LEC 12

CSE 122

Encapsulation, Constructors, More Instance Methods

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Chat with neighbors:

What are you looking forward to for the summer?

Music: 122 25sp Lecture Tunes

Instructor: Brett Wortzman and Adrian Salguero

TAs:

Andrew	Diya	Logan	Steven
Anya	Elizabeth	Mahima	Yang
Brittan	Ivory	Medha	
Carson	Jack	Minh	
Christopher	Jacob	Nicole	
Colin	Ken	Samuel	
Dalton	Kyle	Shivani	
Daniel	Leo	Sreshta	

Lecture Outline

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

Announcements

- Programming Assignment 2 (P2) due tomorrow, Thursday May 15th
  - Creative Project 2 will be released on Friday, focused on OOP
  - Due **Friday May 23rd**
- Resubmission Cycle 4 (R4) out tomorrow
  - **C1**, P1
- Quiz 1 was yesterday
  - Grades will be released before Quiz 2

Lecture Outline

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

Practice : Think

sli.do #cse122

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

```
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

sli.do #cse122

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

```
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)

Lecture Outline

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

static keyword

- A `static` method is a method that does not rely on the state of the class
  - i.e. It does not access fields of the class
- Before, our classes did not have fields (we were in procedural programming word), so all of our methods were `static`

Example #1

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

    public _ int distaceFromX(int otherX) {
        return Math.abs(x - otherX);
    }
}
```

Example #1

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

    public int distaceFromX(int otherX) {
        return Math.abs(x - otherX);
    }
}
```

Example #2

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

    public _ int diff(int xx, int yy) {
        return xx - yy;
    }
}
```

Example #2

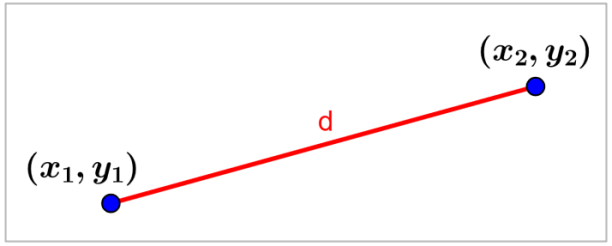
```
public class Point {
    int x;
    int y;

    public static int diff(int xx, int yy) {
        return xx - yy;
    }
}
```

Practice : Think

sli.do #cse122

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

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



(A)

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

(B)

```
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)

```
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)

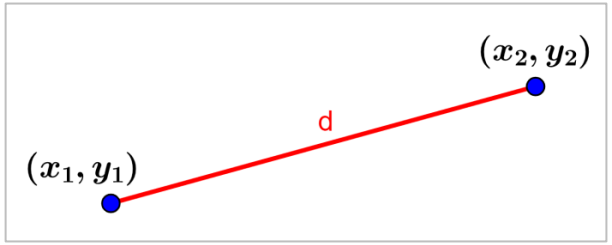
```
public double distanceFrom(int otherX, int otherY) {
    double xTerm = Math.pow(otherX - x, 2);
    double yTerm = Math.pow(otherY - y, 2);
    return Math.sqrt(xTerm + yTerm);
}
```

Practice : Pair

sli.do #cse122

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

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



(A)

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

(B)

```
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)

```
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);
}
```

toString

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

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

toString

```
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

Abstraction

The separation of ideas from details, meaning that we can use something without knowing exactly how it works.

You were able use the `Scanner` class without understanding how it works internally!

Client v. Implementor

We have been the clients of many objects this quarter!

Now we will become the implementors of our own objects!

Encapsulation

Objects **encapsulate** state and expose behavior.

Encapsulation is hiding implementation details of an object from its clients. (Clients = chaos)

Encapsulation provides *abstraction*.

Encapsulation also gives the implementor flexibility!

private

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

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

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.

Accessors ("getters")	Mutators ("setters")
`getX()`	`setX(int newX)`
`getY()`	`setY(int newY)`
	`setLocation(int newX, int newY)`

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*.

Example: Can only accept positive coordinate values

Another Example: 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.

```
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

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

If we write <u>any</u> constructors, Java no longer provides one for us.