Questions during Class?
Raise hand or send here
sli.do    #cse122

BEFORE WE START

Talk to your neighbors:

What are your plans over the summer?

Instructors:  Ido Avnon
TAs:          Abby Williams
             Chloë Mi Cartier
             Connor Sun
             Cynthia Pan
             Katharine Zhang
             Marcus Sanches
             Rohini Arangam
Lecture Outline

• Announcements
• Constructors Recap
• Equals
• Bigger Example
• Interfaces Review
• Shapes!
• Comparable
Announcements

• Programming Assignment 3 (P3) releasing later tonight
  - Focused on OOP and interfaces!
• Quiz 2 (LAST ONE) in section Thursday 8/8
• Finals week coming up! Prep materials coming next week
Lecture Outline

• Announcements

• **Equals**

• Bigger Example

• Interfaces Review

• Shapes!

• Comparable
Equals (PCM Review)

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.

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.
public boolean equals(Object other) {
    if (other == this) {
        return true;
    } else if (other instanceof MyObject) {
        MyObject otherMyObject = (MyObject) other;
        return /* TODO */;
    } else {
        return false;
    }
}

\[ y = x \]
\[ 4 \leq j \leq 6.9 \]
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 Point.
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

• Announcements

• Equals

• Bigger Example

• Interfaces Review

• Shapes!

• Comparable
Student class

Write a Student class that you can construct by saying:

```java
new Student(1234567, "Miya")
```

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

What if we added a field to the Student class:

private boolean isMale;

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

- `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
- `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
Lecture Outline

• Announcements
• Equals
• Bigger Example
• Interfaces Review
• Shapes!
• Comparable
Recall from L6: 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 “certificate”—in order for a class to implement an interface, it must fulfill the certificates requirements.

The certificates requirements are certain methods that the class must implement.
Lists

One ADT we’ve talked a lot about in this course is a list.

Within Java, there exists a List interface – its contract includes methods like:

- add
- clear
- contains
- get
- isEmpty
- size

There’s also an ArrayList class (implementation)

To get the certificate, it must include all 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

• Equals

• Bigger Example

• Interfaces Review

• Shapes!

• Comparable
Classes can Implement Multiple Interfaces

A class can implement multiple interfaces – it’s like one person getting multiple certificates!

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

```java
public interface Parallel {
    public int numParallelPairs();
}

public class Square implements Shape, Parallel {
    ...
    public int numParallelPairs() {
        return 2;
    }
}

But Square would have to implement:
- `getPerimeter`, `getArea` from Shape
  AND
- `numParallelPairs` from Parallel
```

Shape sq = new Square();
Parallel sq = new Square();
```
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

\textbf{Polygon} that extends \textbf{Shape}

Make modifications such that:

- Square is a \textbf{Polygon} (and \textbf{Shape})
- Triangle is a \textbf{Polygon} (and \textbf{Shape})
- Circle is a \textbf{Shape} (but \textit{not} a \textbf{Polygon})
Lecture Outline

• Announcements
• Equals
• Bigger Example
• Interfaces Review
• More Shapes!

• Comparable
Comparable

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

Only has one required method:

```java
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