Chapter 6—Objects and Classes

Rational Numbers

- As a more elaborate example of class definition, section 6.4 defines a class called `Rational` that represents rational numbers, which are simply the quotient of two integers.

- Rational numbers can be useful in cases in which you need exact calculation with fractions. Even if you use a `double`, the floating-point number 0.1 is represented internally as an approximation. The rational number 1 / 10 is exact.

- Rational numbers support the standard arithmetic operations:

  \[
  \begin{align*}
  \frac{a}{b} + \frac{c}{d} & = \frac{ad + bc}{bd} \\
  \frac{a}{b} - \frac{c}{d} & = \frac{ad - bc}{bd} \\
  \frac{a}{b} \times \frac{c}{d} & = \frac{ac}{bd} \\
  \frac{a}{b} \div \frac{c}{d} & = \frac{ad}{bc}
  \end{align*}
  \]
Implementing the `Rational` Class

- The next five slides show the code for the `Rational` class along with some brief annotations.

- As you read through the code, the following features are worth special attention:
  - The constructors for the class are overloaded. Calling the constructor with no argument creates a `Rational` initialized to 0, calling it with one argument creates a `Rational` equal to that integer, and calling it with two arguments creates a fraction.
  - The constructor makes sure that the numerator and denominator of any `Rational` are always reduced to lowest terms. Moreover, since these values never change once a new `Rational` is created, this property will remain in force.
  - The `add`, `subtract`, `multiply`, and `divide` methods are written so that one of the operands is the receiver (signified by the keyword `this`) and the other is passed as an argument. Thus to add `r1` and `r2` you would write:

```java
r1.add(r2)
```

```java
/**
 * The Rational class is used to represent rational numbers, which
 * are defined to be the quotient of two integers.
 */
public class Rational {

private int num; /* The numerator of this Rational */
private int den; /* The denominator of this Rational */

/** Creates a new Rational initialized to zero. */
public Rational() {
    this(0); // Calls the constructor that takes a single int argument.
}

/**
 * Creates a new Rational from the integer argument.
 * @param n The initial value
 */
public Rational(int n) {
    this(n, 1); // Calls the constructor that takes two int arguments.
}
}
```
The Rational Class

```java
/**
 * Creates a new Rational with the value x / y.
 * @param x The numerator of the rational number
 * @param y The denominator of the rational number
 */
public Rational(int x, int y) {
    int g = gcd(Math.abs(x), Math.abs(y));
    num = x / g;
    den = Math.abs(y) / g;
    if (y < 0) num = -num;
}
```

The primary constructor creates a new Rational from the numerator and denominator. The call to gcd ensures that the fraction is reduced to lowest terms.

```java
/**
 * Adds the rational number r to this one and returns the sum.
 * @param r The rational number to be added
 * @return The sum of the current number and r
 */
public Rational add(Rational r) {
    return new Rational(this.num * r.den + r.num * this.den,
                         this.den * r.den);
}
```

The add method creates a new Rational object using the addition rule. The two rational values appear in this and r.

```java
/**
 * Subtracts the rational number r from this one.
 * @param r The rational number to be subtracted
 * @return The result of subtracting r from the current number
 */
public Rational subtract(Rational r) {
    return new Rational(this.num * r.den - r.num * this.den,
                         this.den * r.den);
}
```

These methods (along with divide on the next page) work just like the add method but use a different formula. Note that these methods do have access to the components of r.

```java
/**
 * Multiplies this number by the rational number r.
 * @param r The rational number used as a multiplier
 * @return The result of multiplying the current number by r
 */
public Rational multiply(Rational r) {
    return new Rational(this.num * r.num, this.den * r.den);
}
```
The Rational Class

/**
 * Divides this number by the rational number r.
 * @param r The rational number used as a divisor
 * @return The result of dividing the current number by r
 */
public Rational divide(Rational r) {
    return new Rational(this.num * r.den, this.den * r.num);
}

/**
 * Calculates the greatest common divisor using Euclid’s algorithm.
 * @param x First integer
 * @param y Second integer
 * @return The greatest common divisor of x and y
 */
private int gcd(int x, int y) {
    int r = x % y;
    while (r != 0) {
        x = y;
        y = r;
        r = x % y;
    }
    return y;
}
Simulating Rational Calculation

- The next slide works through all the steps in the calculation of a simple program that adds three rational numbers.

\[
\frac{1}{2} + \frac{1}{3} + \frac{1}{6}
\]

- With rational arithmetic, the computation is exact. If you write this same program using variables of type `double`, the result looks like this:

![Rational values as abstract objects. Chapter 7 reprises the example showing the memory structure.](image)

- The simulation treats the `Rational` values as abstract objects. Chapter 7 reprises the example showing the memory structure.

Adding Three Rational Values

```java
public static void main(String[] args) {
    Rational a = new Rational(1, 2);
    Rational b = new Rational(1, 3);
    Rational c = new Rational(1, 6);
    Rational sum = a.add(b).add(c);
    System.out.println(a + " + " + b + " + " + c + " = " + sum);
}
```

![TestRational](image)