Encapsulation, Constructors, More Instance Methods

BEFORE WE START

Talk to your neighbors:
What godforsaken creature do you think is making those noises in Taylor Swift’s Is It Over Now?

Music: I don’t Live Here Anymore - The War on Drugs, Lucius
Lecture Outline

- Announcements
- Warm Up
- More Instance Methods
- Encapsulation
- Constructors
Announcements

• *Minimum* grade guarantees in [Syllabus](#)
  - Minimum grade calculator tool
• Programming Assignment 2 (P2) still out; due November 9th
• Resubmission Cycle 3 (R3) is out; due November 7th
• Next week: No class on Friday!
• An aside...
Lecture Outline

• Announcements

• **Warm Up**

• More Instance Methods

• Encapsulation

• Constructors
What do p and p2 hold after the following code is executed?

```java
Point p = new Point();
p.x = 3;
p.y = 10;
Point p2 = p;
p2.y = 100;
p = new Point();
p.y = -99;
```

A. p: (3, 10)       p2: (3, 10)
B. p: (3, -99)      p2: (3, 100)
C. p: (0, -99)      p2: (3, 100)
D. p: (3, -99)      p2: (0, 100)
E. p: (0, -99)      p2: (3, 10)
Practice: Pair

What do p and p2 hold after the following code is executed?

```java
Point p = new Point();
p.x = 3;
p.y = 10;
Point p2 = p;
p2.y = 100;
p = new Point();
p.y = -99;
```

A. p: (3, 10)      p2: (3, 10)
B. p: (3, -99)     p2: (3, 100)
C. p: (0, -99)     p2: (3, 100)
D. p: (3, -99)     p2: (0, 100)
E. p: (0, -99)     p2: (3, 10)

p: 0, -99
p2: old p i.e. (3, 100)
Lecture Outline

• Announcements
• Warm Up
• More Instance Methods
• Encapsulation
• Constructors
Client v. Implementor

We have been the clients of many objects this quarter!

Now we will become the implementors of our own objects!
Practice: Think

What is the correct implementation of the `distanceFrom` instance method?

\[ \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \]

(A) 
```java
public double distanceFrom() {
    double xTerm = Math.pow(x - x, 2);
    double yTerm = Math.pow(y - y, 2);
    return Math.sqrt(xTerm + yTerm);
}
```

(B) 
```java
public static double distanceFrom(Point otherPoint) {
    double xTerm = Math.pow(otherPoint.x - x, 2);
    double yTerm = Math.pow(otherPoint.y - y, 2);
    return Math.sqrt(xTerm + yTerm);
}
```

(C) 
```java
public double distanceFrom(Point otherPoint) {
    double xTerm = Math.pow(otherPoint.x - x, 2);
    double yTerm = Math.pow(otherPoint.y - y, 2);
    return Math.sqrt(xTerm + yTerm);
}
```

(D) 
```java
public double distanceFrom(int otherX, int otherY) {
    double xTerm = Math.pow(otherX - x, 2);
    double yTerm = Math.pow(otherY - y, 2);
    return Math.sqrt(xTerm + yTerm);
}
```
Practice: Pair

What is the correct implementation of the distanceFrom instance method?

\[
\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}
\]

(A)

```java
public double distanceFrom() {
    double xTerm = Math.pow(x - x, 2);
    double yTerm = Math.pow(y - y, 2);
    return Math.sqrt(xTerm + yTerm);
}
```

(B)

```java
public static double distanceFrom(Point otherPoint) {
    double xTerm = Math.pow(otherPoint.x - x, 2);
    double yTerm = Math.pow(otherPoint.y - y, 2);
    return Math.sqrt(xTerm + yTerm);
}
```

(C)

```java
public double distanceFrom(Point otherPoint) {
    double xTerm = Math.pow(otherPoint.x - x, 2);
    double yTerm = Math.pow(otherPoint.y - y, 2);
    return Math.sqrt(xTerm + yTerm);
}
```

(D)

```java
public double distanceFrom(int otherX, int otherY) {
    double xTerm = Math.pow(otherX - x, 2);
    double yTerm = Math.pow(otherY - y, 2);
    return Math.sqrt(xTerm + yTerm);
}
```
**toString**

```java
public String toString() {
    return "String representation of object";
}
```

The `toString()` method is automatically called whenever an object is treated like a `String`!
**toString**

```java
public String toString() {
    return "String representation of object";
}
```

The **toString()** method is **automatically** called whenever an object is treated like a String!

**Wait**: Why not write a **print()** method that prints out the String representation to the console? All **toString()** does is return a String!
Lecture Outline

• Announcements
• Warm Up
• More Instance Methods

• Encapsulation
• Constructors
Encapsulation

Objects **encapsulate** state and expose behavior.

**Encapsulation** is hiding implementation details of an object from its clients. (Clients = chaos, y’all.)

Encapsulation provides *abstraction*.
private

The `private` keyword is an access modifier (like `public`)

Fields declared `private` cannot be accessed by any code outside of the object.

We always want to encapsulate our objects’ fields by declaring them `private`. 
## Accessors and Mutators

Declaring fields as private removes all access from the user.

If we want to give some back, we can define instance methods.

<table>
<thead>
<tr>
<th>Accessors (“getters”)</th>
<th>Mutators (“setters”)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getX()</code></td>
<td><code>setX(int newX)</code></td>
</tr>
<tr>
<td><code>getY()</code></td>
<td><code>setY(int newY)</code></td>
</tr>
<tr>
<td></td>
<td><code>setLocation(int newX, int newY)</code></td>
</tr>
</tbody>
</table>
Encapsulation

Objects **encapsulate** state and expose behavior.

**Encapsulation** is hiding implementation details of an object from its clients.

Encapsulation provides *abstraction*.

Encapsulation also gives the implementor flexibility!
Encapsulation

While users can still access and modify our Point’s fields with the instance methods we defined, *we have control of how they do so.*

Can only accept positive coordinate values

Can swap out our underlying implementation to use polar coordinates instead!
Lecture Outline

• Announcements
• Warm Up
• More Instance Methods
• Encapsulation
• Constructors
Constructors

Constructors are called when we first create a new instance of a class.

```java
Point p = new Point();
```

If we don’t write any constructors, Java provides one that takes no parameters and just sets each field to its default value.
Constructor Syntax

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

The **this** keyword refers to the current object in a method or constructor.

You can use it to refer to an object’s fields

```
this.x, this.y
```

You can use it to refer to an object’s instance methods

```
this.setX(newX)
```
Constructor Syntax

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

If we write any constructors, Java no longer provides one for us.
this keyword

The `this` keyword refers to the current object in a method or constructor.

You can use it to refer to an object’s fields

```
this.x, this.y
```

You can use it to refer to an object’s instance methods

```
this.setX(newX)
```

You can use it to call one constructor from another

```
this(0, 0)
```