LEC 10

CSE 122

Constructors; Encapsulation

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Talk to your neighbors:

What is your favorite warm drink? 🧉 🍵

Respond on sli.do!

Instructor: Brett Wortzman

TAs:

Archit	Daniel	Naomi	Sreshta
Blake P.	David	Neha	Suyash
Blake R.	Hanna	Nicole	Trisha
Cady	Isis	Rio	Wesley
Carson	Jun	Rohan	Yash
Cole	Landon	Shreyank	
Connor	Michelle	Somya	

Music: 🎷 CSE 122 26au Lecture Tunes 🎶

Lecture Outline

- Announcements
- Warm Up
- Review: Constructors
- Review: Encapsulation

Announcements

- CQ1 closes Friday
  - ***No slots remaining for Friday!***
  - If you have not scheduled, email Brett ASAP
- PQ1 Tuesday 11/10
  - Practice quiz hopefully out later today
- PQ0 grades out this weekend
- P2 out Friday 11/6, due Thursday 11/12

Lecture Outline

- Announcements
- **Warm Up** 
- Review: Constructors
- Review: 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
- **Review: Constructors**
- Review: Encapsulation

(PCM) 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.

(PCM) 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!

(PCM) Constructor Syntax

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

Notice ***no return type!***

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

(PCM) `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`

(PCM) 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.

(PCM) `this` keyword (again)

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 also use it to call one constructor from another

```
this(3)
```

Lecture Outline

- Announcements
- Warm Up
- Review: Constructors
- **Review: 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!

(PCM) Encapsulation

Encapsulation is protecting internal features of a class/object from clients.

Clients should be able to think about a "shopping cart," not a collection of fields and methods.

Encapsulation supports *abstraction*.

(PCM) Access Modifiers

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

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

We always want to encapsulate our objects' fields by declaring them `private`.

(PCM) 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(List<String> items)`
`getCapacity()`	`setCapacity(int capacity)`

(PCM) 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!