LEC 11

CSE 122

More Constructors, Encapsulation

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Slido vote & chat with neighbors:

What is your favorite Ed announcement pet photo?

Music: 122 26sp Lecture Jams 🎧

Instructor: Elba Garzas

TAs:

David	Caleb	Cole	Yang
William	Neha	Blake R.	Cady
Dani	Wesley	Carson	Diya
Rohan	Isis	Sushma	Elba Garza
Andrew	Colin	Connor	
Ava	Naomi	Mahima	
Shreyank	Hanna	Nicolae	
Nicole	Blake P.	Ivory	

Lecture Outline

- Announcements
- Warm Up
- Constructors
- Encapsulation

Announcements

- Reminder: P2 has an extended deadline!
  - Make sure to submit by May 12th 11:59pm PT
- Resubmission Cycle 3 (R3) due May 12th!
  - P0, C1, P1 eligible for resub
- Quiz 1 was yesterday; grades will be released just before Quiz 2!
  - Quiz 2 on Tuesday, May 26th! (Nested Collections, Objects, and Interfaces)

Lecture Outline

- Announcements
- Warm Up
- Constructors
- Encapsulation

Practice : Think

sli.do #cse122

What do c1 and c2 hold after the following code is executed?

```
ShoppingCart c1 = new ShoppingCart();
c1.items.add("tub");
c1.capacity = 1;
ShoppingCart c2 = c1;
c2.capacity = 100;
c1 = new ShoppingCart();
c1.capacity = 1;
```

A. c1: 1, [] c2: 1, []

B. c1: 1, ["tub"] c2: 100, ["tub"]

C. c1: 1, ["tub"] c2: 1, ["tub"]

D. c1: 1, [] c2: 100, ["tub"]

E. c1: 100, [] c2: 1, ["tub"]

Practice : Pair

sli.do #cse122

What do c1 and c2 hold after the following code is executed?

```
ShoppingCart c1 = new ShoppingCart();
c1.items.add("tub");
c1.capacity = 1;
ShoppingCart c2 = c1;
c2.capacity = 100;
c1 = new ShoppingCart();
c1.capacity = 1;
```

A. c1: 1, [] c2: 1, []

B. c1: 1, ["tub"] c2: 100, ["tub"]

C. c1: 1, ["tub"] c2: 1, ["tub"]

D. c1: 1, [] c2: 100, ["tub"]

E. c1: 100, [] c2: 1, ["tub"]

Practice : Pair

sli.do #cse122

What do c1 and c2 hold after the following code is executed?

```
ShoppingCart c1 = new ShoppingCart();
c1.items.add("tub");
c1.capacity = 1;
ShoppingCart c2 = c1;
c2.capacity = 100;
c1 = new ShoppingCart();
c1.capacity = 1;
```

c1: 0, []

A. c1: 1, [] c2: 1, []

B. c1: 1, ["tub"] c2: 100, ["tub"]

C. c1: 1, ["tub"] c2: 1, ["tub"]

D. c1: 1, [] c2: 100, ["tub"]

E. c1: 100, [] c2: 1, ["tub"]

Practice : Pair

sli.do #cse122

What do c1 and c2 hold after the following code is executed?

```
ShoppingCart c1 = new ShoppingCart();
c1.items.add("tub");
c1.capacity = 1;
ShoppingCart c2 = c1;
c2.capacity = 100;
c1 = new ShoppingCart();
c1.capacity = 1;
```

c1: 0, ["tub"]

A. c1: 1, [] c2: 1, []

B. c1: 1, ["tub"] c2: 100, ["tub"]

C. c1: 1, ["tub"] c2: 1, ["tub"]

D. c1: 1, [] c2: 100, ["tub"]

E. c1: 100, [] c2: 1, ["tub"]

Practice : Pair

sli.do #cse122

What do c1 and c2 hold after the following code is executed?

```
ShoppingCart c1 = new ShoppingCart();
c1.items.add("tub");
c1.capacity = 1;
ShoppingCart c2 = c1;
c2.capacity = 100;
c1 = new ShoppingCart();
c1.capacity = 1;
```

c1: 1, ["tub"]

A. c1: 1, [] c2: 1, []

B. c1: 1, ["tub"] c2: 100, ["tub"]

C. c1: 1, ["tub"] c2: 1, ["tub"]

D. c1: 1, [] c2: 100, ["tub"]

E. c1: 100, [] c2: 1, ["tub"]

Practice : Pair

sli.do #cse122

What do c1 and c2 hold after the following code is executed?

```
ShoppingCart c1 = new ShoppingCart();
c1.items.add("tub");
c1.capacity = 1;
ShoppingCart c2 = c1;
c2.capacity = 100;
c1 = new ShoppingCart();
c1.capacity = 1;
```

A. c1: 1, [] c2: 1, []

B. c1: 1, ["tub"] c2: 100, ["tub"]

C. c1: 1, ["tub"] c2: 1, ["tub"]

D. c1: 1, [] c2: 100, ["tub"]

E. c1: 100, [] c2: 1, ["tub"]

Practice : Pair

sli.do #cse122

What do c1 and c2 hold after the following code is executed?

```
ShoppingCart c1 = new ShoppingCart();
c1.items.add("tub");
c1.capacity = 1;
ShoppingCart c2 = c1;
c2.capacity = 100;
c1 = new ShoppingCart();
c1.capacity = 1;
```

A. c1: 1, [] c2: 1, []

B. c1: 1, ["tub"] c2: 100, ["tub"]

C. c1: 1, ["tub"] c2: 1, ["tub"]

D. c1: 1, [] c2: 100, ["tub"]

E. c1: 100, [] c2: 1, ["tub"]

Practice : Pair

sli.do #cse122

What do c1 and c2 hold after the following code is executed?

```
ShoppingCart c1 = new ShoppingCart();
c1.items.add("tub");
c1.capacity = 1;
ShoppingCart c2 = c1;
c2.capacity = 100;
c1 = new ShoppingCart();
c1.capacity = 1;
```

A. c1: 1, [] c2: 1, []

B. c1: 1, ["tub"] c2: 100, ["tub"]

C. c1: 1, ["tub"] c2: 1, ["tub"]

D. c1: 1, [] c2: 100, ["tub"]

E. c1: 100, [] c2: 1, ["tub"]

c1: 0, [] c2: 100, ["tub"]

Practice : Pair

sli.do #cse122

What do c1 and c2 hold after the following code is executed?

```
ShoppingCart c1 = new ShoppingCart();
c1.items.add("tub");
c1.capacity = 1;
ShoppingCart c2 = c1;
c2.capacity = 100;
c1 = new ShoppingCart();
c1.capacity = 1;
```

A. c1: 1, [] c2: 1, []

B. c1: 1, ["tub"] c2: 100, ["tub"]

C. c1: 1, ["tub"] c2: 1, ["tub"]

D. c1: 1, [] c2: 100, ["tub"]

E. c1: 100, [] c2: 1, ["tub"]

c1: 1, [] c2: 100, ["tub"]

Lecture Outline

- Announcements
- Warm Up
- **Constructors**
- Encapsulation

Constructors

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

```
ShoppingCart cart = new ShoppingCart();
```

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

Constructor Parameters

Constructors can take parameters just like a method.

```
ShoppingCart cart = new ShoppingCart(…);
```

It is up to you to decide what parameters make sense!

Constructor Syntax

```
public ShoppingCart(int initialCapacity) {
    items = new HashSet<>();
    capacity = initialCapacity;
}
```

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.capacity, this.items
```

Multiple Constructors

Java allows you to define more than 1 constructor for a class!

When you call `new ShoppingCart(…)`, Java chooses the constructor that matches your arguments.

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.capacity, this.items
```

You can use it to call one constructor from another

```
this(3)
```

Lecture Outline

- Announcements
- Warm Up
- Constructors
- **Encapsulation**

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, 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")
`getItems()`	`setItems(Set<String> items)`
`getCapacity()`	`setCapacity(int capacity)`

Encapsulation

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

Example: Can only accept a capacity with a positive value

Another Example: Can swap out our underlying implementation to use queues instead!