LEC 12

CSE 122

Encapsulation, Constructors, More Instance Methods

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Talk to your neighbors:

Favorite flavor to pair with chocolate?

Coffee? Orange? Caramel? Strawberry? Matcha?

Music: [122 24sp Lecture Tunes ❀]

Instructors: Miya Natsuhara and Kasey Champion

TAs:

Ayush
Poojitha
Chloe
Ailsa
Jasmine
Lucas
Logan

Kyle
Jacob
Atharva
Rucha
Megana
Eesha
Zane

Colin
Ronald
Saivi
Shivani
Kavya
Steven
Ken

Chaafen
Smriti
Ambika
Elizabeth
Aishah
Minh
Katharine

Lecture Outline

- **Announcements** ◀
- Formative Feedback: Closing the Loop
- Warm Up
- More Instance Methods
- Encapsulation
- Constructors

Announcements

- Programming Assignment 2 (P2) due tomorrow, Thursday May 9
  - Creative Project 2 will be released on Friday, focused on OOP
- *Minimum* grade guarantees in Syllabus
  - Minimum grade calculator tool
- Quiz 1 was yesterday, we have some quiz makeups to administer then we'll be releasing grades
  - Grades will be released before Quiz 2

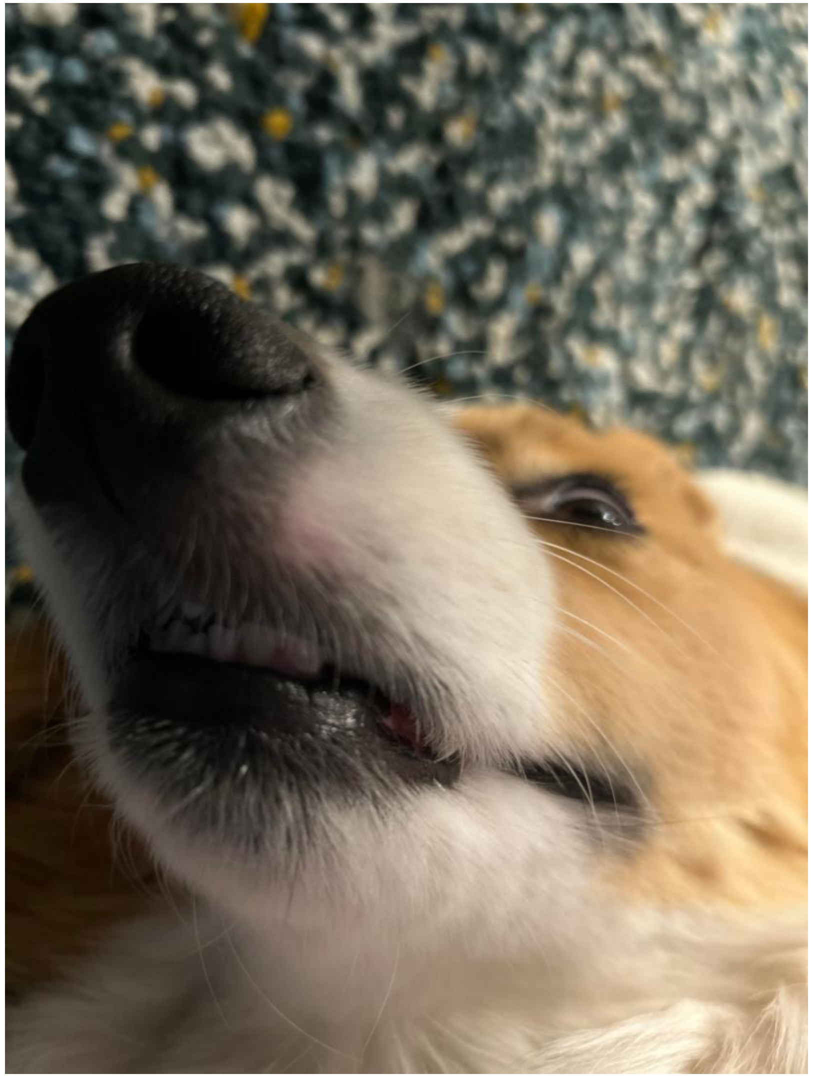
Lecture Outline

- Announcements
- **Formative Feedback: Closing the Loop** ◀
- Warm Up
- More Instance Methods
- Encapsulation
- Constructors

Closing the Loop: The Good

- Quiz sections
- IPL
- PCMs
- Live-coding during lecture!

Closing the Loop: Suggestions



- More practices quizzes
- Less review of PCMs in lecture
- Both "slow down" and "speed up" …
- Release assignments earlier in the day

Closing the Loop: Reminders

- Think-Pair-Share activities serve multiple purposes!
- Office hours outside of the IPL

Lecture Outline

- Announcements
- Formative Feedback: Closing the Loop
- **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
- Formative Feedback: Closing the Loop
- 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!

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
- Formative Feedback: Closing the Loop
- 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 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

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
- Formative Feedback: Closing the Loop
- 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

```
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)
```