Design Rationale- Assignment 3

Architecture – MVC

The program was based on the Model-View-Controller (MVC) pattern. In this scenario, the MVC are the following

- Model - are the classes that define that data structure and manage the data and business logic.
    - For example, classes such as OnSiteBooking and HomeBooking are responsible for the business logic of managing the booking data
- View - this is where the information is displayed.
    - In our situation, this would be our frontend (HTML/JavaScript) which is presented to the user
- Controller - these are the classes that contain logic that updates the view in response to the input given off by the users.
    - For instance, all the RestController classes are responsible for returning the updated data back to the view to be displayed in the frontend

The reason for the use of MVC was because it best fitted the Java framework we implemented in SpringBoot. This resulted in our class diagram, for the view package to be empty. Because of the use of spring boot, it causes our View package to have no classes since everything in the View package is related to the frontend which helps us display the information to users. As shown by (Bodnar,2022) and (Anand , 2020) it can be seen that Sspringboot official view in the MVC is a html.

Advantages

- **Ease of testing** - MVC makes it easy to test due the separation of storing, displaying and and updating data. This allows it to be tested individually, making it easier to perform unit testing if needed
- **Ease of change** - MVC makes it much easier to change the frontend requirements as all the HTML pages are not dependent on the model classes, so when modifying or adding new html pages are done, it has no effect on the model
- **Single responsibility principle** - the MVC applies to the single responsibility principle due to separating the classes to their respected packages, it allows the classes to be simplified into one goal

Disadvantages

- **Complexity** - the html pages are more event-driven, causing it to be harder to debug from the extra levels of indirection
- **Breaks acyclic dependency (Refer to page 4)**

Requirements

Requirement 1- Users Modifying/ cancelling Bookings

To implement the ability for users to modify or cancel their bookings was not complicated due to our pre-existing booking abstract class. The sub classes which are under these include homeBooking,onSiteBooking and newly implemented phoneBooking. As such new methods were added into the abstract class being ‘cancelBooking’ and ‘modifyBooking’ which are simply inherited from the child classes.

Open/Closed Principle

The implementation of this requirement utilised the open/closed principle in which the abstract class is a method which is open to add features and functionality (new methods such as stated above) but closed for modification by the controller class.

Advantages

- The advantages of designing it like this include less repetition of code as subclasses inherit methods from the abstract class
- Much easier to add new features as shown through the addition of the modify and cancel and if any other needed feature in the future.
- Another reason it was designed like this was so that in the future there were more ways to book, another class like the newly implemented ‘phoneBooking’ can easily be added.

Disadvantages

- One of the disadvantages is that if by chance a new booking subclass is added and it wont need the ability to modify the booking for example it would not be as efficient to inherit it from the abstract class.

Requirement 2 - Receptionist Modify residents bookings from phone

To implement the requirement, I created two new classes in PhoneBookingController and PhoneBooking. The PhoneBooking is a child class of the abstract class Booking and the PhoneBookingController is an additional controller that was added for the implementation of this specific requirement.

Singleton

The singleton design pattern was reused for the PhoneBooking class. Singleton allowed us to have only one instance of an object created, making the PhoneBooking constructor private to ensure no other class creates the object. PhoneBooking was called from the Controller package through the PhoneBookingController where the instance can be get through the static method getInstance(), allowing us to call the methods in PhoneBooking.

Advantages

- Singleton improves the code connectivity and prevents the ability of other objects from instantiation their own copies of the PhoneBooking class
- Creating a global access point, allows it to share resources such as API data and avoid instantiation overhead. Not only does it save efficiency and not waste memory, it instead uses the one instance that we have already created

Disadvantages

- Singleton however violates the *Single Responsibility Principle*. This is because the PhoneBooking class should be able to control the number of instances it can create

- Singleton makes it very difficult to be maintained in the future especially during unit testing because of the creation of a single instance in a class. It introduces a global state to the application as you are providing a global access point to that instance to other classes such as the PhoneBookingController in this scenario
- This also promotes tight coupling between the model and controller. Relating back to the maintenance, this makes it the classes as well as the MVC pattern more complex, making it even more difficult to refactor and change

Requirement 3 - Receptionist Modify/Delete/Cancel residents booking from onsite
To implement the requirement, as discussed beforehand, along with creating the 2 new methods in the abstract class being ‘cancelBooking’ and ‘modifyBooking’, ‘deleteBooking’ was also created as well and inherited from the child classes.

Liskov Substitution Principle
The Liskov substitution principle was used with the abstract Booking class creating three new methods in ‘cancelBooking’, ‘modifyBooking’ and ‘deleteBooking’ where the subclasses would expand and show an extension of the parent class’s behaviour.

Advantages
- Liskov makes it easier to maintain and reduces coupling between classes. It makes it more reusable and has a lower degree of dependency

Disadvantages
- Relating back to the open close principle, a con is if modification is being made to the abstraction class Booking, it will be an expensive change to the module’s interface. As both the high level and low level classes are both dependent on the abstraction, there could be many changes to the existing code, increasing the chances of the existing system to break.

In the previous assignment, the design patterns we implemented were singleton, strategy and abstract factory method. During the refactoring and implementation of the MVC architecture, a facade method has been implemented.

Facade
The facade design pattern was used in our userFacadeController in the controller packages. The aim of this is to provide a simple interface which contains many aspects of a complex system of controllers.
Advantages
- We used this as it simplifies the interface and decouples from all the subsystem of components
- This also provided the ability for the isolation of specific code in one class

Disadvantages
- The main disadvantage of this is that it is currently a god class as many objects are coupled to that class

Package principles

Pros of design

Common Closure Principle

The common closure principle was demonstrated throughout our program. This was to ensure that our program was made in a way that changes in one package would not affect other components. An example of this would be the booking package has no relation to the on site testing package as they are separate. The pros of making the design this way is the maintainability of the system. This allows it to be more convenient and easier to debug in the future, allowing less cost and time spent refactoring the packaging and program. However, one of the cons would be for future implementation in which by using this principle the packages would become very large. As such changes in one component in the package will result in many changes in the whole package.

Release reuse equivalency

The release reuse equivalency principle was used throughout the program and the class diagram. Classes were assigned into different families depending on their relationship with the package. For example all booking classes such as the phoneBooking, homeBooking and oneSiteBooking are all in one booking package. Each of the classes shared its own single responsibility, illustrating effectively how the code in each package is not being reused and copied.

Cons of design

Acyclic Dependency Principle

One of the cons of using the MVC was that it breaks the acyclic dependency principle by breaking down the IO module into controller and view as seen in the class diagram. The implementation of this results in different packages being heavily dependent on one another which can cause many issues in terms of coupling.

Refactoring/Changes

Packages

Due to using The MVC model the packages needed to be changed to fit this. This included changing the original packages and adding a model,controller and view package and subsequently inside them are other packages which hold classes which are associated with the model, controller or view.

Renaming classes

Another minor change was renaming the onsiteTesting class to covidTesting to make it more clear of its association with the covidTestType class.

Abstract Booking class additional methods

The abstract class was refactored to add three new methods which are modifybooking,delete booking and cancel bookings. The reasoning behind this decision was discussed in the requirement section.

Extract Function Technique

From our previous assignment there were a lot of code fragments that could be grouped together. In order to reduce the length of some of our functions we used the extract function and created new methods to reduce code duplication as well. One example of this was the use of the ‘setJsonString()’ method and the ‘getRequestBody()’ which was pieces of code duplicated throughout our program. By adding these methods, every time these pieces of code were needed it was much much more efficient to call these functions.

Replace Temp with Query

Replace temp with a query, the use of an expression being moved to a separate method and returning it instead of tha variable. This was implemented and refactored in our current design through the use of checking ths html status code for each of our responses. Instead of using the code as a variable we pass it through a function call.

References

Bodnar, J., 2022. *Spring Boot model - using Model, ModelMap, and ModelAndView in Spring Boot*. [online] Zetcode.com. Available at: <https://zetcode.com/springboot/model/> [Accessed 26 May 2022].

Stack Overflow. 2020. *MVC - Spring Boot and Angular - What is the View in Spring?*. [online] Available at: <https://stackoverflow.com/questions/62497156/mvc-spring-boot-and-angular-what-is-the-view-in-spring>.