Constructors, More Instance Methods

Talk to your neighbors:

Share a boring fact about yourself!

Instructor: Melissa Lin
TAs:
- Poojitha Arangam
- Darel Gunawan
- Colton Harris
- Atharva Kashyap
- Eesha Kunisetty

Audrey Lin
Di Mao
Steven Nguyen
Ben Wang
Jaylyn Zhang
Lecture Outline

• Announcements

• Warm up

• Review: Encapsulation, Constructors, toString()

• Larger Example
  - Code Quality
Announcements

• P2: Absurdle due yesterday
• Resub 4 form + C2 released today
Lecture Outline

• Announcements

• Warm up

• Review: Encapsulation, Constructors, toString()

• Larger Example
  - Code Quality
What is the correct implementation of the distanceFrom instance method?

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

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

(C) `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) `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); }`
What is the correct implementation of the distanceFrom instance method?

(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 : Think

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

Point p = new Point();
p.setX(3);
p.setY(10);
Point p2 = p;
p2.setY(100);
p = new Point();
p.setY(-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)
What do p and p2 hold after the following code is executed?

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

```
p: (0, -99)  
p2: (3, 100)
```
Lecture Outline

• Announcements

• Warm up

• Review: Encapsulation, Constructors, toString()

• Larger Example
  - Code Quality
Encapsulation

Objects *encapsulate* state and expose behavior.

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

Encapsulation provides *abstraction*.
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 (&quot;getters&quot;)</th>
<th>Mutators (&quot;setters&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>getX()</td>
<td>setX(int newX)</td>
</tr>
<tr>
<td>getY()</td>
<td>setY(int newY)</td>
</tr>
<tr>
<td></td>
<td>setLocation(int newX, int newY)</td>
</tr>
</tbody>
</table>
(PCM) 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!
Constructors

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

\[
\text{Point } p = \text{ new Point();}
\]

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;
}
```

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)
```
**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

\[
\text{this}.x, \text{this}.y
\]

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

\[
\text{this}.\text{setX}(\text{newX})
\]

You can use it to call one constructor from another

\[
\text{this}(0, 0)
\]
(PCM) toString

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

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

Why not write a print() method that prints out the String representation to the console?
Lecture Outline

• Announcements

• Warm up

• Review: Encapsulation, Constructors, toString()

• Larger Example
  - Code Quality
Lecture Outline

• Announcements

• Warm up

• Review: Encapsulation, Constructors, toString()

• Larger Example
  - Code Quality