Design Patterns

University of Alberta

Peer-graded Assignment: Capstone Assignment 2.3 – Identify and Fix Code Smells

Instructions | My submission | Discussions

Now that we've built an application that implements the Command Pattern and MVC Pattern, it's time to evaluate the code.

In the first part of this assignment you will be challenged to identify Code Smells in the given code base and brainstorm ways to fix them.

In the second part of this assignment you are going to fix the Long Method code smell.

How to create your assignment

Part 1

Use the given code base to identify two code smells that are described in the Anti Patterns & Code Smells lecture -- **excluding Long Method**.

For each of the two code smells you have identified:

- Document them by clearly stating where the code smell is (which class, which method, etc.)
- Which code smell from the lecture it is, and why you think the code fits the description of the code smell.
- Give your reasoning on why each code smell is a problem and should be fixed.
- Suggest a solution to fix the code smell.

Part 1 Notes:

- If the same piece of code has more than one code smell in it, all code smells will count, as long as each has their own code smell type and solution. However, identifying the same code smell twice in different parts of the code will not count.
- We do not want you to actually fix these two code smells!
- You should only have to write one paragraph per code smell and submit no more than 600 words for your answer.
- If you would like you may use UML diagrams alongside your explanation to illustrate how something is a smell or how it should be fixed.

Part 2

In this second part of the assignment, the code smell you will be fixing is Long Method.

To fix this code smell, you will Implement a validateInput() method in the following activities:

- AddItemActivity
- EditItemActivity
- AddContactActivity
- EditContactActivity

These four activities all contain "save" methods that consist of several lines of code for validating user input. The lines of code responsible for validating the input should be moved to the validateInput() method. This new method can then be called from inside the save method.

Part 2 Hints:

1. Replace multiple lines of input validation in the "save" method with the following lines:

```
If ( !validateInput() ) {
  return;
}
```

2. Create and implement the validateInput() method with the following signature:

```
public boolean validateInput() {
 // Input validation goes here...
 // ...
}
```

3. You will have to modify the scope of some of variables in each activity to implement this. For example, in EditItemActivity you will have to modify the scope of title_str, maker_str, description_str, length_str, width_str, height_str, and contact.

Lastly, don't forget -- the overall functionality of the application should not change!

Submission Instructions

Part 1: Upload a PDF of your solution where prompted.

Part 2: Include the following files in a folder:

- AddItemActivity.java
- EditItemActivity.java
- AddContactActivity.java, and
- EditContactActivity.java

Compress the folder into a ZIP folder. Windows users can use 7zip or WinRAR. Upload it where prompted.

Prepared by: Ng Wan Ying
Date: 27/09/2022

Part 1

Comments

The “comments” code smell described in the Anti-Patterns & Code Smells lecture has been identified in the ContactController class. This code smell can occur between two extremes and in the context of the ContactController class, no comments are provided in the code. This makes it hard for someone else, or even the original developer, to return to understand what the code is doing or should be doing after some time has passed. A solution to this code smell is to include comments documenting the methods in the ContactController class so that developers can understand what it does without having to read the details of the implementation.

Data Clumps

The “data clumps” code smell described in the Anti-Patterns & Code Smells lecture has been identified in the setDimensions method of the Item class. This code smell occurs when groups of data appear together in the instance variables of a class or parameters to methods. In the context of the setDimensions method of the Item class, the String variables, length, width, and height are used as parameters repeatedly when it is better to have an object as the parameter. Therefore, to make better code and abstraction, a solution to this code smell is to introduce a parameter object for the setDimensions method by replacing the String variables with the existing Dimensions object.

Part 2

Please refer to the following files:

- AddItemActivity.java
- EditItemActivity.java
- AddContactActivity.java
- EditContactActivity.java