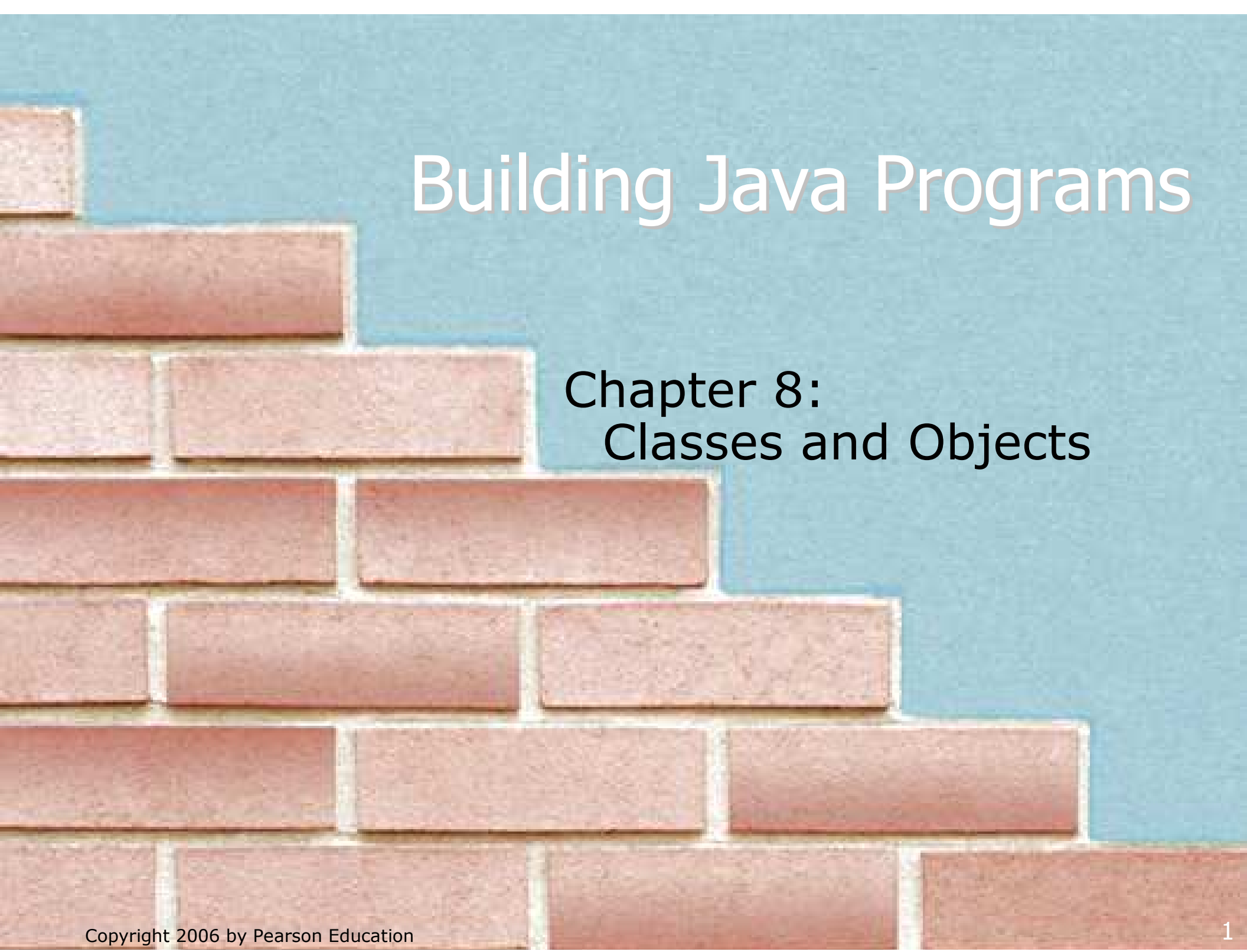


Copyright 2006 by Pearson Education

Lecture outline

- advanced classes
  - the `toString` method
  - the keyword `this`

Copyright 2006 by Pearson Education

Copyright 2006 by Pearson Education

Printing objects

- By default, Java doesn't know how to print objects:

```
Point p = new Point(10, 7);
System.out.println("p is " + p);  // p is Point@9e8c34
```

- We can print a better string (but this is cumbersome):

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

- We'd like to be able to print the object itself:

```
// desired behavior
System.out.println("p is " + p);  // p is (10, 7)
```

Copyright 2006 by Pearson Education

The toString method

- The special method `toString`:
  - Tells Java how to convert your object into a `String` as needed.
  - Is called when an object is printed or concatenated to a `String`.

```
Point p1 = new Point(7, 2);
System.out.println("p1 is " + p1);
```

  - If you prefer, you can write the `.toString()` explicitly.

```
System.out.println("p1 is " + p1.toString());
```

- Every class has a `toString`, even if it isn't in your code.
  - The default `toString` returns the class's name followed by a hexadecimal (base-16) number:

```
"Point@9e8c34"
```

Copyright 2006 by Pearson Education

toString method syntax

- You can replace the default behavior by defining a `toString` method in your class.

```
public String toString() {
    <statement(s) that return an appropriate String> ;
}
```

  - Example:

```
// Returns a String representing this Point.
public String toString() {
    return "(" + x + ", " + y + ")";
}
```

Copyright 2006 by Pearson Education

Client code question

- Recall our client program that produces this output:

```
p1 is (7, 2)
p1's distance from origin = 7.280109889280518
p2 is (4, 3)
p2's distance from origin = 5.0
p1 is (18, 8)
p2 is (5, 10)
distance from p1 to p2 = 13.0
```

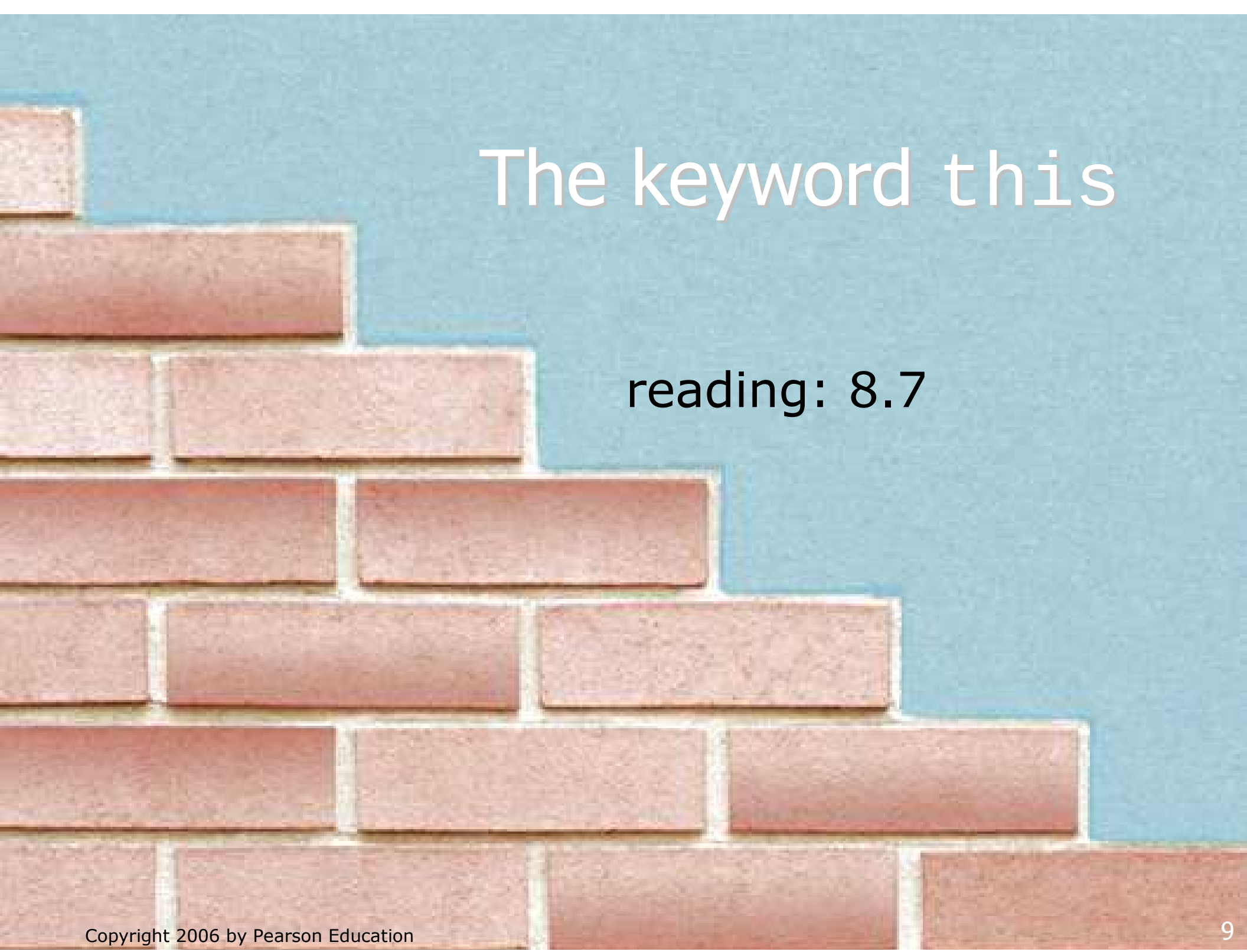
- Modify the program to use our new `toString` method.

Copyright 2006 by Pearson Education

Client code answer

```
// This client program uses the Point class.
public class PointMain {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point(7, 2);
        Point p2 = new Point(4, 3);

        // print each point
        System.out.println("p1 is " + p1);
        System.out.println("p2 is " + p2);

        // compute/print each point's distance from the origin
        System.out.println("p1's distance from origin = " + p1.distanceFromOrigin());
        System.out.println("p2's distance from origin = " + p1.distanceFromOrigin());

        // move p1 and p2 and print them again
        p1.translate(11, 6);
        p2.translate(1, 7);
        System.out.println("p1 is " + p1);
        System.out.println("p2 is " + p2);

        // compute/print distance from p1 to p2
        System.out.println("distance from p1 to p2 = " + p1.distance(p2));
    }
}
```

Copyright 2006 by Pearson Education



Copyright 2006 by Pearson Education

Using the keyword `this`

- **`this`** : A reference to the implicit parameter.
  - *implicit parameter:* object on which a method/constructor is called

- `this` keyword, general syntax:
  - To refer to a field:
    `this.`***`<field name>`***
  - To call a method:
    `this.`***`<method name>`***`(`***`<parameters>`***`);`
  - To call a constructor from another constructor:
    `this(`***`<parameters>`***`);`

Copyright 2006 by Pearson Education

Variable names and scope

- Usually it is illegal to have two variables in the same scope with the same name.

- Recall: `Point` class's `setLocation` method:
  - Params named `newX` and `newY` to be distinct from fields `x` and `y`

```
public class Point {
    int x;
    int y;
    ...
    public void setLocation(int newX, int newY) {
        if (newX < 0 || newY < 0) {
            throw new IllegalArgumentException();
        }
        x = newX;
        y = newY;
    }
}
```

Copyright 2006 by Pearson Education

Variable shadowing

- However, a class's method can have a parameter whose name is the same as one of the class's fields.
  - Example:

```
// this is legal
public void setLocation(int x, int y) {
    ...
}
```

  - Fields `x` and `y` are *shadowed* by parameters with same names.
  - Any `setLocation` code that refers to `x` or `y` will use the parameter, not the field.

- **shadowed variable**: A field that is "covered up" by a parameter or local variable with the same name.

Copyright 2006 by Pearson Education

Avoiding shadowing with this

- The keyword this prevents shadowing:

```
public class Point {
    private int x;
    private int y;

    ...

    public void setLocation(int x, int y) {
        if (x < 0 || y < 0) {
            throw new IllegalArgumentException();
        }
        this.x = x;
        this.y = y;
    }
}
```

Inside the setLocation method:

- When this.x is seen, the *field* x is used.
- When x is seen, the *parameter* x is used.

Copyright 2006 by Pearson Education

Multiple constructors

- It is legal to have more than one constructor in a class.
  - The constructors must accept different parameters.

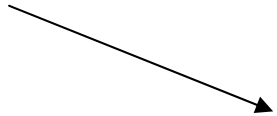
```
public class Point {
    private int x;
    private int y;

    public Point() {
        x = 0;
        y = 0;
    }

    public Point(int initialX, int initialY) {
        x = initialX;
        y = initialY;
    }

    ...
}
```

Copyright 2006 by Pearson Education

Multiple constructors w/ this

- One constructor can call another using `this`
  - We can also rename the parameters and use `this.` field syntax.

```
public class Point {
    private int x;
    private int y;

    public Point() {
        this(0, 0);   // calls the (x, y) constructor
    }

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    ...
}
```



Copyright 2006 by Pearson Education