CSCI 5308 Adv Topics in Software Development

Assignment 2

Name: Jay Sanjaybhai Patel
CSID: jspatel
Banneer ID: B00982253
Gitlab Link: https://git.cs.dal.ca/courses/2024-summer/csci-5308/assignment2/jspatel.git

Table of Contents

Single Responsibility Principle

1.1 Bad

1.1.1 Overview

This program is an authentication service featuring user login, logout, and registration functionalities. It includes validation checks for email format, PIN validity, and user existence, logging actions to a file, and managing user login states.

1.1.2 Description of classes

Validation Class:

- Handles validation tasks such as checking email format (isEmailValid), PIN validity (isPinValid), and user existence (isEmailAlreadyExist, isPINAlreadyExist, isUserValid) in a collection.

AuthService Class:

- Manages user authentication operations (logIn, logOut, register) using the Validation class for input validation.
- Maintains a list of authenticated users (currentLoginUsers) and logs authentication activities (log) to a file.

DriverClass:

- Contains the main method to demonstrate and test the functionalities of AuthService.

1.1.3 Explanation for Violation of SRP

- It combines user authentication logic (logIn, logOut, register) with logging (log method) and user state management (currentLoginUsers).
- This makes the class responsible for both authentication business logic and secondary concerns like logging and user state management.

1.2 Good

1.2.1 Description of classes

TxtLog Class (implements Log interface):

- Handles logging functionality (writeLog method) independently from AuthService.
- Uses the Log interface, ensuring that different logging implementations can be swapped without affecting AuthService.

Users Class:

- Manages user data (allUsers and currentLoginUsers) independently, following to the Single Responsibility Principle by focusing solely on data management.

1.2.2 Solution of SRP

- Each class (AuthService, TxtLog, Users, Validation) now follows SRP by focusing on a single aspect of functionality.
- AuthService manages authentication logic, TxtLog handles logging, Users manages user data, and Validation handles input validation.

Open/Closed Principle

2.1 Bad

2.1.1 Overview

The program manages inventory items related to stationery supplies. It includes classes for InventoryItem (individual inventory item details and operations), Stationary (collection of inventory items with operations like add, delete, update, and calculations), DriverClass (main program execution).

2.1.2 Description of classes

InventoryItem:

- Represents an individual item in the inventory with attributes like name, quantity, price, company name, and level.
- Provides methods for calculating inventory value, checking availability, and adjusting quantity.

Stationary:

- Manages a collection of InventoryItem objects.
- Provides methods to add, delete, update, and retrieve items, calculate total inventory value and quantity, and print the inventory.

DriverClass:

- Creates inventory items like books and pens, performs operations such as adding, updating, and deleting items in the Stationary collection, and calculates inventory values.

2.1.3 Explanation for violation of OCP

- To add new types of inventory items (e.g., adding a new type of stationary item like a calculator), modifications are required in the existing Inventory class, rather than extending them.
- If we add item like Pencil we have to add some more attributes and methods for pencil in Inventory class, so it violating the principle of closed for modification but open for extension.

2.2 Good

2.2.1 Description of classes

InventoryItem (Abstract Class):

- Serves as the base class for all inventory items.
- Defines abstract methods (calculateInventoryValue(), isAvailable(), increasedQuantity(), decreasedQuantity()) that are implemented differently by each subclass (TenthStdBook, EleventhStdBook, TwelveStdBook, Pen).
- Contains common attributes (name, quantity, price) and methods to set/get these attributes.

TenthStdBook, EleventhStdBook, TwelveStdBook, Pen (Concrete Subclasses of InventoryItem):

- Each subclass represents a specific type of inventory item with its own attributes (companyName, level for books; companyName, type, color for pen).

2.2.2 Solution of OCP

- Introduced abstract class InventoryItem to define common behaviors and attributes shared among different types of inventory items.
- New types of inventory items can be added by creating a new subclass of InventoryItem and implementing its abstract methods, following to the principle of being open for extension but closed for modification.

Liskov Substitution Principle

3.1 Bad

3.1.1 Overview

The program shows government facilities, featuring a base class GovernmentFacility for general facility management tasks like staff and service management. It includes a specialized subclass Hospital that extends these functionalities with specific hospital-related operations such as managing bed capacity and handling patient services. The design showcases how different types of government facilities can be represented with varying operational specifics while sharing common management functionalities.

3.1.2 Description of classes

GovernmentFacility:

- Represents a generic government facility with operations like managing staff, serving people, and changing its active state.

Hospital:

- Extends GovernmentFacility with additional attributes like the number of beds.
- It overrides the servePeople method to handle patient capacity, and introduces specific methods like closeHospital for hospital-specific operations.

3.1.3 Explanation of violation of LSP

- Hospital, as a subclass of GovernmentFacility, changes the behavior of the servePeople method in a way that is not compatible with the behavior of its superclass.
- The overridden servePeople method in Hospital introduces a precondition (isActive) that differs from the behavior defined in GovernmentFacility.
- It throws an exception (UnsupportedOperationException) for a condition (numberOfPeople > numberOfBeds) that is not applicable to other types of government facilities.

3.2 Good

3.2.1 Description of classes

GovernmentFacility:

- An interface defining common operations for government facilities, including operate(), addStaff(), and removeStaff().

BedProvider:

- An interface extending GovernmentFacility that includes an additional method servePeople(int numberOfPeople), which is specific to facilities that provide beds for their services, such as hospitals.

Hospital:

- Implements GovernmentFacility and BedProvider, specializing in hospital-specific operations such as managing staff, serving patients based on bed capacity, and specific methods like setNumberOfBeds().

AddharCard:

- Implements GovernmentFacility, focusing on generic facility operations like managing staff and serving people.

3.2.2 Solution of LSP

- Introduce an interface BedProvider extending GovernmentFacility to segregate specific behaviors related to providing beds.
- Ensure that each subclass (Hospital, AddharCard) follows to the common contract defined by GovernmentFacility without introducing unexpected behaviors or exceptions that are not compatible with the superclass contract.

Interface Segregation Principle

4.1 Bad

4.1.1 Overview

This program shows a university system where students can register for courses, view and their grades, and drop courses. Professors can teach courses, view grades of their students, and grade assignments. It include DriverClass for testing interactions between them.

4.1.2 Description of classes

Student Class:

- Manages student details such as name and course enrollment.
- Supports operations like course registration, grade viewing, and course dropping.
- Implements methods from the University interface, some of which are non-functional or throw exceptions to follow to interface requirements but do not apply logically to student.

Professor Class:

- Manages professor details including assigned courses and students.
- Supports operations like teaching courses, viewing student grades, and grading assignments.
- Implements methods from the University interface, some of which are non-functional or throw exceptions to follow to interface requirements but do not apply logically to professor.

DriverClass:

- Serves as the entry point for the program, demonstrating interactions between student and professor.

4.1.3 Explanation for violation of ISP

- The University interface imposes methods that both Student and Professor classes must implement, regardless of their distinct functionalities. Methods like registerForCourse, and dropCourse are not applicable to professors, yet they are forced to implement these methods.

- Similarly grade assignments are not applicable to students. yet they implemented these methods.

4.2 Good

4.2.1 Description of classes

UniversityStudent Interface:
Define methods specific to student actions such as registerForCourse, viewGrades, and dropCourse.

UniversityProfessor Interface:
Define methods specific to professor actions such as teachCourse, viewGrades, and gradeAssignment.

Student Class:
Implement the UniversityStudent interface, which segregates student-specific functionalities.

Professor Class:
Implement the UniversityProfessor interface, which segregates professor-specific functionalities.

4.2.2 Solution of ISP

- Introduced two interfaces, UniversityStudent and UniversityProfessor, each focused on the specific needs of students and professors respectively, which avoid unnecessary methods that would force implementations to throw exceptions or be non-functional.

Dependency Inversion Principle

5.1 Bad

5.1.1 Overview

The program shows two kitchen appliances, a Toaster and a Microwave, each controlled by a Display and a HeatingElement. It demonstrates basic functionalities like setting timers, adjusting temperatures, and starting/stopping operations.

5.1.2 Description of classes

Display:

- Manages the user interface by displaying current temperatures and timers for the appliances.
- Allows setting and clearing timers, and displays warnings for high temperatures.

HeatingElement:

- Represents the heating component of the appliances. Controls heating up, cooling down, turning on/off, and setting temperatures.
- Ensures temperature safety checks and operations based on current state (on/off, heating/cooling).

Toaster and Microwave:

- Toaster and Microwave both are example of kitchen appliances which uses HeatingElement and Display to operate.

5.1.3 Explanation for violation of DIP

- Classes like Toaster and Microwave directly depend on concrete implementations (HeatingElement and Display), which are low-level components.
- Any changes to HeatingElement or Display require modifications in the Toaster and Microwave, making the system rigid and less adaptable to change.

5.2 Good

5.2.1 Description of classes

Display Interface:

- Defines methods for displaying timing and managing timers on a display interface.

DisplayImpl Class:

- Implements the Display interface, providing functionality to display the current temperature, manage timers, and show warnings for high temperatures.
- It collaborates with HeatingElement to obtain temperature data.

HeatingElement Interface:

- Defines methods for heating and cooling operations, setting and getting temperatures, and managing the on/off state of a heating element.

HeatingElementImpl Class:

- Implements the HeatingElement interface, providing concrete implementations for heating up, cooling down, setting temperature, and managing the on/off and cooling states of a heating element.

5.2.2 Solution of DIP

- Interfaces (Display and HeatingElement) are introduced to define contracts that high-level modules (Toaster and Microwave) depend on.
- Toaster and Microwave now depend on abstractions (Display and HeatingElement interfaces) rather than concrete implementations (DisplayImpl and HeatingElementImpl).
- If there's a need to change the implementation of Display or HeatingElement, such as adding new features or replacing them with different implementations, the changes can be made without affecting the high-level modules (Toaster, Microwave).