Fraction Class Notes

**1. Boolean Precondition (positive)**

* Purpose: The `positive` boolean field is used to track the sign (positive or negative) of the fraction. This helps in performing operations like multiplication or addition correctly by considering the sign of the fraction.
* Where it's used: The `positive` field is passed into the constructor as `isPositive` and initialized as part of the Fraction object. This ensures that the sign is set when a new fraction is created.

Constructor with Precondition:

public Fraction(boolean isPositive, int numerator, int denominator) {  
 positive = isPositive;  
 num = numerator;  
 den = denominator;  
 simplify();  
 }

* This constructor takes in a boolean `isPositive`, which indicates whether the fraction is positive or negative.
* Precondition: The comment specifies that the numerator and denominator are expected to be positive. The boolean `isPositive` indicates whether the entire fraction is positive or negative.
* This is a common way to handle fractions since it avoids storing negative values for the numerator or denominator and instead uses the boolean field for sign management.

In Methods:

* The `positive` boolean field is checked in methods like `toString()`, `asDecimal()`, and `asMixedNumber()` to ensure the correct sign is applied when outputting the fraction in different formats.

For example, in the `toString()` method:

public String toString() {  
 String out = "";  
 if(!positive) { // if it's a negative value print "-"  
 out = "-";  
 }  
 return out + num + "/" + den;  
 }

If the fraction is negative (i.e., `positive` is false), a `-` sign is added to the output.

**2. Passing the Fraction Class as an Argument**

In Java, objects like instances of the `Fraction` class can be passed as arguments to methods, allowing for operations between different objects of the same type.

* Where it's used: The `Fraction` class uses this concept in methods like `times()`, `plus()`, and `isGreaterThan()`. In these methods, a `Fraction` object is passed as an argument, and the method operates on both the calling object and the passed object.

Example: `times()` Method:

public Fraction times(Fraction otherFraction) {  
 boolean pos = positive == otherFraction.positive; // result is positive if both have the same sign  
 int numerator = num \* otherFraction.num;  
 int denominator = den \* otherFraction.den;  
 return new Fraction(pos, numerator, denominator);  
 }

Explanation:

* This method takes another `Fraction` object, `otherFraction`, as a parameter.  
   - It multiplies the numerators and denominators of both fractions.  
   - The result’s sign (`pos`) is determined by comparing the `positive` fields of both fractions. If both have the same sign, the result is positive.
* A new `Fraction` object is returned with the calculated numerator, denominator, and sign.

Example: `plus()` Method:

public Fraction plus(Fraction otherFraction) {  
 int num1 = num \* otherFraction.den;  
 if(!positive) {  
 num1 \*= -1;  
 }  
 int num2 = otherFraction.num \* den;  
 if(!otherFraction.positive) {  
 num2 \*= -1;  
 }  
 int numerator = num1 + num2;  
 int denominator = den \* otherFraction.den;  
 if(numerator >= 0) {  
 return new Fraction(true, numerator, denominator);  
 }  
 return new Fraction(false, -numerator, denominator);  
 }

Explanation:

* This method adds the current fraction to another fraction passed in as `otherFraction`.  
   - It cross-multiplies the numerators and denominators to bring both fractions to a common denominator before adding.  
   - The signs of the fractions are taken into account by checking the `positive` fields and adjusting the numerators accordingly.

**Summary**

1. Boolean Precondition:  
- The `positive` boolean is a field used to manage the sign of the fraction without directly storing negative values in the numerator or denominator.  
- It is passed into the constructor and handled in various methods like `toString()`, `asDecimal()`, and `asMixedNumber()` to format the output correctly.

2. Passing Objects as Arguments:  
- The `Fraction` class demonstrates how to pass objects (instances of `Fraction`) to methods.  
- Methods like `times()` and `plus()` take another `Fraction` object as an argument and perform operations between the calling object and the passed object.

This approach encapsulates fraction operations and makes the class reusable and modular.