Recap

- UML class diagrams
  - Inheritance, aggregation, and composition

- OO design principles
  - Composition over inheritance
  - Open/closed principle
  - Liskov substitution principle
Recap: aggregation vs. composition

```java
public class Bank {
    ...
    public void newCustomer(Customer c) {
        ...
    }
    ...
}

public class Stack<E> {
    private List<E> l = new LinkedList<>();
    public void push(E elem) {
        ...
    }
    ...
}

Bank b = new Bank();
Customer c = new Customer();
b.newCustomer(c);
...

Stack<Integer> s = new Stack<>();
s.push(...);
...
Recap: composition over inheritance

public class Stack<E> {
    extends List<E> {
        ...
    }
}

public class Stack<E> {
    private List<E> l = new LinkedList<>();
    ...
}
Today

- Design problems & potential solutions
- Design patterns
  - What is a design pattern?
  - Categories of design patterns.
A first design problem

Weather station revisited

<table>
<thead>
<tr>
<th>Current</th>
<th>30 day history</th>
</tr>
</thead>
<tbody>
<tr>
<td>25° F</td>
<td></td>
</tr>
<tr>
<td>-3.9° C</td>
<td>min: 20° F</td>
</tr>
<tr>
<td></td>
<td>max: 35° F</td>
</tr>
</tbody>
</table>

Temp. sensor

Reset history button

Reset
A first design problem

Weather station revisited

<table>
<thead>
<tr>
<th>Current</th>
<th>30 day history</th>
</tr>
</thead>
<tbody>
<tr>
<td>25° F</td>
<td></td>
</tr>
<tr>
<td>-3.9° C</td>
<td>min: 20° F</td>
</tr>
<tr>
<td></td>
<td>max: 35° F</td>
</tr>
</tbody>
</table>

We focus on the view -> what would be a good design?
Weather station: view

<<interface>>
View
+ draw(d:Data)

SimpleView
+ draw(d:Data)

GraphView
+ draw(d:Data)

...View
+ draw(d:Data)

ComplexView
+ draw(d:Data)
+ addView(v:View)
Weather station: view

public void draw(Data d) {
    for (View v : allViews) {
        v.draw(d);
    }
}

Design pattern: Composite

Component

+ operation()

CompA

+ operation()

CompB

+ operation()

Composite

+ operation()
+ addComp(c:Component)
+ removeComp(c:Component)
Design pattern: Composite

<<interface>>

Component

+ operation()

CompA

+ operation()

CompB

+ operation()

Composite

+ operation()
+ addComp(c:Component)
+ removeComp(c:Component)
What is a design pattern?

- Addresses a recurring, common design problem.
- Provides a generalizable solution.
- Provides a common terminology.
What is a design pattern?

- Addresses a recurring, common design problem.
- Provides a generalizable solution.
- Provides a common terminology.

Pros
- Improves communication and documentation
- “toolbox” for novice developers

Cons
- Risk of over-engineering
- Potential impact on system performance

More than just a name for common sense and best practices.
Design patterns: categories

1. Creational

2. Structural

3. Behavioral
Design patterns: categories

1. Creational
   - Singleton
   - Factory (method)
   - ...

2. Structural
   - Composite
   - Decorator
   - ...

3. Behavioral
   - Template method
   - Visitor
   - ...
Design patterns: categories

1. Creational
   - Singleton
   - Factory (method)
   - ...

2. Structural
   - Composite
   - Decorator
   - ...

3. Behavioral
   - Template method
   - Visitor
   - ...
Another design problem: I/O streams

```
<<interface>>
InputStream
+ read():int
```

FileInputStream
+ read():int

URLInputStream
+ read():int

BufferedInputStream
+ read():int
Design pattern: Decorator

<<interface>>
Component
+ operation()

CompA
+ operation()

CompB
+ operation()

Decorator
+ operation()
Decorator vs. composite

**Component**
- + operation()

**Decorator**
- + operation()

**CompA**
- + operation()

**Composite**
- + operation()
- + addComp(c:Component)
- + removeComp(c:Component)
Decorator vs. composite

- **Component**
  - + operation()

- **Composite**
  - + operation()
  - + addComp(c:Component)
  - + removeComp(c:Component)

- **Decorator**
  - + operation()

- **CompA**
  - + operation()