LEC 13

CSE 122

Interfaces

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Talk to your neighbors:

What is your most-used emoji?

Music: Hunter/Miya's Playlist

Instructor Hunter Schafer / **Miya Natsuhara**

TAs

Ajay	Gaurav	Melissa
Andrew	Hilal	Noa
Anson	Hitesh	Parker
Anthony	Jake	Poojitha
Audrey	Jin	Samuel
Chloe	Joe	Sara
Colton	Joe	Simon
Connor	Karen	Sravani
Elizabeth	Kyler	Tan
Evelyn	Leon	Vivek

Lecture Outline

- Announcements
- Interfaces Review
- More Shapes!
- Comparable

Announcements

- Thanksgiving week (11/21 - 11/25)
  - No Quiz Retakes on Tues 11/22
  - No Quiz Sections on Tues 11/22
  - No class on Wed 11/23
  - Thurs, Fri (11/24, 11/25) are University Holidays
  - Limited hours at the IPL
- P3 will be released on Fri (11/18) but will not be due until 12/1
- Reminder that the final exam is scheduled for **Tuesday (Dec 13) 12:30pm-2:30pm**

Lecture Outline

- Announcements
- Interfaces Review
- More Shapes!
- Comparable

(PCM) Interfaces

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

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

(PCM) `List` Interface

`List` is an interface – its contract includes method like

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

So any classes that implement the `List` interface must include all of these methods (and any others the `List` interface specifies)

(PCM) 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
- …

(PCM) Why interfaces?

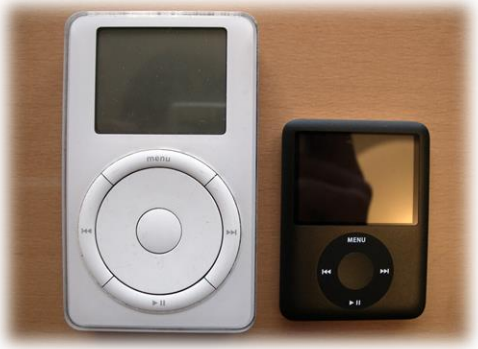
Flexibility

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

This method can accept *either* a `HashSet<String>` *or* a `TreeSet<String>` or any other class that implements `Set` and whose element type is `String`!

(PCM) 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 *and* 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 of the methods in A's interface *and* all of the methods in B's interface

An interface can `extend` another

We can write another interface

`Polygon` that extends `Shape`

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

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();
```

Lecture Outline

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

Recall the Student / Course Example from Fri

Course stored a field

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

We also had a suggestion to 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 the `hashCode()` business I mentioned on Fri
- `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 one required method:

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

Its return value is:

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