8. Defining Your Own Classes Part2

1 Oct 2015

Objectives

- Describe how objects are returned from methods.
- Define overloaded methods and constructors.
- Describe the uses of the reserved word this.
- Define class methods and variables.

Objectives

- Describe how the arguments are passed to the parameters in method definitions with the pass-by-value scheme.
- Organize classes into a package.

Returning an Object from a Method

- In this section, we learn how to return objects from methods. We use the Fraction class to illustrate the returning of an object from a method.

```
class Fraction {

   private int numerator;

   private int denominator;

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

Returning an Object from a Method

```
public int getDenominator( ) {

    return denominator;
}

public int getNumerator( ) {

    return numerator;
}

public void setDenominator(int denom) {
    if (denom == 0) {
        //Fatal error
        System.err.println("Fatal Error");
        System.exit(1);
    }
    denominator = denom;
}

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

Returning an Object from a Method

```
public int gcd(int m, int n) {
    //it doesn't matter which of n and m is bigger
    //this method will work fine either way

    //assume m,n >= 1

    int r = n % m;

    while (r !=0) {

        n = m;

        m = r;

        r = n % m;
    }

    return m;
}
```

Returning an Object from a Method

```
public Fraction simplify( ) {

    int num   = getNumerator();
    int denom = getDenominator();
    int gcd   = gcd(num, denom);

    Fraction simp = new Fraction(num/gcd, denom/gcd);

    return simp;
}

public String toString( ) {

     return getNumerator() + "/" + getDenominator();
}
```

Returning an Object from a Method

- In main() method

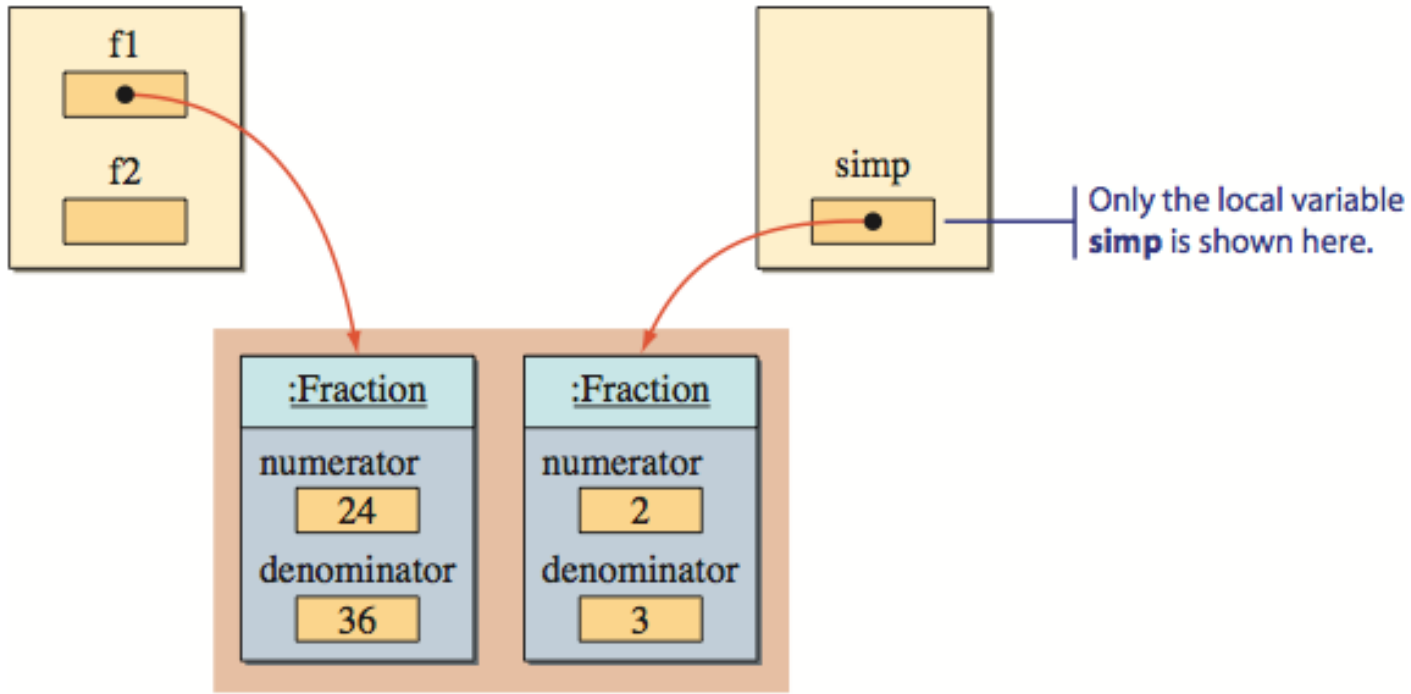
```
Fraction f1, f2;

f1 = new Fraction(24, 36);

f2 = f1.simplify( );

System.out.println(f1.toString() + "can be reduced to " +
                   f2.toString());
```

```
24/36 can be reduced to 2/3
```

Returning an Object from a Method

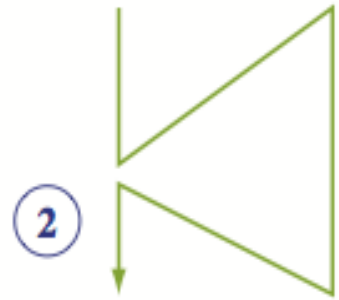
```
f2 = f1.simplify();
```

```
public Fraction simplify() {

   Fraction simp;
   ...
   simp = new Fraction(...);  (1)

   return simp;
}
```

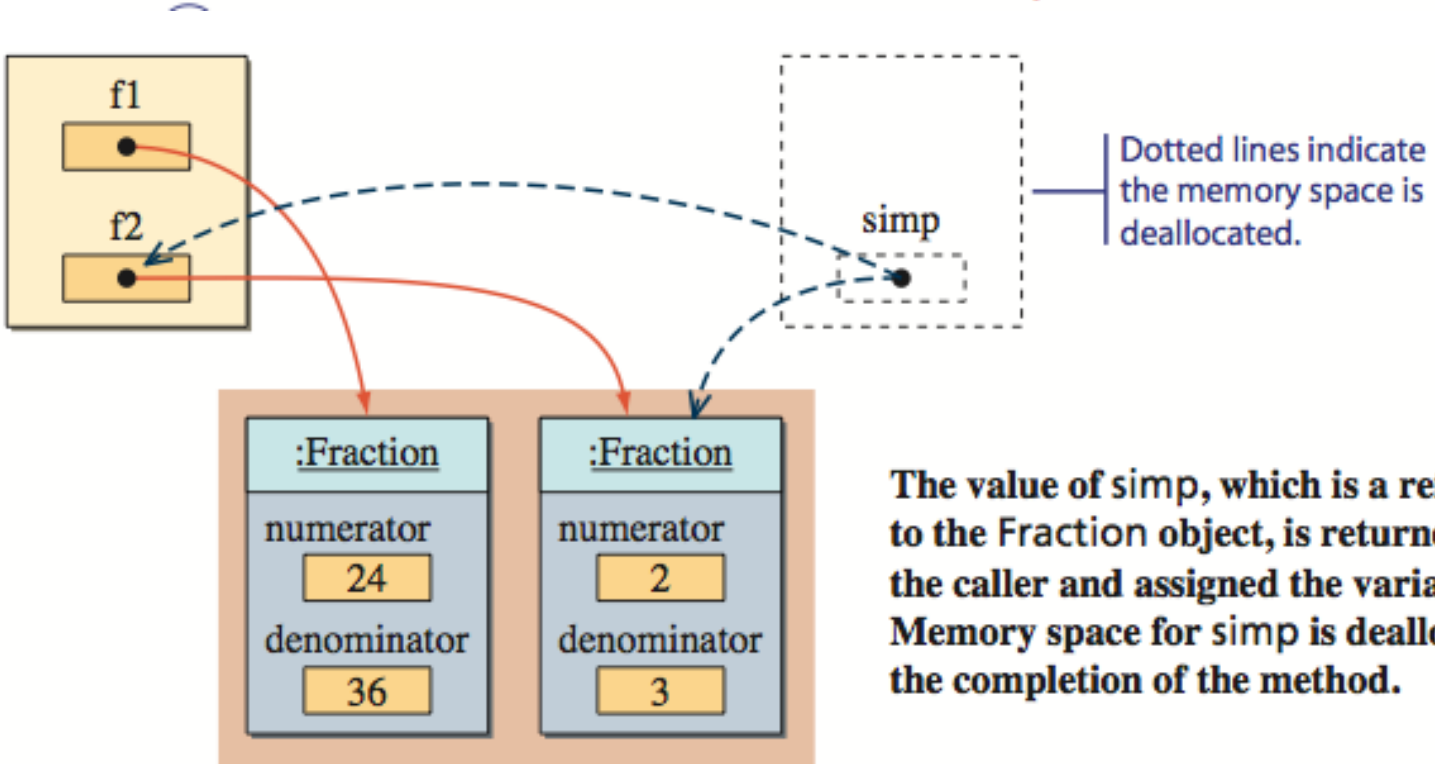


Returning an Object from a Method

```
f2 = f1.simplify();
```




```
public Fraction simplify() {

  Fraction simp;
  ...
  simp = new Fraction(...);

  return simp;
}
```



The value of simp, which is a reference to the Fraction object, is returned --> to the caller and assigned the variable f2. Memory space for simp is deallocated upon the completion of the method.

The Reserved Word ‘this’

- The reserved word ‘this’ is called a ‘self-referencing pointer’ because it is used to refer to the receiving object of a message from within this object’s method.

The Reserved Word ‘this’

- Ex. We now consider the four arithmetic operations for Fraction class.

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

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

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

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

The Reserved Word 'this'

– add() method (Inside Fraction class)

```
public Fraction add( Fraction frac) {

   int a, b, c, d;

   Fraction sum;

   a = this.getNumerator();     //get the receiving
   b = this.getDenominator();   //object's num and denom

   c = frac.getNumerator();    //get frac's num
   d = frac.getDenominator(); //and denom

   sum = new Fraction(a*d + b*c, b*d);

   return sum;

}
```

The Reserved Word ‘this’

– main() method

```
Fraction f1, f2, f3;

f1 = new Fraction(1, 2);
f2 = new Fraction(1, 4);

f3 = f1.add(f2);

System.out.println("Sum of " + f1.toString() + " and " +
                                f2.toString() + " is " +
                                f3.toString();
```

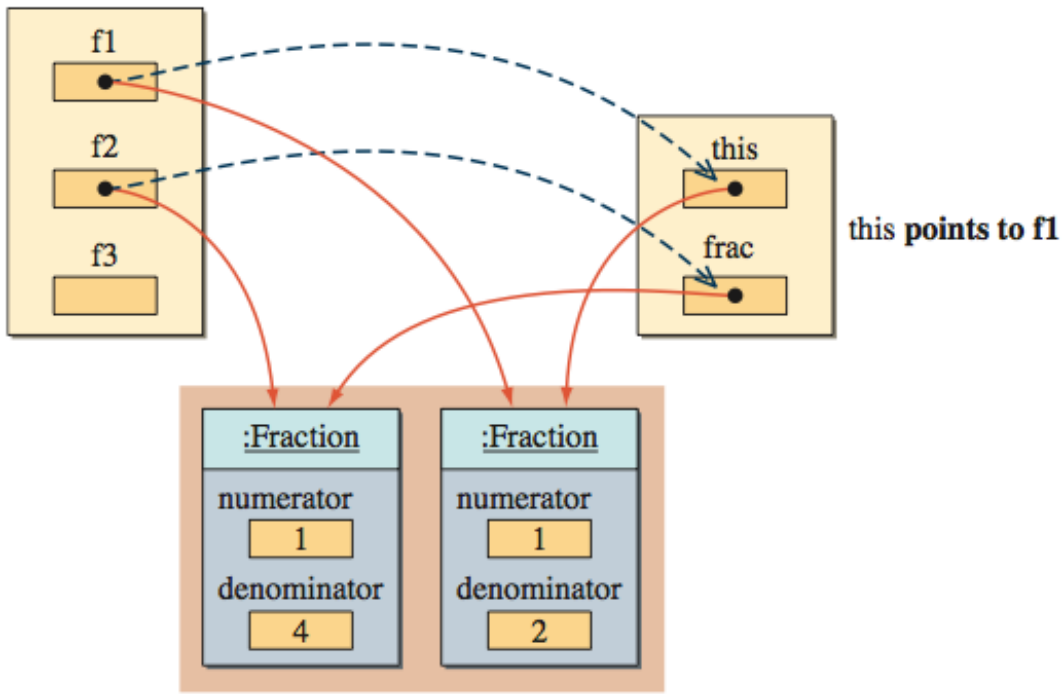
```
Sum of 1/2 and 1/4 is 6/8
```

The Reserved Word ‘this’

– State of memory for: f3 = f1.add(f2);

```
f3 = f1.add(f2);
```

```
public Fraction add(Fraction frac) {

  Fraction sum;
  ...
  sum = new Fraction(...);

  return sum;
}
```

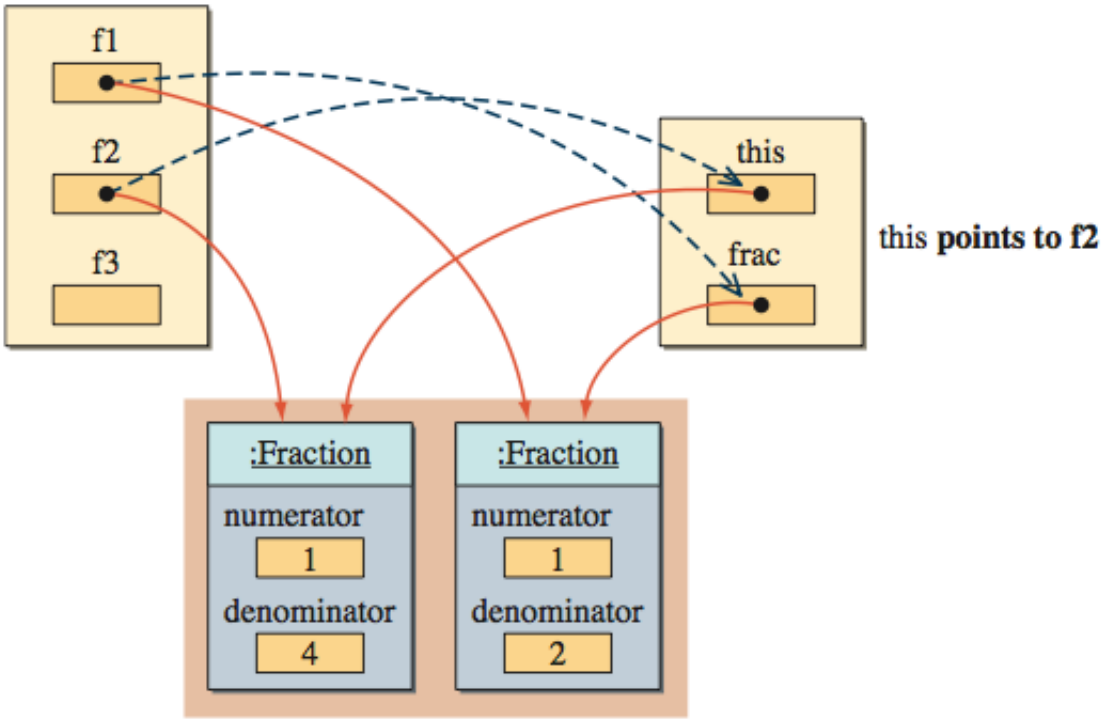


The Reserved Word ‘this’

– State of memory for: f3 = f2.add(f1);

```
f3 = f2.add(f1);
```

```
public Fraction add(Fraction frac) {

  Fraction sum;
  ...
  sum = new Fraction(...);

  return sum;
}
```



The Reserved Word ‘this’

- The use of the reserved word ‘this’ for method called is actually optional.

```
class Sample {

   public void m1( ) {
      ...
   }

   public void m2( ) {
      m1();
   }
}
```

```
class Sample [

   public void m1( ) {
      ...
   }

   public void m2( ) {

      this.m1();
   }
}
```

The reserved word **this** is added by the compiler.

The Reserved Word ‘this’

- Another use of ‘this’ to point to instance data member

```
class MusicCD {

   private String   artist;
   private String   title;

   private String   id;


   public MusicCD(String artist, String title) {

      this.artist = artist;
      this.title  = title;
      id      = artist.substring(0,2) + "-" +
                title.substring(0,9);
   }
```

This refers to the parameter.

Overloaded Methods and Constructors

- Method overloading is a programming concept that allows programmers to define two or more methods with the same name.

- Multiple methods can share the same name as long as one of the following rules is met:
  1. They have a different number of parameters.
  2. The parameters are of different data types when the number of parameters is the same.

Overloaded Methods and Constructors

- Ex. Method overloading with different number of parameters

```
public void myMethod(int x, int y) { ... }
public void myMethod(int x) { ... }
```

Overloaded Methods and Constructors

- Ex. Method overloading with different data types of parameters
  - add() method with Fraction object parameter

```
public Fraction add(Fraction frac) {

   //same as before

}
```

Overloaded Methods and Constructors

– add() method with int parameter

```
public Fraction add(int number) {

   Fraction sum;
   int a, b, c, d;

   a = getNumerator();
   b = getDenominator();
   c = number;
   d = 1;

   sum = new Fraction(a*d + c*b, b*d);
   return sum;
}
```

Including **d** here is redundant because its value is **1**. We include it here anyway for the sake of clarity.

Overloaded Methods and Constructors

- Overloading Constructors: We can define multiple constructors in a programmer-defined class. The same rules for overloaded methods apply.

Overloaded Methods and Constructors

- Ex. From Fraction class, we have first constructor

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

Overloaded Methods and Constructors

– We can create overloading constructors like these

```
public Fraction( ) { //creates 0/1
    setNumerator(0);
    setDenominator(1);
}

public Fraction(int number) { //creates number/1
    setNumerator(number);
    setDenominator(1);
}

public Fraction(Fraction frac) { //copy constructor
    setNumerator(frac.getNumerator());
    setDenominator(frac.getDenominator());
}
```

Overloaded Methods and Constructors

- We can use 'this' to call a constructor from another constructor of the same class. Here's how we can rewrite the last 3 constructors of the Fraction class by using the reserved word 'this':
  - This constructor is called by the other three constructors

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

Overloaded Methods and Constructors

– The last 3 constructors can rewrite by using ‘this’

```
public Fraction( ) { //creates 0/1
    this(0, 1);
}

public Fraction(int number) { //creates number/1
    this(number, 1);
}

public Fraction(Fraction frac) { //copy constructor
    this( frac.getNumerator(),
          frac.getDenominator() );
}
```

Class Variables and Methods

- We introduced the concepts of class methods (static method), class variables (static variables), and class constants in previous chapter.
- The sample of class method is min() method in Math class.

```
int i, j, smaller;

i = ...;
j = ...;

smaller = Math.min(i, j);
```

Class Variables and Methods

- We can also create min() method as class method in Fraction class

```
public static Fraction min(Fraction f1, Fraction f2) {
   //convert to decimals and then compare
   double f1_dec = f1.decimal();
   double f2_dec = f2.decimal();
   if ( f1_dec <= f2_dec) {
      return f1;
   } else {
      return f2;
   }
}
private double decimal( ) {
   //returns the decimal equivalent
   return (double) getNumerator() / getDenominator();
}
```

Class Variables and Methods

– We can called min() method using class name like this

```
Fraction f1, f2, smaller;

f1 = new Fraction(1, 6);
f2 = new Fraction(4, 5);

smaller = Fraction.min(f1, f2);
```

Class Variables and Methods

- Like min() method, we can also create add() method as class method in Fraction class

```
public static Fraction add(Fraction f1, Fraction f2) {
   int a, b, c, d;
   Fraction sum;
   a = f1.getNumerator();
   b = f1.getDenominator();
   c = f2.getNumerator();
   d = f2.getDenominator();
   sum = new Fraction(a*d + b*c, b*d);
   return sum;
}
```

Class Variables and Methods

– Now we can called add() method using class name like this

```
Fraction x = new Fraction(1, 8);
Fraction y = new Fraction(4, 9);

Fraction sum = Fraction.add(x, y);
```

```
Fraction sum = Fraction.add(Fraction.add(x,y), z);
```

Class Variables and Methods

- Now let's look at an example of class variables. Suppose we want to assign a tag number automatically when a new instance of the Bicycle class is created. We want the tag numbers to be ABC-101, ABC-102, ABC-103, and so forth.

Class Variables and Methods

```
class Bicycle {
   private static int counter = 101;
   // Data Members
   private String id;
   ...
   public Bicycle( ) {
      id = "ABC-" + counter;
      ...
      counter++;
   }
```

Call-by-Value Parameter Passing

- When a method is called, the value of the argument is copied and passed to the matching parameter. This way of passing the value of arguments is called a ‘pass-by-value’ or ‘call-by-value’ scheme.
- Pass-by-value (also known as call-by-value) is the only parameter passing mechanism Java supports.

Call-by-Value Parameter Passing

- When passing primitive data types to method
  - Tester class

```
class Tester {

   public void myMethod(int one, double two ) {

      one = 25;
      two = 35.4;
   }
}
```

Call-by-Value Parameter Passing

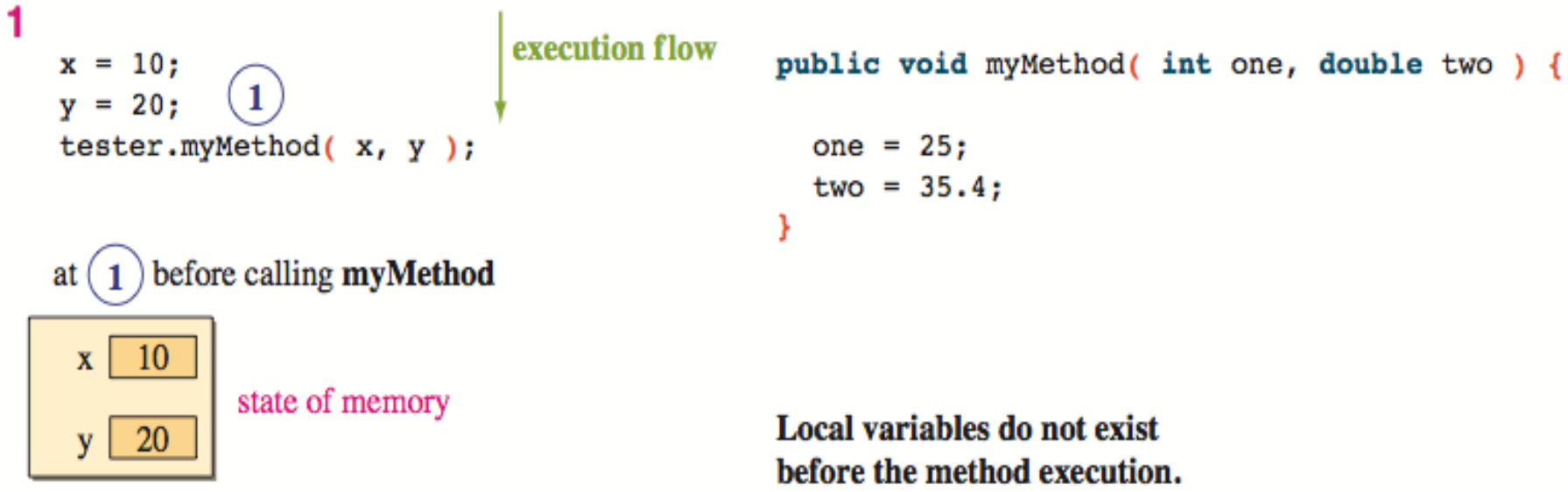
– main() method

```
Tester tester;
int x, y;

tester = new Tester();
x = 10;
y = 20;

tester.myMethod( x, y );

System.out.println( x + "    " + y );
```

```
10    20
```

Call-by-Value Parameter Passing

– How memory space for the parameters is allocated when passing primitive data type.

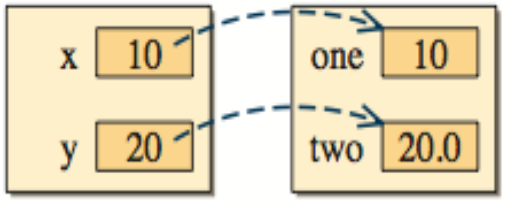


Call-by-Value Parameter Passing

2

```
x = 10;
y = 20;
tester.myMethod( x, y );
```

values are copied at (2)




```
public void myMethod( int one, double two ) {   (2)

  one = 25;
  two = 35.4;
}
```

Memory space for myMethod is allocated, and the values of arguments are copied to the parameters.

Call-by-Value Parameter Passing

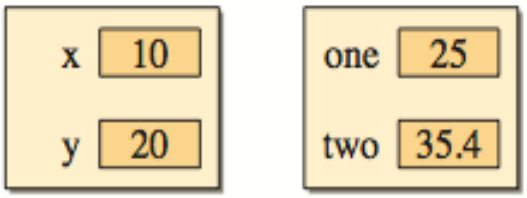
3

```
x = 10;
y = 20;
tester.myMethod( x, y );
```

```
public void myMethod( int one, double two ) {

   one = 25;
   two = 35.4;
}
```

3

at (3) before return



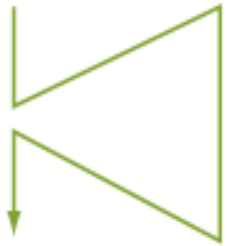
The values of parameters are changed.

Call-by-Value Parameter Passing

4

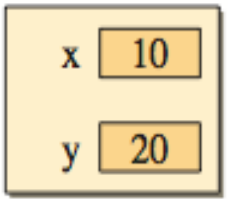
```
x = 10;
y = 20;
tester.myMethod( x, y );
```

4



```
public void myMethod( int one, double two ) {

   one = 25;
   two = 35.4;
}
```

at 4 after **myMethod**



Memory space for myMethod is deallocated, and parameters are erased. Arguments are unchanged.

Call-by-Value Parameter Passing

- When passing object to method
  - ObjectTester class

```
class ObjectTester {

   public void swap(Fraction f1, Fraction f2) {

      Fraction temp;

      temp = f1; //swap the two fractions
      f1   = f2;
      f2   = temp;
   }
}
```

Call-by-Value Parameter Passing

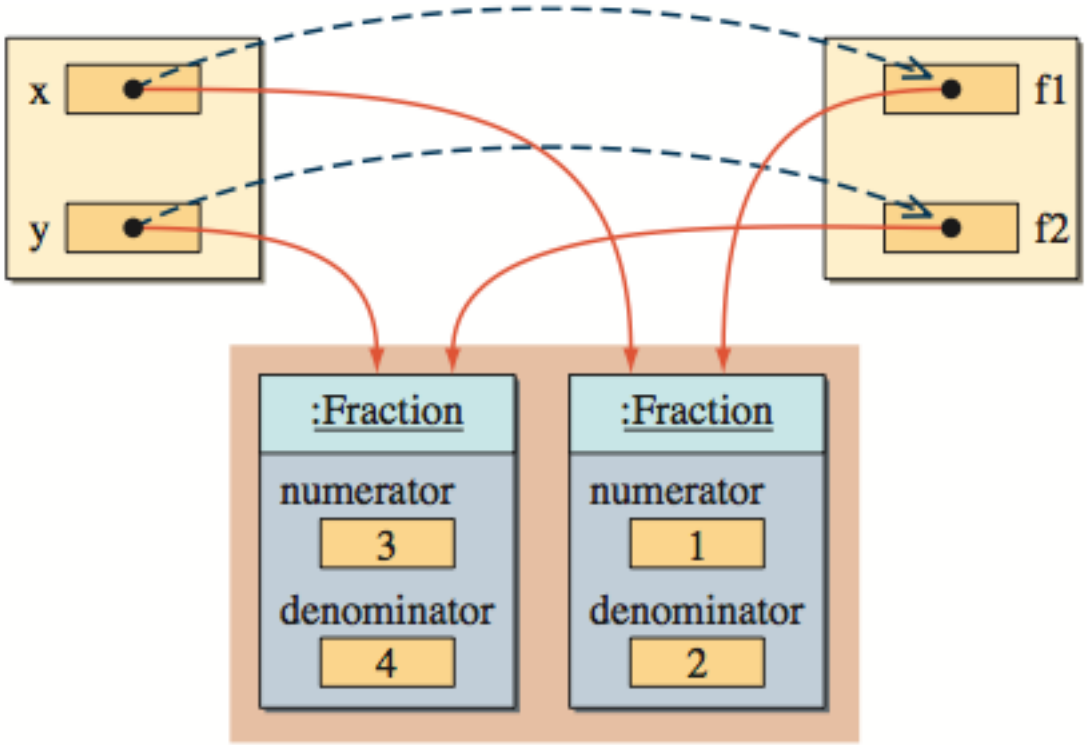
– main() method

```
ObjectTester tester;
Fraction     x, y;

tester = new ObjectTester();

x = new Fraction(1, 2);

y = new Fraction(3, 4);

tester.swap(x, y);

System.out.println("x = " + x.toString());
System.out.println("y = " + y.toString());
```

```
x = 1/2
y = 3/4
```

Call-by-Value Parameter Passing

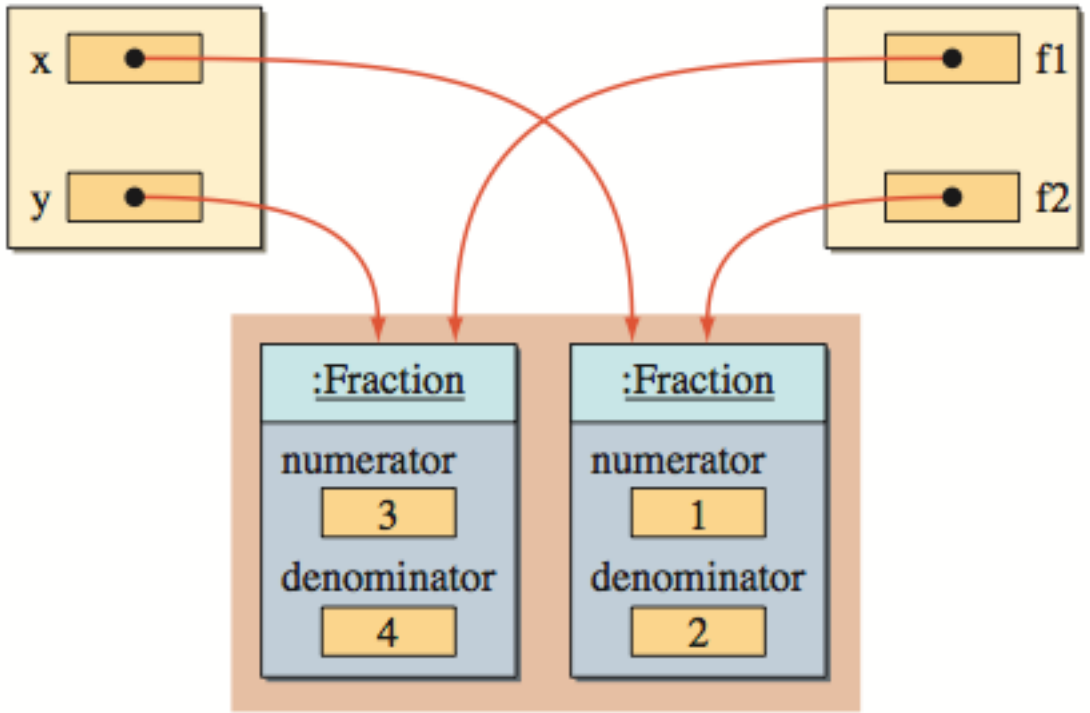
– How memory space for the parameters is allocated when passing object.



At the beginning of the **swap** method.

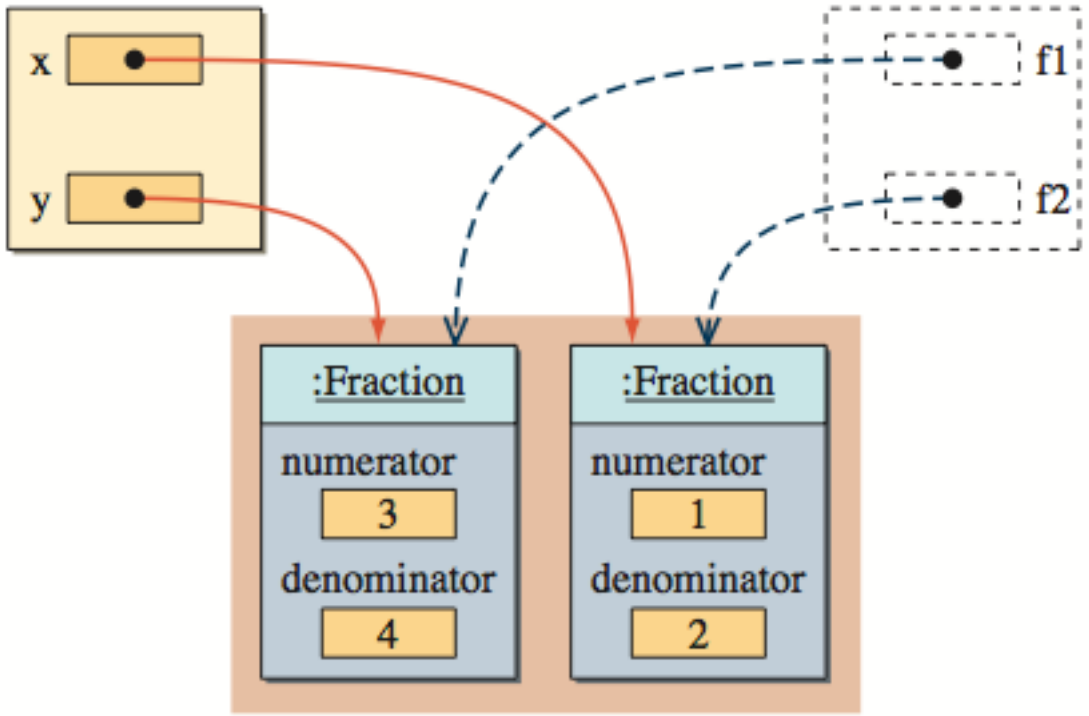
Call-by-Value Parameter Passing

At the end of the **swap** method when **f1** and **f2** are swapped.



Call-by-Value Parameter Passing

After the **swap** method terminates. No changes made to parameters **f1** and **f2** are reflected back to the arguments. Both x and y still point to the same objects as before the call.



Call-by-Value Parameter Passing

- In Java, the content of a variable is either a value of primitive data type or a reference to an object, the content of a variable is passed and copied into a parameter, it is a call by value.
- If a programming language supports the pass-by-reference mechanism, then it is possible, for example, to swap the values of two arguments in a single method call. No such thing is possible in Java.

Organizing Classes into a Package

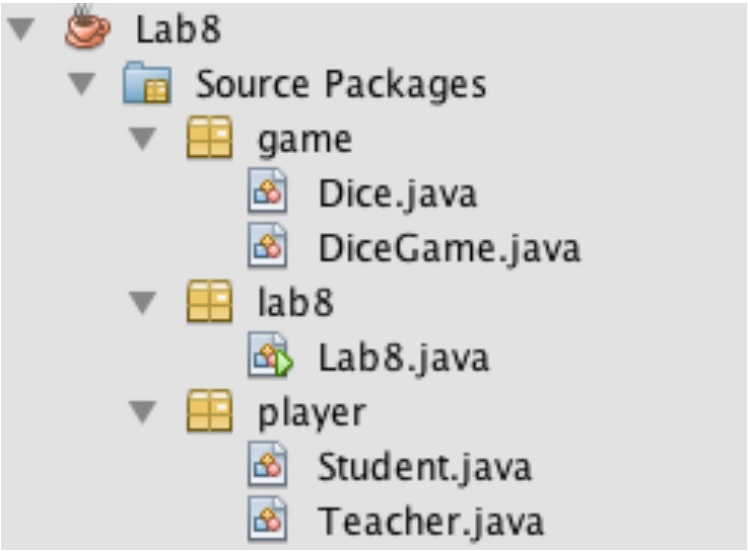
- A Java package is a mechanism for grouping related Java classes together into the same group. Conceptually you can think of packages as being similar to folders on your computer.
- When we want to use some classes, we must import the package of that classes.

Organizing Classes into a Package

- Sample Java standard packages

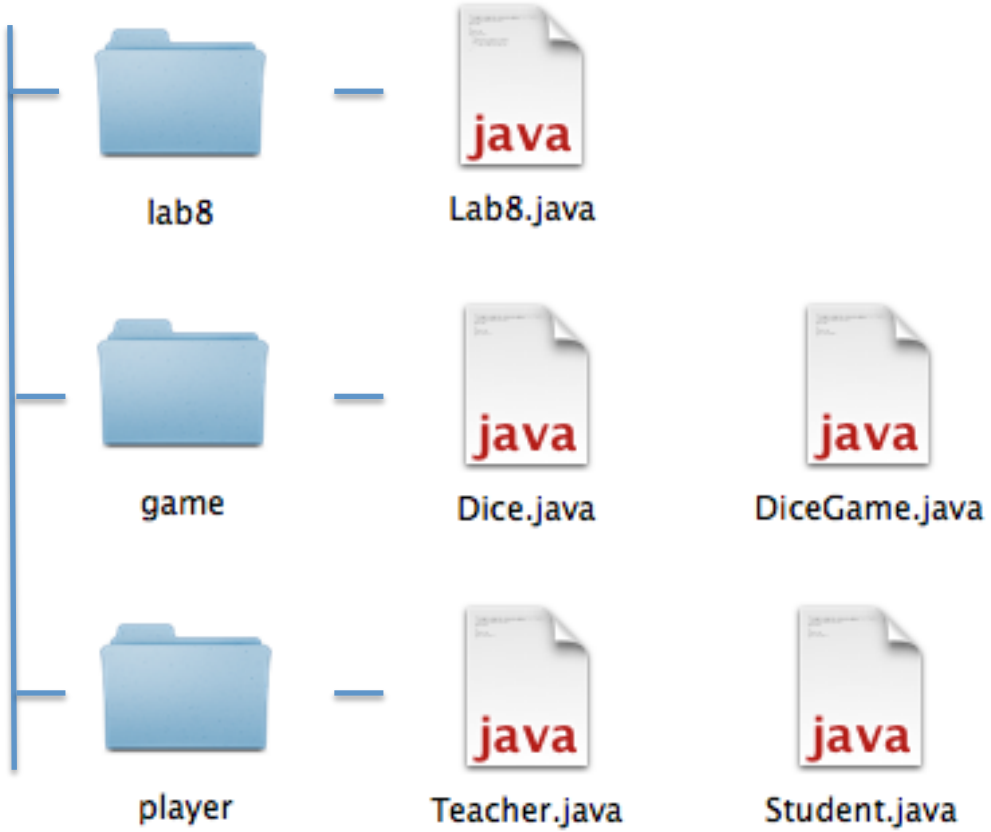
Organizing Classes into a Package

- Sample programmer-defined packages
  - In Lab8 project, we have 3 packages: lab8, game and player



Organizing Classes into a Package

– In Lab8 project's src folder



Organizing Classes into a Package

– In ‘player’ package: There are 2 classes (Student and Teacher class).

```
package player;

public class Student {

}
```

```
package player;

public class Teacher {

}
```

Organizing Classes into a Package

– In ‘game’ package: There are 2 classes (Dice and DiceGame class).

```
package game;

public class Dice {

}
```

```
package game;

public class DiceGame {

}
```

Organizing Classes into a Package

– In ‘lab8’ package (package of main class): There is 1 class (Lab8 class).

```
package lab8;

import game.*;
import player.Student;

public class Lab8 {

    public static void main(String[] args) {
        Dice d =  new Dice();
        DiceGame game = new DiceGame();

        Student s = new Student();
    }
}
```

Summary

- When a method returns an object, it is actually returning a reference to this object.
- ‘this’ is used to refer to a receiving object of a message from within this object’s method.
- A class may include multiple methods with the same name as long as their signatures are different. The signature of a method refers to the name of the method and the number and data types of its parameters. They are called overloaded methods.

Summary

- A class may include multiple constructors as long as their signatures are different. They are called overloaded constructors.
- A constructor can call another constructor of the same class using the reserved word ‘this’.
- Class variables and class methods are declared by using the reserved word ‘static’.
- Class methods can access only the class variables and the class constants.

Summary

- Instance methods can access all types of data members.
- Arguments are passed to the methods by using the call-by-value which the value of an argument is passed. The value is the actual data in the case of a primitive data type and a reference to an object in the case of a reference data type.
- Programmer-defined classes can be grouped into a package.

Reference

- C. Thomas Wu, An Introduction to Object-Oriented Programming with Java, 5th Edition
  - Chapter 7: Defining Your Own Classes - Part 2

Question?