CS 520
Theory and Practice of Software Engineering
Fall 2017

OO design patterns

September 26, 2017
Today

- MAP.
- Recap: OO design principles.
- Design problems & potential solutions.
- Design patterns:
  - What is a design pattern?
  - Categories of design patterns.
  - Structural design patterns.
Recap

**OO design principles**
- Information hiding (and encapsulation)
- Open/closed principle
- Liskov substitution principle
- Composition/aggregation over inheritance
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

Reset history button
What’s a good design for the view?

Client sees uses manipulates

- 25° F
- -3.9° C
  - min: 20° F
  - max: 35° F

09/01, 12°
09/02, 14°
...

Temp. sensor

Reset button

Reset history
Weather station: view

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

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

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

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

```
ComplexView
-views:List<View>
+draw(d:Data)
+addView(v:View)
```

**How do we need to implement**
**draw(d:Data)?**

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25° F</td>
<td>-3.9° C</td>
</tr>
<tr>
<td>min: 20° F</td>
<td>max: 35° F</td>
</tr>
</tbody>
</table>
Weather station: view

```
public void draw(Data d) {
    for (View v : views) {
        v.draw(d);
    }
}
```
Design pattern: Composite

Component

+operation()

CompA

+operation()

CompB

+operation()

Composite

-comps:Collection<Component>

+operation()

+addComp(c:Component)

+removeComp(c:Component)
Iterate over all composed components (`comps`), call `operation()` on each, and potentially aggregate the results.
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. Structural
   - Composite
   - Decorator
   - ...

2. Behavioral
   - Template method
   - Visitor
   - ...

3. Creational
   - Singleton
   - Factory (method)
   - ...
Design patterns: categories

1. Structural
   ● Composite
   ● Decorator
   ● ...

2. Behavioral
   ● Template method
   ● Visitor
   ● ...

3. Creational
   ● Singleton
   ● Factory (method)
   ● ...
Another design problem: I/O streams

```
InputStream is = new FileInputStream(...);

int b;
while((b=is.read()) != -1) {
    // do something
}
```

```
<<interface>>
InputStream

+read():int
+read(buf:byte[]):int
```
Another design problem: I/O streams

Problem: filesystem I/O is expensive

InputStream is =
    new FileInputStream(...);

int b;
while((b=is.read()) != -1) {
    // do something
}

FileInputStream
+read():int
+read(buf:byte[]):int

<<interface>>
InputStream
+read():int
+read(buf:byte[]):int
Another design problem: I/O streams

Problem: filesystem I/O is expensive

Solution: use a buffer!

Why not simply implement the buffering in the client or subclass?

```java
InputStream is = new FileInputStream(...);
int b;
while((b=is.read()) != -1) {
    // do something
}
...
```
Another design problem: I/O streams

```java
InputStream is = new BufferedInputStream(new FileInputStream(...));
int b;
while((b = is.read()) != -1) {
  // do something
}
```

Still returns one byte (int) at a time, but from its buffer, which is filled by calling `read(buf:byte[])`. 
Design pattern: Decorator

```
<<interface>>
Component
+operation()

Decorator
-decorated: Component
+Decorator(d: Component)
+operation()
```

CompA
+operation()

CompB
+operation()
Composite vs. Decorator

<<interface>>
Component
+operation()

Composite
-comps:Collection<Component>
+operation()
+addComp(c:Component)
+removeComp(c:Component)

CompA
+operation()

Decorator
-decorated:Component
+Decorator(d:Component)
+operation()