CS 520
Theory and Practice of Software Engineering
Fall 2022

Collaborative development

October 4, 2022
Today

- Recap: Design patterns

- Collaborative development:
  - Agile development
  - Scrum
  - Pair programming
  - Collaborative development exercise
Recap: Design patterns

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

1. Structural
   - Composite
   - Decorator
   - ...

2. Behavioral
   - Observer
   - Strategy
   - Template (method)
   - ...

3. Creational
   - Factory (method)
   - ...

Design pattern: Decorator

<<interface>>
Component
+operation()

1

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

java.io.Reader
java.io.FileReader
java.io.BufferedReader

CompA
+operation()
read(): int
java.io.FileReader

CompB
+operation()
read(): int

java.io.Reader
java.io.BufferedReader

Readers
java.io.FileReader
java.io.BufferedReader
Design pattern: Template method

- The template method (median) implements the algorithm but leaves the sorting of the array undefined.

- The concrete subclass only needs to implement the actual sorting.
“median” delegates the sorting of the array to a “sortStrategy”
Template method vs. strategy

Two solutions to the same problem

Template method
- Behavior selected at compile time.
- Template method is usually final.

Strategy
- Behavior selected at runtime.
- Composition/aggregation over inheritance.
Design pattern example 1

Here is the Java Arrays utility class:
https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/Arrays.html

We’ll focus on the following method:
public static <T> void sort(T[] a, Comparator<? super T> c)

This method takes as input a Comparator class. Which design pattern is being applied?
● Composite
● Decorator
● Strategy
Design pattern example 2

Here is the Java Collections utility class:
https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/Collections.html

We’ll focus on the following method:
\[
\text{public static } <T> \text{ List}<T> \text{ unmodifiableList(List}\langle? \text{ extends } T\rangle \text{ list)}
\]

For this method, which design pattern is being applied?

- Composite
- Decorator
- Strategy
Design pattern example 3

Here is the Java AbstractList class:
https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/AbstractList.html

Here is the ArrayList class:
https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/ArrayList.html

Which design pattern is being applied?
• Decorator
• Strategy
• Template (method)
Agile development

• Fast paced
• Frequent releases
• Developer centered
  ─ Do we need managers?
Scrum

• A very popular flavor of Agile to rapidly iterate in Sprints
  – Each Sprint develops then releases the product

• Three pillars:
  – Transparency
  – Inspection
  – Adaptation

• Used by large tech companies such as Facebook, Google, Microsoft

https://www.scrum.org
Three roles

• Product owner
  – represents the customer specifying the goal

• Development team
  – Performs Sprints
  – delivers software product that satisfies that goal

• Scrum master
  – buffer between team and outside world
  – prevents distractions, barriers
Many aspects of Scrum

• Sprints
• Stand-up meetings
  – What did I do yesterday?
  – What will I do today?
  – Do I see any impediment from our goal?
• Reviews
Pair programming

• Requirements specification, designing, implementing, testing, etc.

• Pair-work facilitates
  – transparency
  – no single point of failure
  – decision making
  – focus
  – creativity
Collaborative development exercise

• Further develop a Figure editor available here: https://github.com/LASER-UMASS/cs520-Spring2020.git

• Form pairs that will collaboratively work on:
  – documentation
  – design
  – implementation
Figure editor
Figure editor (v1): MVC architecture

- **Client** sees View and uses Controller.
- **View** updates Model.
- **Controller** manipulates Model.

Separates data representation (Model), visualization (View), and client interaction (Controller).
**Figure editor (v1): Model API**

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.lang.String</code></td>
<td><code>getCaption()</code></td>
</tr>
<tr>
<td><code>javax.swing.ImageIcon</code></td>
<td><code>getImage()</code></td>
</tr>
<tr>
<td><code>boolean</code></td>
<td><code>isComplete()</code></td>
</tr>
<tr>
<td></td>
<td>Returns true if this figure is complete, meaning its Image is non-null and its caption is non-null and non-empty, and false otherwise.</td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>setCaption(java.lang.String newCaption)</code></td>
</tr>
<tr>
<td></td>
<td>Sets the caption to the given non-null and non-empty String.</td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>setImage(javax.swing.ImageIcon newImage)</code></td>
</tr>
<tr>
<td></td>
<td>Sets the image to the given non-null ImageIcon.</td>
</tr>
</tbody>
</table>
Model (v1): Implementation

- Use the Model API for the implementation

```
public void setImage(javax.swing.ImageIcon newImage)
Sets the image to the given non-null ImageIcon.
Parameters:
newImage - The ImageIcon must be non-null
Throws:
java.lang.IllegalArgumentException - if the ImageIcon is null
```
Figure editor (v2): MVC architecture

Separates data representation (Model), visualization (View), and client interaction (Controller)

Apply Observer pattern
Figure editor (v1): Observer pattern

Observable
{abstract}
- # observers:Set<Observer>
- + register(o:Observer)
  + unregister(o:Observer)
  + stateChanged()

MyObservable
- state:State
  + getState():State
  + setState(state:State)

<<interface>>
Observer
- + update()

MyObserver
+ update()
Figure editor (v1): Observer pattern

Observable

- # observers:Set<Observer>
- + register(o:Observer)
- + unregister(o:Observer)
- + stateChanged()

<<interface>>

Observer

- + update()

MyObservable

- state:State
- + getState():State
- + setState(state:State)

MyObserver

+ update()
Java provides the following two classes:

- [https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeSupport.html](https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeSupport.html)
- [https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeListener.html](https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeListener.html)

How could the Observer design pattern be implemented using these classes?
Topics covered

• Documentation, e.g.,
  – README, javadoc, internal comments

• Architecture & design, e.g.,
  – Patterns (MVC, Observer)
  – Class diagrams

• Implementation
  – Pair programming
  – Java, AWT, Swing
Week 5 Participation Questionnaire

Submit your collaborative exercise:
• documentation
• design
• implementation