

# Building Java Programs

Chapter 8  
Lecture 8-2: Object Behavior (Methods)  
and Constructors

**reading: 8.2 - 8.3**

Copyright 2010 by Pearson Education

## Recall: Instance methods

- **instance method** (or **object method**): Exists inside each object of a class and gives behavior to each object.

```
public type name(parameters) {  
    statements;  
}
```

- same syntax as static methods, but without `static` keyword

Example:

```
public void shout() {  
    System.out.println("HELLO THERE!");  
}
```

Copyright 2010 by Pearson Education

2

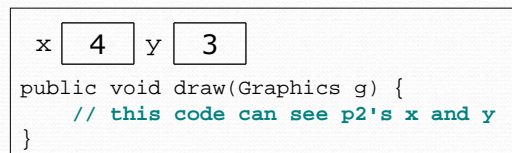
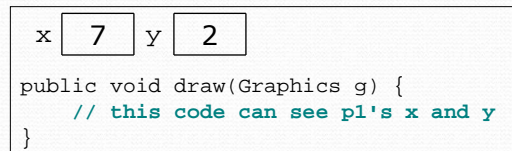
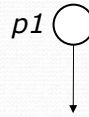
## Point objects w/ method

- Each Point object has its own copy of the draw method, which operates on that object's state:

```
Point p1 = new Point();  
p1.x = 7;  
p1.y = 2;
```

```
Point p2 = new Point();  
p2.x = 4;  
p2.y = 3;
```

```
p1.draw(g);  
p2.draw(g);
```



## The implicit parameter

- **implicit parameter:**

The object on which an instance method is called.

- During the call `p1.draw(g);`  
the object referred to by `p1` is the implicit parameter.
- During the call `p2.draw(g);`  
the object referred to by `p2` is the implicit parameter.
- The instance method can refer to that object's fields.
  - We say that it executes in the *context* of a particular object.
  - `draw` can refer to the `x` and `y` of the object it was called on.

## Point class, version 2

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

    // Changes the location of this Point object.
    public void draw(Graphics g) {
        g.fillOval(x, y, 3, 3);
        g.drawString("(" + x + ", " + y + ")", x, y);
    }
}
```

- Each Point object contains a draw method that draws that point at its current x/y position.

## Kinds of methods

- **accessor**: A method that lets clients examine object state.
  - Examples: distance, distanceFromOrigin
  - often has a non-void return type
- **mutator**: A method that modifies an object's state.
  - Examples: setLocation, translate



## Mutator method questions

- Write a method `setLocation` that changes a `Point`'s location to the  $(x, y)$  values passed.
- Write a method `translate` that changes a `Point`'s location by a given  $dx, dy$  amount.
  - Modify the `Point` and client code to use these methods.

## Mutator method answers

```
public void setLocation(int newX, int newY) {  
    x = newX;  
    y = newY;  
}
```

```
public void translate(int dx, int dy) {  
    x = x + dx;  
    y = y + dy;  
}
```

```
// alternative solution that utilizes setLocation  
public void translate(int dx, int dy) {  
    setLocation(x + dx, y + dy);  
}
```

## Accessor method questions

- Write a method `distance` that computes the distance between a `Point` and another `Point` parameter.

Use the formula:  $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

- Write a method `distanceFromOrigin` that returns the distance between a `Point` and the origin, (0, 0).
  - Modify the client code to use these methods.

## Accessor method answers

```
public double distance(Point other) {
    int dx = x - other.x;
    int dy = y - other.y;
    return Math.sqrt(dx * dx + dy * dy);
}
```

```
public double distanceFromOrigin() {
    return Math.sqrt(x * x + y * y);
}
```

```
// alternative solution that uses distance
public double distanceFromOrigin() {
    Point origin = new Point();
    return distance(origin);
}
```

# Printing objects

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

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

```
// better, but cumbersome;           p is (10, 7)
System.out.println("p is (" + p.x + ", " + p.y + ")");
```

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

# The toString method

*tells Java how to convert an object into a String*

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

// the above code is really calling the following:
System.out.println("p1: " + p1.toString());
```

- Every class has a `toString`, even if it isn't in your code.
  - Default: class's name @ object's memory address (base 16)

Point@9e8c34



## toString syntax

```
public String toString() {  
    code that returns a String representing this object;  
}
```

- Method name, return, and parameters must match exactly.
- Example:

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

## Object initialization: constructors

**reading: 8.3**

# Initializing objects

- Currently it takes 3 lines to create a `Point` and initialize it:

```
Point p = new Point();  
p.x = 3;  
p.y = 8;                                // tedious
```

- We'd rather specify the fields' initial values at the start:

```
Point p = new Point(3, 8); // better!
```

- We are able to do this with most types of objects in Java.

# Constructors

- **constructor**: Initializes the state of new objects.

```
public type(parameters) {  
    statements;  
}
```

- runs when the client uses the `new` keyword
- no return type is specified;  
it implicitly "returns" the new object being created
- If a class has no constructor, Java gives it a *default constructor* with no parameters that sets all fields to 0.



## Constructor example

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

    // Constructs a Point at the given x/y location.
    public Point(int initialX, int initialY) {
        x = initialX;
        y = initialY;
    }

    public void translate(int dx, int dy) {
        x = x + dx;
        y = y + dy;
    }

    ...
}
```

## Tracing a constructor call

- What happens when the following call is made?

```
Point p1 = new Point(7, 2);
```

p1 ○ →

<pre>x <input type="text"/>      y <input type="text"/></pre> <pre>public Point(int initialX, int initialY) {     x = initialX;     y = initialY; }  public void translate(int dx, int dy) {     x += dx;     y += dy; }</pre>
--

## Client code, version 3

```
public class PointMain3 {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point(5, 2);
        Point p2 = new Point(4, 3);

        // print each point
        System.out.println("p1: (" + p1.x + ", " + p1.y + ")");
        System.out.println("p2: (" + p2.x + ", " + p2.y + ")");

        // move p2 and then print it again
        p2.translate(2, 4);
        System.out.println("p2: (" + p2.x + ", " + p2.y + ")");
    }
}
```

### OUTPUT:

```
p1: (5, 2)
p2: (4, 3)
p2: (6, 7)
```

## Multiple constructors

- A class can have multiple constructors.
  - Each one must accept a unique set of parameters.
- *Exercise:* Write a `Point` constructor with no parameters that initializes the point to (0, 0).

```
// Constructs a new point at (0, 0).
public Point() {
    x = 0;
    y = 0;
}
```

# Common constructor bugs

## 1. Re-declaring fields as local variables ("shadowing"):

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

- This declares local variables with the same name as the fields, rather than storing values into the fields. The fields remain 0.

## 2. Accidentally giving the constructor a return type:

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

- This is actually not a constructor, but a method named `Point`