Recap

- UML class diagrams
  - Inheritance, aggregation, and composition

- OO design principles
  - Composition over inheritance
  - Open/closed principle
  - Liskov substitution principle

Recap: aggregation vs. composition

Customer
Bank
LinkedList
Stack

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

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

Recap: composition over inheritance

LinkedList
Stack

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

### Recap: aggregation vs. composition

```java
public class Bank {
  ...
  public void newCustomer(Customer c) {
    ...
  }
  ...
}
Customer c = new Customer();
b.newCustomer(c);
...
...
Stack<Integer> s = new Stack<>();
s.push(...);
...
Today

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

A first design problem

Weather station revisited

Current | 30 day history
---|---
25° F |  
-3.9° C | min: 20° F max: 35° F

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

```
+ draw(d:Data)
```

```
SimpleView
+ draw(d:Data)
```

```
GraphView
+ draw(d:Data)
```

```
...View
+ draw(d:Data)
```

```
ComplexView
+ draw(d:Data)
+ addView(v:View)
```

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

Design pattern: Composite

```
+ operation()
+ addComp(c:Component)
+ removeComp(c:Component)
```

```
CompA
+ operation()
```

```
CompB
+ operation()
```

```
Composite
+ operation()
```

```
+ 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

### Creational
1. Singleton
2. Factory (method)

### Structural
1. Composite
2. Decorator

### Behavioral
1. Template method
2. Visitor

---

[Design patterns: categories]

<table>
<thead>
<tr>
<th>Category</th>
<th>Subcategories</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Creational</td>
<td>Singleton, Factory (method)</td>
</tr>
<tr>
<td>2. Structural</td>
<td>Composite, Decorator</td>
</tr>
<tr>
<td>3. Behavioral</td>
<td>Template method, Visitor</td>
</tr>
</tbody>
</table>
Another design problem: I/O streams

<<interface>>
InputStream
+ read():int
FileInputStream
+ read():int
1
... URLInputStream
+ read():int
BufferedInputStream
+ read():int

Design pattern: Decorator

<<interface>>
Component
+ operation()
CompA CompB
+ operation()
+ operation()
+ operation()
1

Decorator vs. composite

<<interface>>
Component
+ operation()
CompA
+ operation()
+ operation()
1
Composite
+ operation()
+ addComp(c:Component)
+ removeComp(c:Component)
1..n

Decorator vs. composite

<<interface>>
Component
+ operation()
CompA
+ operation()
+ operation()
1
Composite
+ operation()
+ addComp(c:Component)
+ removeComp(c:Component)
1..n