Before we start

Talk to your neighbors:
What did you get up to during the long weekend? Anything other than sleeping?

Music: 122 24sp Lecture Tunes 🌼

Instructors
Miya Natsuhara and Kasey Champion

TAs
Ayush
Poojitha
Chloe
Ailsa
Jasmine
Lucas
Logan

Kyle
Jacob
Atharva
Rucha
Megana
Eesha
Zane

Colin
Ronald
Saivi
Shivani
Kavya
Steven
Ken

Chaafen
Smriti
Ambika
Elizabeth
Aishah
Minh
Katharine
Lecture Outline

• Announcements

• Interfaces Review

• More Shapes!

• Comparable
Announcements

• Creative Project 2 (C2) due Thursday, May 16th
• Resubmission Cycle 5 (R5) out Thursday, May 16th
• Programming Assignment 3 (P3) out soon!
  - Due May 23rd by 11:59 PM
• Quiz 2 Tuesday, May 21st
• Reminder on Final Exam: Thursday, June 6\textsuperscript{th} 8:30-10:20AM
Lecture Outline

• Announcements

• Interfaces Review

• More 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
• Interfaces Review
• More 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`
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
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
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