LEC 14

CSE 122

Interfaces

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Talk to your neighbors:

How do you feel about overalls?
Comfy af? Fashion faux pas?

Music: Ateo – C. Tanaga, Nathy Peluso

Instructor Elba Garza

TAs Abigail, Autumn, Claire, Jacob, Kevin, Mia, Rucha, Shreya, Ambika, Ayush, Colin, Jasmine, Kyle, Poojitha, Saivi, Smriti, Arthur, Chaafen, Elizabeth, Jaylyn, Marcus, Rishi, Shananda, Steven, Atharva, Chloë, Helena, Kavya, Megana, Rohini, Shivani, Zane

Lecture Outline

- Announcements
- Interfaces Review
- More Shapes!
- Comparable

Announcements

- Creative Project 2 (C2) due Thursday, November 16th
- Resubmission Cycle 5 (R5) out Thursday, November 16th
- Programming Assignment 3 (P3) out soon!
  - Due November 27th by 11:59 PM
  - Note IPL will be **very limited** next week; plan accordingly
- Quiz 2 **delayed** to November 28th
  - No quiz section on November 21st! 
- Reminder on next class, November 17th:
  - No in-person class; only recording!
  - Elba Office hours cancelled! (Need to talk? email me!)
- Reminder on Final Exam: **Tuesday, December 12th 12:30 – 2:20 PM**

Lecture Outline

- Announcements
- **Interfaces Review**
- More Shapes!
- Comparable

Recall from L4: Wait, ADT? Interfaces?

- **Abstract Data Type (ADT)**: A *description of the idea* of a data structure including what operations are available on it and how those operations should behave. For example, the English explanation of what a list should be.

- **Interface**: Java construct that lets programmers *specify what methods a class should have*. For example the `List` interface in java.

- **Implementation:** *Concrete code* that meets the specified interface. For example, the `ArrayList` and `LinkedList` classes that implement the `List` interface.

Interfaces

Interfaces serve as a sort of "contract" – in order for a class to <u>implement</u> an interface, it must fulfill the contract.

The contract's requirements are certain methods that the class must implement.

`List` Interface

`List` is an interface – its contract includes methods like:

`add`, `clear`, `contains`, `get`, `isEmpty`, `size`

So any classes that implement the `List` interface <u>must</u> include <u>all</u> these methods (and any others the `List` interface specifies)

Interfaces vs. Implementation

Interfaces require certain methods, but they do not say anything about how those methods should be implemented – that's up to the class! 🏅

`List` is an interface

- `ArrayList` is a class that implements the `List` interface
- `LinkedList` is a class that implements the `List` interface
- …

Why interfaces?

Flexibility

```
public static void method(Set<String> s) {…}
```

This method can accept either a:

- `HashSet<String>` or
- `TreeSet<String>` or
- Any other class that implements `Set` and whose element type is `String`!

Why interfaces?

Abstraction

Interfaces also support *abstraction*
(the separation of ideas from details)

Lecture Outline

- Announcements
- Interfaces Review
- **More Shapes!**
- Comparable

Classes can Implement Multiple Interfaces

A class can implement multiple interfaces – it's like one person signing multiple contracts!

If a class implements an interface A <u>and</u> an interface B, it'll just have to include all of A's required methods along with all of B's required methods

Classes can Implement Multiple Interfaces

```
public interface Company {
    public String getName();

    public String getMissionStatement();
}


public class Square implements Shape, Company {
    ...
}
```

But `Square` would have to implement:

- `getPerimeter`, `getArea` from `Shape`

  AND

- `getName`, `getMissionStatement` from `Company`

An interface can `extend` another

You can have one interface extend another

So if `public interface A extends B`, then any class that implements A must include all the methods in A's interface and all the methods in B's interface

An interface can extend another

We can write another interface

`Polygon` that extends `Shape`

`Make modifications such that:`

- `Square` is a `Polygon` (and `Shape`)
- `Triangle` is a `Polygon` (and `Shape`)
- `Circle` is a `Shape` (but *not* a `Polygon`)

Lecture Outline

- Announcements
- Interfaces Review
- More Shapes!
- **Comparable**

Recall the Student / Course Example from Wed

Course stored a field

```
private List<Student> roster;
```

Why not use a Set to store the students?…

Seems like a great idea (no duplicates, not worried about keeping a specific order or indexing into it) but … Java reasons:

- HashSet won't work because of lack of hashCode() implementation
- TreeSet won't work because what does it mean to "sort" Students

Comparable

`TreeSet` uses an **interface** called `Comparable<E>` to know how to sort its elements!

Only has <u>one</u> required method:

```
public int compareTo(E other)
```

Its return value is:

- `< 0` if `this` is "less than" `other`
- `0` if `this` is <u>equal</u> to `other`
- `> 0` if `this` is "greater than" `other`

Practice : Think

sli.do #cse122

Select all of the following statements that would cause an error.

```
public interface A {
    public void a();
}

public interface B extends A {
    public void b();
}

public interface C {
    public void c();
}

public interface D extends A {
    public void d();

    public void e();
}
```

```
public class One implements A {
    ...
}

public class Two implements B, D {
    ...
}

public class Three implements B, C {
    ...
}
```

A)
```
B w = new Two();
w.b();
```

B)
```
B x = new Two();
x.e();
```

C)
```
D y = new Three();
y.b();
```

D)
```
C z = new Three();
z.c();
```

Practice : Pair

sli.do #cse122

Select all of the following statements that would cause an error.

```
public interface A {
    public void a();
}

public interface B extends A {
    public void b();
}

public interface C {
    public void c();
}

public interface D extends A {
    public void d();

    public void e();
}
```

```
public class One implements A {
    ...
}

public class Two implements B, D {
    ...
}

public class Three implements B, C {
    ...
}
```

A)
```
B w = new Two();
w.b();
```

B)
```
B x = new Two();
x.e();
```

C)
```
D y = new Three();
y.b();
```

D)
```
C z = new Three();
z.c();
```