Unit 9
- Parameter Passing in Java

Passing Parameters to Constructors and Methods

- Any expression that has an appropriate data type can serve as a parameter:

```
double u = 3, v = –4;
...
Polynomial p = new Polynomial (1.0,  –(u + v),  u * v);
double y = p.getValue (2 * v – u);
```

```
public class Polynomial
{
   public Polynomial (double a, double b, double c) { ... }
   public double getValue (double x) { ... }
   ...
```

Passing Parameters (cont'd)

- `int` is promoted to `double` when necessary:

```
...
Polynomial p = new Polynomial (1,  –5,  6);
double y = p.getValue (3);
```

The same as: **(3.0)**

The same as: **(1.0, -5.0, 6.0)**

Passing Parameters (cont'd)

- Primitive data types are always passed "by value": the value is copied into the parameter.

```
double x = 3.0;
double y = p.getValue ( x );
```

```
public class Polynomial
{
   ...
   public double getValue (double u)
   {
      double v;
      ...
   }
}
```

copy

u acts like a local variable in **getValue**

Passing Parameters (cont'd)

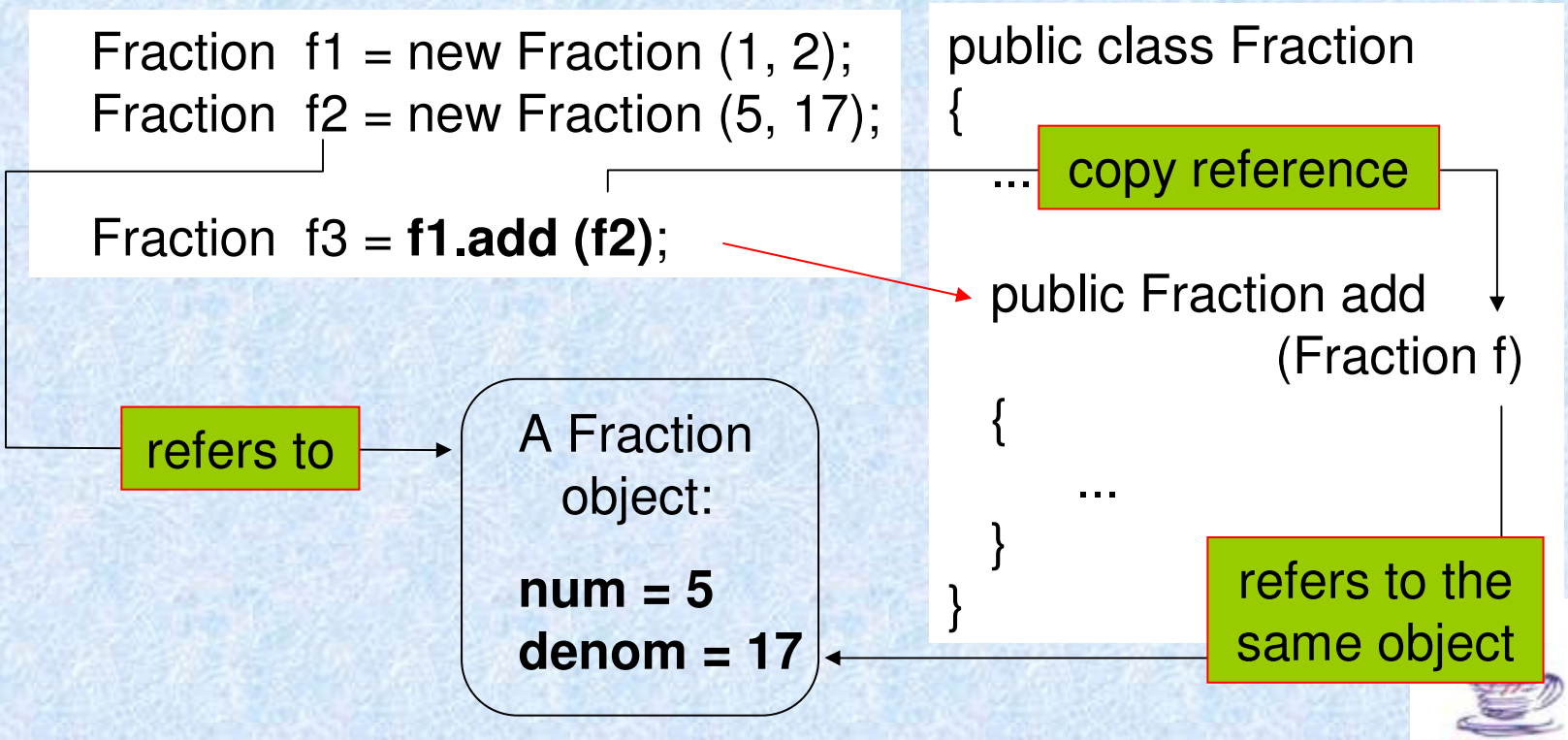
```
public class Test
{
   public double square (double x)
   {
      x *= x;
      return x;
   }

   public static void main(String[ ] args)
   {
      Test calc = new Test ();
      double x = 3.0;
      double y = calc.square (x);
      System.out.println (x + "  " + y);
   }
}
```

x here is a copy of the parameter passed to **square**. The copy is changed, but...

... the original **x** is unchanged. Output: **3 9**

Passing Parameters (cont'd)

- Objects are always passed as references: the reference is copied, not the object.



Passing Parameters (cont'd)

- A method can change an object passed to it as a parameter (because the method gets a reference to the original object).
- A method can change the object for which it was called (this object acts like an implicit parameter):

Passing Parameters (cont'd)

- Inside a method, **`this`** refers to the object for which the method was called. this can be passed to other constructors and methods as a parameter:

```
public class ChessGame
{
    ...
    Player player1 = new Player (this);
    ...
```

A reference to this **ChessGame** object

Example: Suppose the method fun is defined as:

```
public int fun(int x, int y)
{
      y -= x;
      return y;
}
```

What is printed to the screen after the following code is executed?

```
int a = 3, b = 7;
b = fun(a, b);
a = fun(b, a);
System.out.println(a + " " + b);
```