UNIVERSITY *of* WASHINGTON

LEC 12

CSE 122

Encapsulation, Constructors, More Instance Methods

Questions during Class?

Raise hand or send here

sli.do   #cse122

BEFORE WE START

Slido vote & chat with neighbors:

What are you most looking forward to in spring?

Music: [122 25wi Lecture Tunes](#)

Instructor:	Elba Garza		
TAs:	Anya	Daniel Ryan	Ken	Nicole
	Ashley	Diya	Kuhu	Nicole
	Cady	Elizabeth	Kyle	Niyati
	Caleb	Hannah	Leo	Sai
	Carson	Harshitha	Logan	Steven
	Chaafen	Ivory	Maggie	Yang
	Colin	Izak	Mahima	Zach
	Connor	Jack	Marcus	
	Dalton	Jacob	Minh	

Lecture Outline

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

Announcements

- Programming Assignment 2 (P2) due tomorrow, Thursday Feb 20th
  - Creative Project 2 will be released on Friday, focused on OOP
- Resubmission Cycle 4 (R4) for out soon!
- Quiz 1 was yesterday, we have some quiz makeups to administer then we'll be releasing grades
  - Grades will be released just before Quiz 2

Lecture Outline

- Announcements

- **Warm Up** ◀

- More Instance Methods

- Encapsulation

- Constructors

Practice : Think

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

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)

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

Lecture Outline

- Announcements

- Warm Up

- **More Instance Methods** ◀

- Encapsulation

- Constructors

Abstraction

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

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

Client v. Implementor

We have been the <u>clients</u> of many objects this quarter!

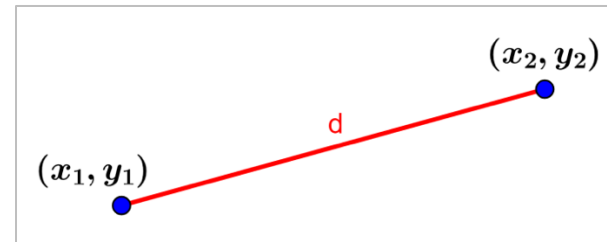Now we will become the <u>implementors</u> of our own objects!

Practice : Think

What is the correct implementation of the `distanceFrom` <u>instance</u> method?

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

(x_2, y_2)

d

(x_1, y_1)

(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);
}
```
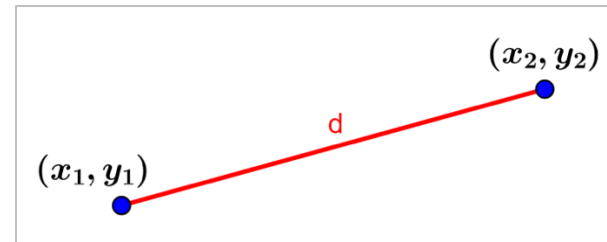
Practice : Pair

sli.do   #cse122

What is the correct implementation of the `distanceFrom` <u>instance</u> 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 <u>automatically</u> called whenever an object is treated like a String!

toString

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

The `toString()` method is <u>automatically</u> 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 <u>return</u> 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 class.

We <u>always</u> 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

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*.

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

```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 <u>any</u> 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)`