.0;
    }

}
```