Classes (Part 2)

SE 206

Variable scoping

- A variable is visible within the block it is declared in
  - Called the "scope" of the variable

```
class Scoping {
        int z

        public static void foo (int x) {
                // ...
        }

        public static void bar () {
                // ...
        }

        public static void main (String[] args) {
                int y;
                // ...
        }
}
```

This instance variable is visible anywhere in the Scoping class

This parameter is visible only in the foo() method

This local variable is visible until the end of the main() method

Variable initialization

- A local variable is NOT initialized to a default value
  - This is any variable declared within a method
    - Or within a block within a method

- Parameters are initialized to whatever value they are passed

- Instance and class variables are initialized to default values
  - Numbers to zero, booleans to false, references to null
  - This means any field in a class
    - Either class variables or instance variables

Rational class

What we've seen so far

- An example of creating a class
  - Car

- Up next: another example
  - Rational
    - Represents rational numbers
    - A rational number is any number that can be expressed as a fraction
    - Both the numerator and denominator must be integers!

What properties should our Rational class have?

- The numerator (top part of the fraction)
- The denominator (bottom part of the fraction)
- Not much else…

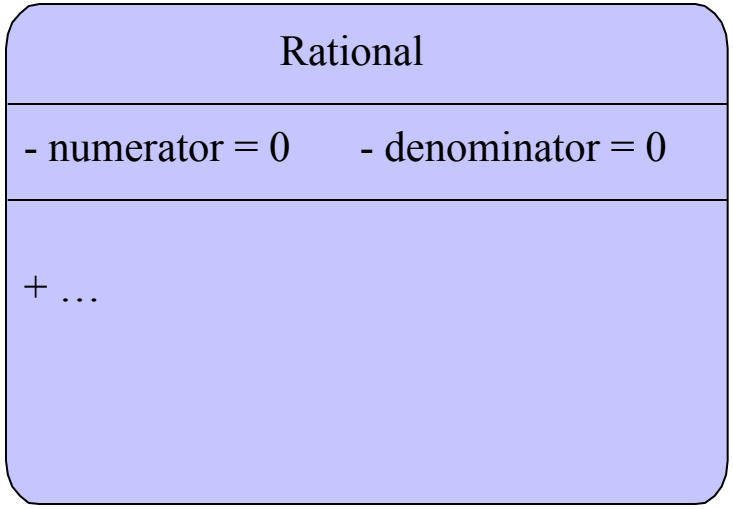
What do we want our Rational class to do?

- Obviously, the ability to create new Rational objects
- Setting the numerator and denominator
- Getting the values of the numerator and denominator
- Perform basic operations with rational numbers: + - * /
- Ability to print to the screen

Our first take at our Rational class

- Our first take

```
class Rational {
      private int numerator;
      private int denominator;
   //...
}
```



- This does not represent a valid Rational number!
  - Why not?

- Java initializes instance variables to zero
  - Both the numerator and denominator are thus set to zero
  - 0/0 is not a valid number!

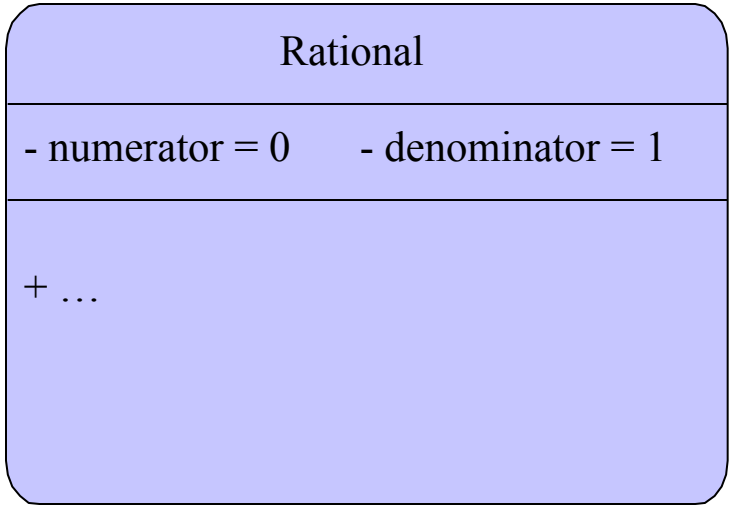
Our next take at our Rational class

- Our next take

```
public class Rational {
      private int numerator = 0;
      private int denominator = 1;
   //...
}
```

- We've defined the attributes of our class

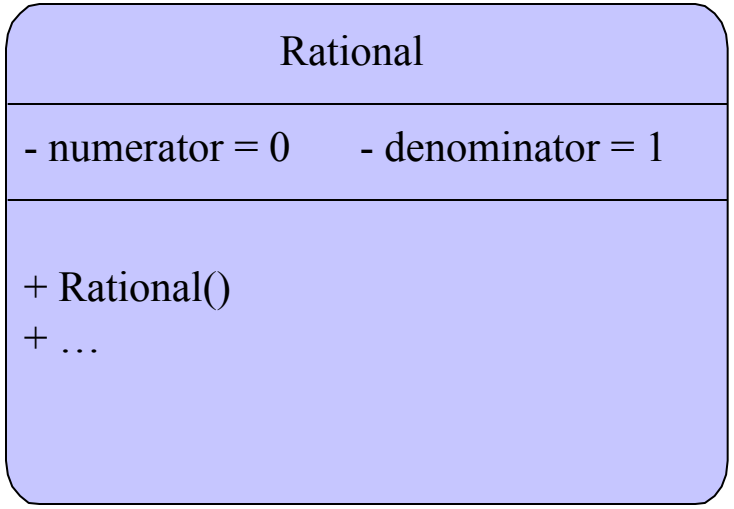
- Next up: the behaviors



The default constructor

- Ready?

```
public Rational() {
}
```



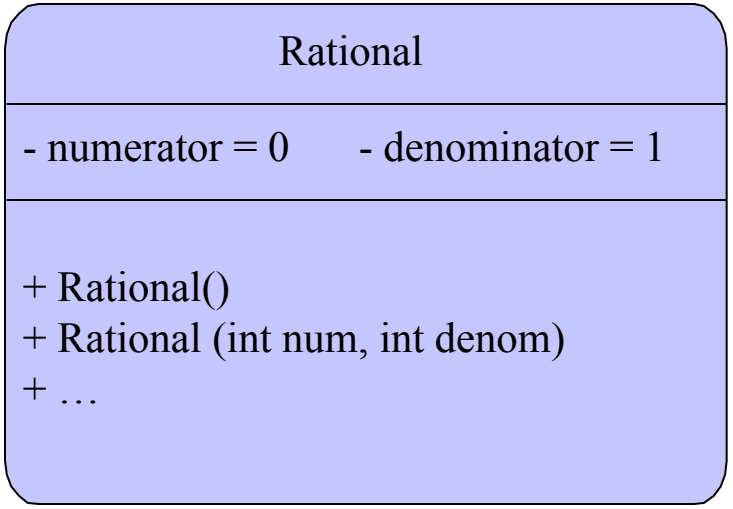
- Note that we could have initialized the instance variables here instead
- The default constructor is called that because, if you don't specify ANY constructors, then Java includes one by default
- Default constructors do not take parameters

The specific constructor

- Called the specific constructor because it is one that the user specifies
  - They take one or more parameters

```
public Rational (int num, int denom) {
        setNumerator (num);
        setDenominator (denom);
}
```

- Note that the specific constructor calls the mutator methods instead of setting the instance variables directly

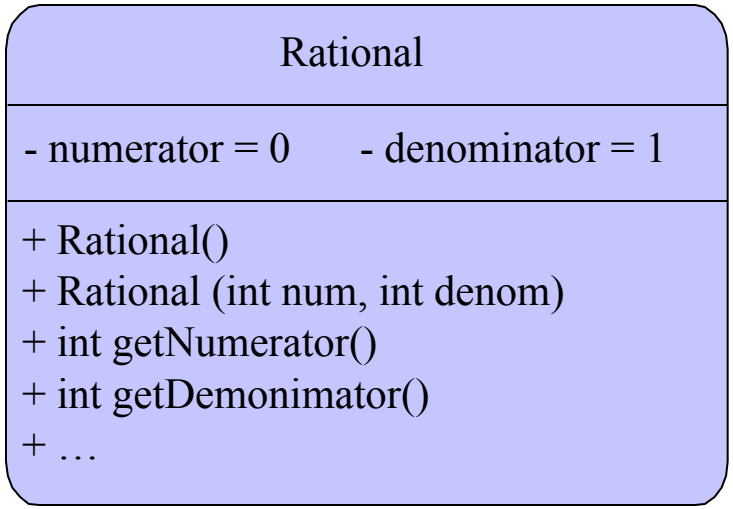


Accessor methods

- Our two accessor methods:

```
public int getNumerator () {
   return numerator;
}

public int getDenominator () {
   return denominator;
}
```



Mutator methods

- Our two mutator methods:

```
public void setNumerator (int towhat) {
   numerator = towhat;
}

public void setDenominator (int towhat) {
   denominator = towhat;
}
```

Rational addition

- How to do Rational addition: $\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}$

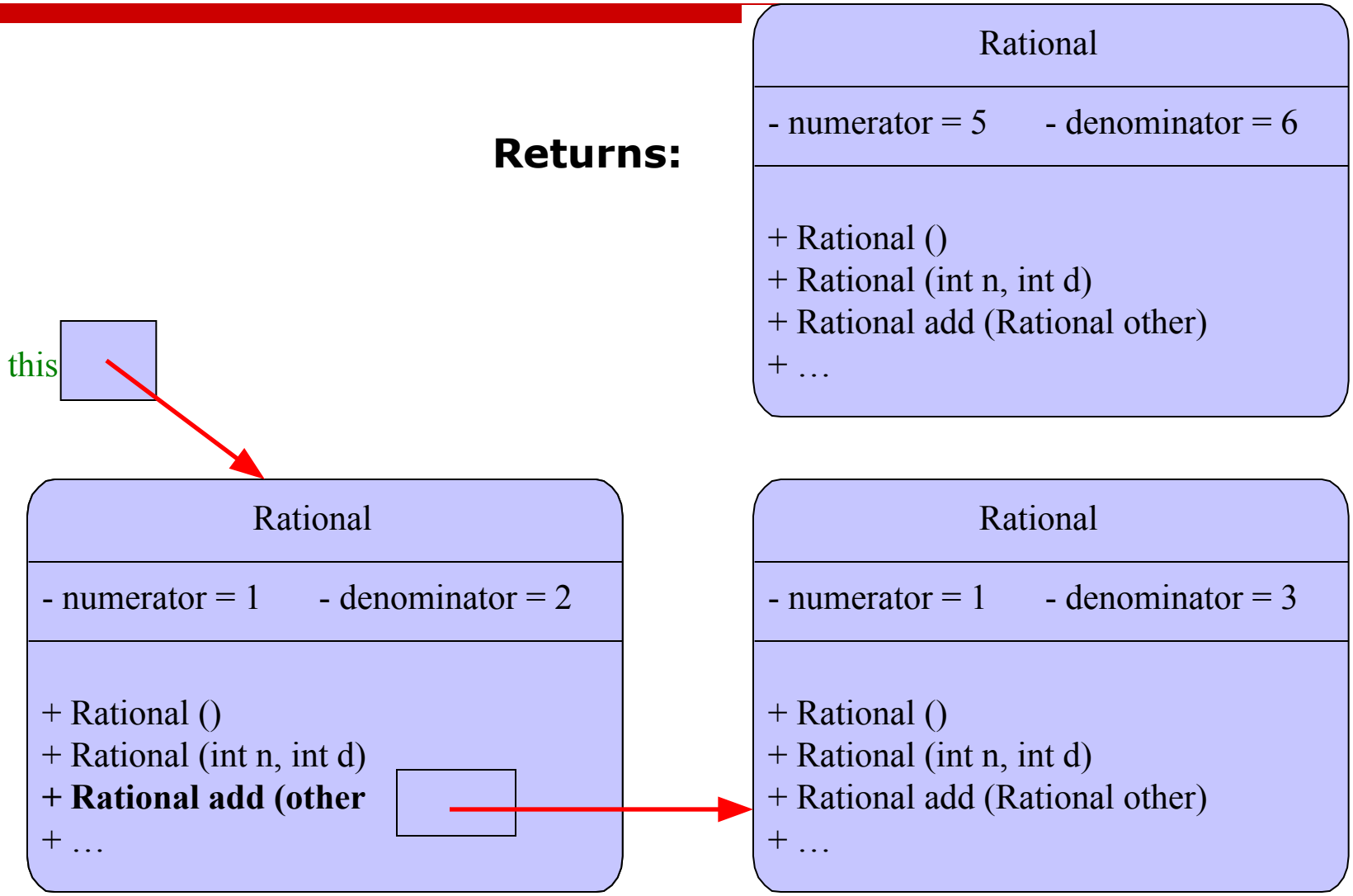
- Our add() method:

```
public Rational add (Rational other) {



}
```

The this keyword



The this keyword

- this is a reference to whatever object we are currently in
- Will not work in static methods
  - We'll see why later
  - Note that the main() method is a static method
- While we're at it, when defining a class, note that NONE of the methods so far were static

Rational addition

- How to do Rational addition:

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}$$

- Our add() method:

```
public Rational add (Rational other) {
   int a = this.getNumerator();
   int b = this.getDenominator();
   int c = other.getNumerator();
   int d = other.getDenominator();
   return new Rational (a*d+b*c, b*d);
}
```

Rational addition

- The following method is equivalent:

- Our add() method:

```
public Rational add (Rational other) {
   int a = getNumerator();
   int b = getDenominator();
   int c = other.getNumerator();
   int d = other.getDenominator();
   return new Rational (a*d+b*c, b*d);
}
```

Rational addition

- The following method is equivalent, but not preferred:

- Our add() method:

```
public Rational add (Rational other) {
   int a = numerator;
   int b = denominator;
   int c = other.numerator;
   int d = other.nenominator;
   return new Rational (a*d+b*c, b*d);
}
```

Rational addition

- The following method is equivalent, but not preferred:

- Our add() method:

```
public Rational add (Rational other) {
   int a = this.numerator;
   int b = this.denominator;
   int c = other.numerator;
   int d = other.nenominator;
   return new Rational (a*d+b*c, b*d);
}
```

Rational subtraction

- How to do Rational subtraction:

$$\frac{a}{b} - \frac{c}{d} = \frac{ad - bc}{bd}$$

- Our subtract() method:

```
public Rational subtract (Rational other) {
   int a = this.getNumerator();
   int b = this.getDenominator();
   int c = other.getNumerator();
   int d = other.getDenominator();
   return new Rational (a*d-b*c, b*d);
}
```

Rational multiplication

- How to do Rational multiplication:

$$\frac{a}{b} * \frac{c}{d} = \frac{ac}{bd}$$

- Our multiply() method:

```
public Rational multiply (Rational other) {
   int a = this.getNumerator();
   int b = this.getDenominator();
   int c = other.getNumerator();
   int d = other.getDenominator();
   return new Rational (a*c, b*d);
}
```

Rational division

- How to do Rational division:

$$\frac{a}{b} \div \frac{c}{d} = \frac{ad}{bc}$$

- Our divide() method:

```
public Rational divide (Rational other) {
   int a = this.getNumerator();
   int b = this.getDenominator();
   int c = other.getNumerator();
   int d = other.getDenominator();
   return new Rational (a*d, b*c);
}
```

Printing it to the screen

- If we try printing a Rational object to the screen:

```
Rational r = new Rational (1,2);
System.out.println (r);
```

- We get the following:

```
Rational@82ba41
```

- Ideally, we'd like something more informative printed to the screen

- The question is: how does Java know how to print a custom class to the screen?

The toString() method

- When an object is put into a print statement:

```
Rational r = new Rational (1,2);
System.out.println (r);
```

- Java will try to call the toString() method to covert the object to a String
  - If the toString() method is not found, a default one is included
    - Hence the Rational@82ba41 from the previous slide

- So let's include our own toString() method

The toString() method

- Our toString() method is defined as follows:

```
public String toString () {
    return getNumerator() + "/" + getDenominator();
}
```

- Note that the prototype must ALWAYS be defined as shown
  - The prototype is the 'public String toString()'

Printing it to the screen

- Now, when we try printing a Rational object to the screen:

```
Rational r = new Rational (1,2);
System.out.println (r);
```

- We get the following:

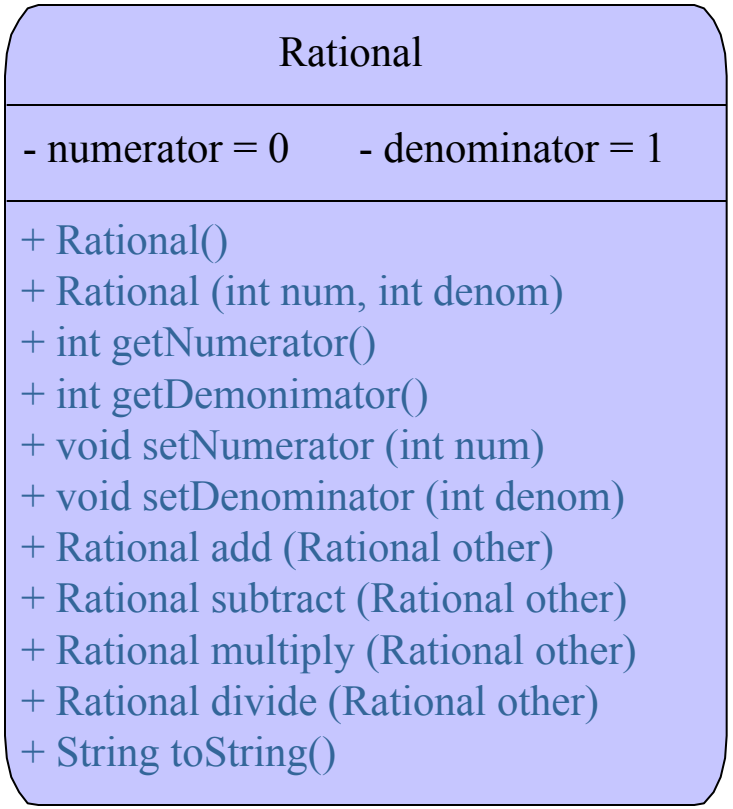
```
1/2
```

- Which is what we wanted!

- Note that the following two lines are (mostly) equivalent:

```
System.out.println (r);
System.out.println (r.toString());
```

Our full Rational class



Our Rational class in use, part 1 of 4

- This code is in a main() method of a RationalDemo class
- First, we extract the values for our first Rational object:

```
Scanner stdin = new Scanner(System.in);

System.out.println();

// extract values for rationals r and s
Rational r = new Rational();
System.out.print("Enter numerator of a rational number: ");
int a = stdin.nextInt();
System.out.print("Enter denominator of a rational number: ");
int b = stdin.nextInt();
r.setNumerator(a);
r.setDenominator(b);
```

Our Rational class in use, part 2 of 4

- Next, we extract the values for our second Rational object:

```
Rational s = new Rational();
System.out.print("Enter numerator of a rational number: ");
int c = stdin.nextInt();
System.out.print("Enter denominator of a rational number: “);
int d = stdin.nextInt();
s.setNumerator(c);
s.setDenominator(d);
```

- Notice that I didn’t create another Scanner object!
  - Doing so would be bad
  - I used the same one

Our Rational class in use, part 3 of 4

- Next, we do the arithmetic:

```
// operate on r and s
Rational sum = r.add(s);
Rational difference = r.subtract(s);
Rational product = r.multiply(s);
Rational quotient = r.divide(s);
```

Our Rational class in use, part 4 of 4

- Lastly, we print the results

```
// display operation results
System.out.println("For r = " + r.toString() + " and s = "
                        + s.toString());
System.out.println("    r + s = " + sum.toString());
System.out.println("    r - s = " + difference.toString());
System.out.println("    r * s = " + product.toString());
System.out.println("    r / s = " + quotient.toString());
System.out.println();
```

Other things we might want to add to our Rational class

- The ability to reduce the fraction
  - So that 2/4 becomes 1/2
  - Not as easy as it sounds!
- More complicated arithmetic
  - Such as exponents, etc.
- Invert
  - Switches the numerator and denominator
- Negate
  - Changes the rational number into its (additive) negation
- We won't see any of that here

Topics that are covered:

- Variable Scoping
- Passing Objects as a parameter
- Return Objects
- ***this***

Reading:

- Java2: The Complete Reference (Herbert Schildt)
  - Chapter 6: Introducing Classes
  - Chapter 7: A Closer Look at Methods and Classes