LEC 12

CSE 122

Advanced OOP

BEFORE WE START

Slido vote & chat with neighbors:

What is your favorite warm drink?

Music: 122 26sp Lecture Jams 🎧

Instructor: Elba Garza

TAs:

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

Questions during Class?

Raise hand or send here

sli.do #cse122

Lecture Outline (1/4)

- Announcements
- Recap
- Equals
- Bigger Example

Announcements

- Creative Project 2 will be released later today!
  - Focused on OOP
  - Due **Thursday**, May 21st by 11:59pm PT
- Resubmission Cycle 4 (R4) out soon, too
  - Due Tuesday, May 19th by 11:59pm PT
  - Eligible assignments: **C1**, P1
- Quiz 2 on **Tuesday**, May 26th—coming up fast!
  - Topics: Nested Collections, Objects, and Interfaces
  - Expect Quiz 1 grades day(s) before Quiz 2

Lecture Outline (2/4)

- Announcements
- Recap
- Equals
- Bigger Example

Constructor Syntax

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

All fields should be initialized in the constructor(s)!

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

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 refer to an object's instance methods:

```
this.setCapacity(newCapacity)
```

Also, you can use it to call one constructor from another:

```
this(3)
```

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`.

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!

Lecture Outline (3/4)

- Announcements
- Recap
- **Equals**
- Bigger Example
- For next quarter…

Equals

The `equals()` method returns `true` if the given parameter is considered equal to this object, and `false` otherwise.

- Used by lots of library methods! e.g. `contains`, `remove` for specific elements, etc.

Let's do it!

Each class has one provided by Java, but it checks for **reference equality**. (Thanks?)

If you want equals to check for **value equality**, you need to write this method yourself.

Object

By taking a parameter of type `Object`, the equals method can be passed any type of object.

More to come in CSE 123 on the Java mechanisms that make this work!

We can use the `instanceof` keyword in Java to determine if the parameter is actually a `ShoppingCart`

ShoppingCart's equals()

Hunter Schafer 1 minute ago

I also think it would be good to highlight the fact that every Java programmer and their mother just copies (or has memorized) that equals method template and the "real work" is filling in the middle case

👍 1

Almost there…

This is actually **still an imperfect implementation** because we would also need to write a `hashCode()` method for our object to work with `HashSet`, `HashMap`, etc. but more to come on that in CSE 331 and beyond ☺

Lecture Outline (4/4)

- Announcements
- Recap
- Equals
- **Bigger Example**

Student class

Write a `Student` class that you can construct by saying:

```
new Student(1234567, "Cornbear")
```

where the first parameter is their student number and the second parameter is their name. Your `Student` class should also implement the following methods:

- `getName()` returns the student's name
- `getStudentNumber()` returns the student's number
- `setName(String newName)` sets the student's name to the given `newname`
- `toString()` returns a `String` representation of the student formatted as `"name (studentNumber)"`
- `equals(Object other)` that returns `true` if the given parameter is considered equal to this object

Student class

A cautionary tale:

What if we added a field to the Student class:

```
private boolean isLocalStudent;
```

Yikes—You are the **designer** now. Think carefully about what assumptions you are making!

Also…

Why shouldn't we include a `setStudentNumber` method?

Course class

Write a Course class that represents a course at UW. Implement the following methods and constructors:

Constructors

- Write a constructor so that you can construct a Course by saying `new Course(23213, "CSE 122", 4)` where the first parameter is the course's SLN, the second parameter is the code for the course, and the third parameter is the number of credits.
- Write another constructor so that you can construct a Course by saying `new Course(23239, "CSE 122", 4, enrollment)` where the first parameter is the course's SLN, the second parameter is the code for the course, the third parameter is the number of credits, and the fourth parameter is a `Student[]` containing a `Student` for each student enrolled in the course.

Course class

Instance Methods

- `getSLN()` returns the course's SLN
- `getCourseCode()` returns the course's code
- `getCredits()` returns the number of credits for the course
- `getRoster()` returns a copy of the course's roster

Course class

Instance Methods

- `updateRoster(Student[] students)` replaces the current roster with the content of the given students
- `addStudent(Student s)` adds the given student to the roster if they are not already on it
- `dropStudent(Student s)` removes the given student from the roster if they are on it
- `checkStudentEnrolled(Student s)` returns true if the given student is on the current roster, and false otherwise