Building Java Programs

Chapter 8

Lecture 8-2: Object Behavior (Methods)

reading: 8.3

self-checks: #1-12

exercises: #1-4, 9, 11, 14, 16

Client code redundancy

- Our client program wants to draw `Point` objects:

```
// draw each city
g.fillOval(cities[i].x, cities[i].y, 3, 3);
g.drawString("(" + cities[i].x + ", " + cities[i].y + ")",
             cities[i].x, cities[i].y);
```

- To draw them in other places, the code must be repeated.
  - We can remove this redundancy using a method.

Eliminating redundancy, v1

- We can eliminate the redundancy with a static method:

```
// Draws the given point on the DrawingPanel.
public static void draw(Point p, Graphics g) {
    g.fillOval(p.x, p.y, 3, 3);
    g.drawString("(" + p.x + ", " + p.y + ")", p.x, p.y);
}
```

- `main` would call the method as follows:

```
// draw each city
draw(cities[i], g);
```

Problems with static solution

- We are missing a major benefit of objects: code reuse.
  - Every program that draws `Point`s would need a `draw` method.

- The syntax doesn't match how we're used to using objects.

```
draw(cities[i], g);    // static (bad)
```

- The point of classes is to combine state and behavior.
  - The `draw` behavior is closely related to a `Point`'s data.
  - The method belongs *inside* each `Point` object.

```
cities[i].draw(g);     // inside object (better)
```

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!");
  }
  ```

Instance method example

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

    // Draws this Point object with the given pen.
    public void draw(Graphics g) {
        ...
    }
}
```

- The `draw` method no longer has a `Point p` parameter.
- How will the method know which point to draw?
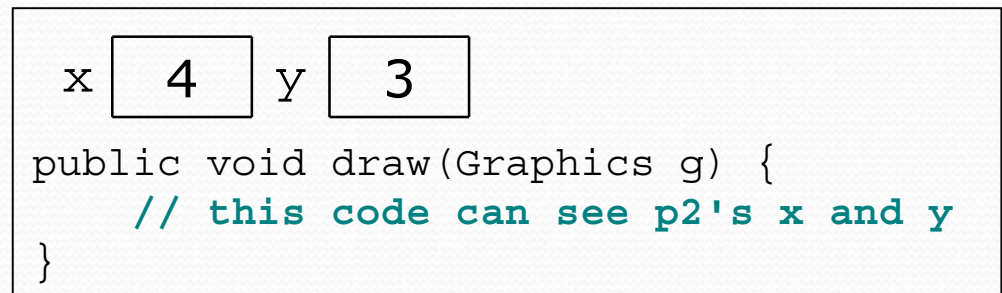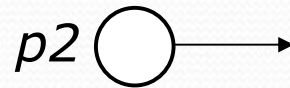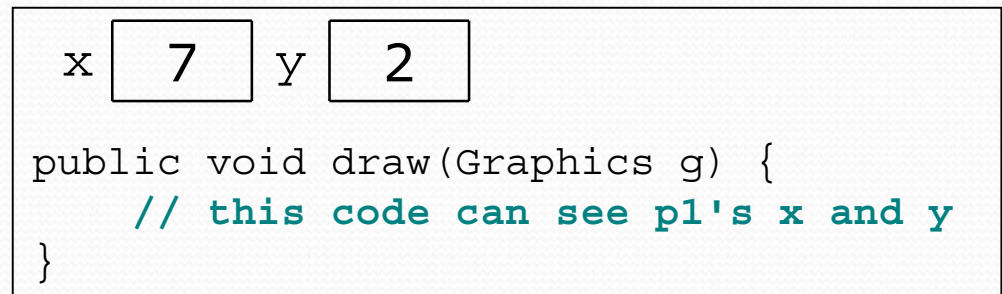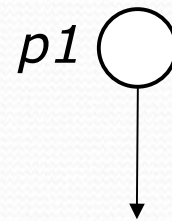 - How will the method access that point's x/y data?

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);
```

p1 ◯

x [7] y [2]

```
public void draw(Graphics g) {
    // this code can see p1's x and y
}
```

p2 ◯ ⟶

x [4] y [3]

```
public void draw(Graphics g) {
    // this code can see p2's x and y
}
```

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

```java
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

```java
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);
}
```