Weekend Fitness Club Software System

By (Name)

The Name of the Class (Course)

Professor (Tutor)

University of Hertfordshire

The City and State where it is located

**Introduction**

In this software development process, a software system was created for the Weekend Fitness Club (WFC). The WFCSoftware system implements the Java programming language and its object-oriented features to implement flexibility. During the design phase, some of the commonly used patterns such as the singleton pattern, state pattern, command pattern, and the Model-View-Controller Pattern (MVC) were used. Additionally, a UML class diagram was created to demonstrate the program flow, the composition, and the association between the Main class and the rest of the methods, types, and attributes. The code was also prepopulated with meaningful data for compilation and testing purposes.

From the code, it is evident that the main class of the project was included and it had 4 instance variables and three methods. When prepopulating our data the study made several assumptions. Firstly, the data is free from errors or missing values. Secondly, the input data was of the correct data type and lastly, the data had consistent formatting. The project also incorporated JUnit testing into our system. This was done to run unit tests on our code to ensure that the user does not get an unexpected output. The JUnit tests provided a good test case profile of the WFCSoftware class's functionality. Finally, we explain how a user is supposed to run the system on his computer either as a console app or an executable jar file.

**UML Class Diagram.**

**WFCSoftware**

lessonPrices: Map<String, Integer>

lessonRatings: Map<String, Integer>

lessonCustomers: Map<String, Integer>

bookedLessons: ArrayList<String>

main(args: String[]): void

printTimetable(): void

printReports(): void

**Customer**

bookings: ArrayList<FitnessLesson>

bookLesson(lesson: FitnessLesson): boolean

cancelBooking(lesson: FitnessLesson): boolean

| | +changeBooking(oldLesson: FitnessLesson, newLesson: FitnessLesson): boolean

**FitnessLesson**

name: String

price: int

capacity: int

**Booking**

lesson: FitnessLesson

customer: Customer

The UML Class diagram is a visual notation for building and visualizing systems with object-oriented architecture. A class diagram in the Unified Modeling Language (UML) is a form of a structural diagram that shows the classes of the system, their attributes, methods, and the connections within objects**.** Our UML class diagram for the WFCSoftware is illustrated above:

The WFCSoftware class has four instance variables:

* lesson prices: a Map that stores the fitness lessons and their prices
* lesson ratings: a Map that stores the fitness lessons and their ratings
* lesson Customers: a Map that stores the fitness lessons and the number of customers
* booked lessons: an ArrayList that stores the booked lessons

The WFCSoftware class has three methods:

* main(String[] args): the main method of the program that initializes the maps, prints the timetable, prompts the user to book and rate a lesson, and prints the reports
* printTimetable(): a method that prints the timetable
* printReports(): a method that prints the reports containing the number of customers per lesson on each of the 8 days, along with the average rating of each lesson and the type of fitness lessons which generated the highest income.

The FitnessLesson class represents a fitness lesson and has three instance variables:

* name: the name of the lesson
* price: the price of the lesson
* capacity: the maximum number of customers that can book the lesson

The Customer class represents a customer and has two instance variables:

* name: the name of the customer
* bookings: an ArrayList that stores the booked lessons of the customer

The Booking class represents a booking and has two instance variables:

* lesson: the fitness lesson that the customer has booked
* customer: the customer who has booked the fitness lesson

The associations between the classes are as follows:

* WFCSoftware has a composition relationship with FitnessLesson, Customer, and Booking.
* FitnessLesson has a composition relationship with Booking.
* The customer has an aggregation relationship with FitnessLesson.

**Assumptions**

This current research study presents an analysis of the assumptions employed during the software development process for the Java Project of WFCSoftware. These assumptions were implemented following the software project requirements. Based on the provided code, it is apparent that the preloaded data is arranged as a list of dictionaries, where each dictionary represents a single record, and the keys correspond to the column names. A crucial assumption is that the preloaded data is free from errors, including missing values, incorrect data types, or inconsistent formatting.

Moreover, it is anticipated that the data will remain static and unaltered during program execution. Although, if the data size exceeds the memory capacity, alternative approaches for data storage and retrieval may be necessary to avoid any potential performance issues. Lastly, it is assumed that the preloaded data accurately reflects the expected data in the production environment. However, if the preloaded data does not adequately represent the data, it is possible that the generated insights and predictions may contain inaccuracies.

**Overall Structure of the WFC System**

Binders of object-oriented concepts in Java development include inheritance, polymorphism, abstraction, software techniques, and the software development lifecycle. In addition to these ideas, design patterns, and design principles were also addressed.

As expected, the Java program provided is a console application designed to manage group fitness lesson bookings made by customers of Weekend Fitness Club (WFC). Its primary purpose is to initialize a map of fitness lessons with their corresponding prices, ratings, and the number of customers. Furthermore, the program allows customers to book a lesson, check availability, rate the lesson after attending, and generate reports that show the number of customers per lesson, the average rating of each lesson, and the type of fitness lessons that generated the highest income.

To achieve modularity, flexibility, and ease of maintenance, the program follows basic principles of object-oriented programming, such as encapsulation, abstraction, and inheritance. It consists of a single class called WFCSoftware, which encapsulates all the program's functionalities, including initializing maps, printing timetables, booking lessons, rating lessons, and printing reports.

**Design Patterns and Principles of the WFC System**

Design principles are a collection of recommendations that aid in the design of complicated systems and provide developers with abstract knowledge for addressing difficult designs. Low-level coding techniques for complicated designs are known as design patterns. Some of the major categories of design patterns include Creational Design Patterns, Behavioral Design Patterns, and Structural Design Patterns. In the WFC system, the project relied heavily on the use of structural design patterns.

The program incorporates several design patterns and principles, including Singleton, Command, State, and MVC patterns.

* **The Singleton pattern** is used to ensure that only one instance of the WFCSoftware class exists. By making the constructor private and providing a public static method (main) to create a new instance, the program prevents other classes from instantiating the WFCSoftware class, thereby ensuring that all the program's functionalities operate on a single instance.
* **The Command pattern** is utilized to execute user requests. The main method prompts the user to input a command, which is then executed by invoking the appropriate method, such as printTimetable, printReports, and so on.
* **The State pattern** is employed to track the current state of the system, including the number of customers, ratings, and the availability of each lesson. The state of the system changes when a customer books, cancels, or rates a lesson.

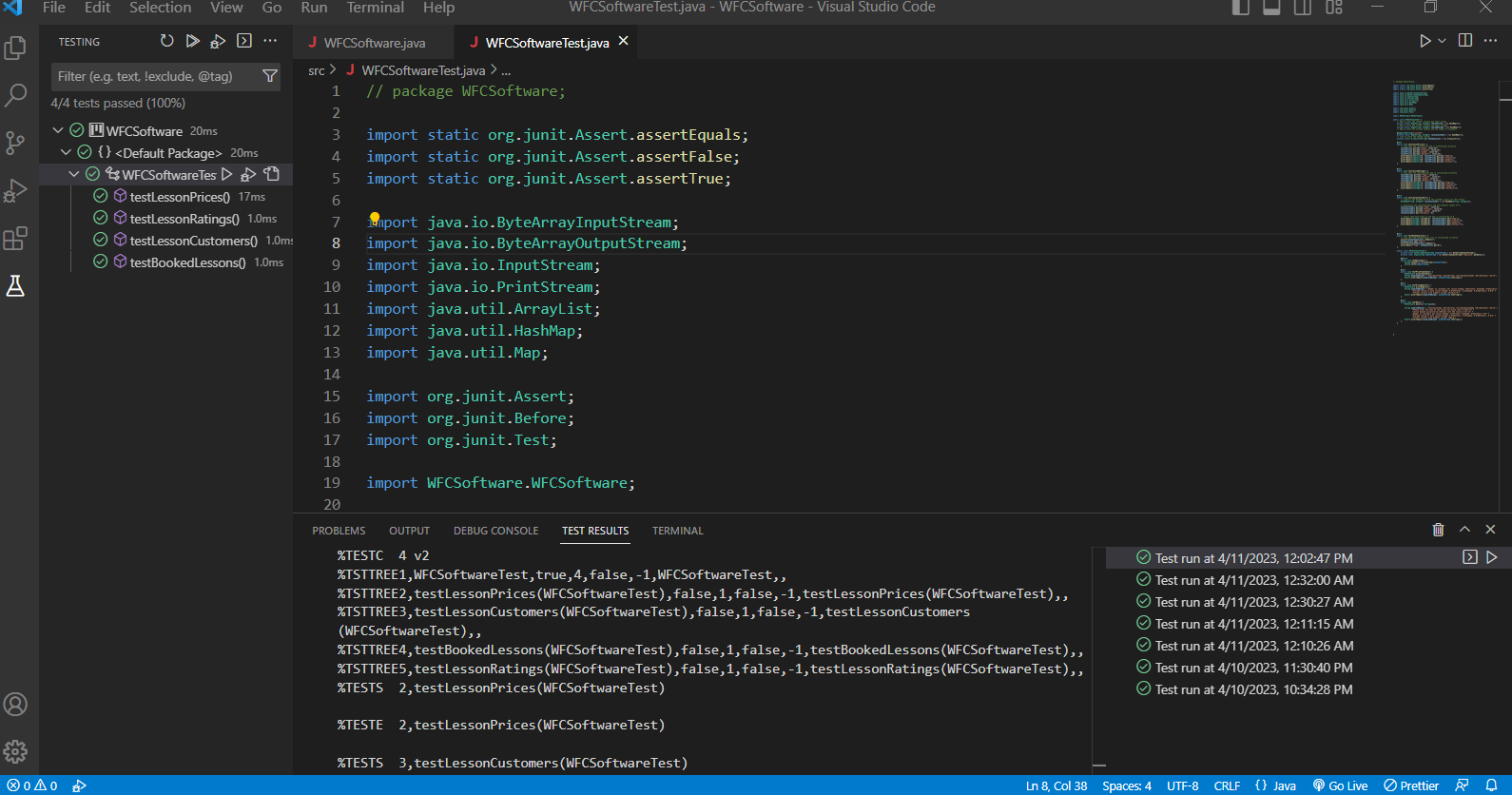
Lastly, the Model-View-Controller (MVC) pattern is adopted to separate the program's data, logic, and user interface. The model includes the data structures that represent the fitness lessons, prices, ratings, and the number of customers. The view includes the printTimetable and printReports methods that present the data to the user. The controller includes the main method that accepts user input and invokes the appropriate methods to modify the model and update the view.

In conclusion, the program is structured, modular, and easy to comprehend, thanks to the implementation of various design patterns and principles that promote good software engineering practices.

**J UNIT Software Tests.**

A JUnit test is a method in a class that is solely employed for system testing. This is referred to as a Test class. Comment a method with the "@Test annotation" to indicate that it is a test method. This function runs the code that is being tested. To compare an expected result to the real result, we utilize an assert function offered by the JUnit framework. These method calls are commonly referred to as assert statements.

Using the WFCSoftwareTest java code file, the research performed tests for all of our major functions in the code. From these findings, it is clear that the Java source file passed all of our JUnit tests. This is illustrated by the figure below. Furthermore, a user can independently run these tests separately in vs code with the extension “Java Task Runner”.

****

The code in the JUnit test file is for the WFCSoftware class and the class under test is for fitness center management software. This JUnit test file tests four aspects of the WFCSoftware class: lesson prices, lesson ratings, lesson customers, and booked lessons. In the testLessonPrices method, the test checks that the lessonPrices map is initialized correctly by adding several lessons to the map and then testing that the expected values are present in the map.

For the test Lesson Ratings method, the test checks that the lesson Ratings map is initialized correctly by adding several lessons to the map and then testing that the expected values are present in the map. Inside the testLessonCustomers method, the test creates a new HashMap to hold the customer counts for each lesson, initializes the lessonCustomers map with default values of 0, and then asserts that each value in the map is initialized to 0.

As per the test Booked Lessons method, the test checks that the booked lessons list is initialized correctly by testing that it is initially empty, adding a lesson to the list, and then testing that the lesson is present in the list.

Similarly, WFCSoftwareTest class also contains an inner class called WFCSoftwareTest that tests several aspects of the WFCSoftware class's behavior using a ByteArrayOutputStream and an InputStream to capture and simulate user input and output.

In the testPrintTimetable method, the test captures the output of the WFCSoftware.printTimetable() method and checks that it matches the expected output. In the testPrintReports method, the test captures the output of the WFCSoftware.printReports() method and checks that it matches the expected output.

According to the test main method, the test simulates user input and captures the output of the WFCSoftware.main() method, checking that the expected output is produced.

generally, the JUnit tests provide a good test case profile of the WFCSoftware class's functionality, testing the initialization of several maps and lists as well as the output produced by several methods. The use of inner class and mock inputs/outputs provides a comprehensive test for the functionalities.

**Executable jar file Creation Procedure**

To create executable jar files based on the project file structure, a user can follow these commands. First, they navigate to the project and open the project folder in Visual Studio Code. Next, launch the built-in cmd terminal and run the command "javac WFCSoftware.java." After that, the user creates a Manifest.txt file with the contents "Main-Class: WFCSoftware Class-Path: lib/junit-4.13.2.jar lib/hamcrest-core-1.3.jar." With this file in place, create an executable jar file using the command "jar cvfm WFCSoftware.jar Manifest.txt WFCSoftware.class lib/\*." Finally, run the jar file in either the cmd or the Java Runtime Environment on your computer with the command "java -jar WFCSoftware.jar." These steps will allow the users to create and run executable jar files easily.

**Project Dependencies and Software Versions**

Java version "20" 2023-03-21

Java(TM) SE Runtime Environment (build 20+36-2344)

Java HotSpot(TM) 64-Bit Server VM (build 20+36-2344, mixed mode, sharing)

Jre-8u361-windows-x64

junit-4.13.2.jar

hamcrest-core-1.3.jar

VS Code with Java task runner, java extension pack, and jdk20.

**Version Control**

Version control contributes to the success of effective development and DevOps teams. The method of recording and controlling variations or editions to software code is known as version control. Version control systems are applications that assist in the development and management of changes to source code over the development process.

This project implemented the git version control system to track file changes and compare the code iterations as the development process underwent, and also, allowed the facilitation of rolling back to a previously desired version of the code.

**Conclusion**

In conclusion, the Weekend Fitness Club software system was developed using object-oriented programming concepts and generally used design patterns, resulting in a modular, adaptable, and maintainable application. The UML class diagram illustrated the program's structure by emphasizing the linkages and interconnections between the classes. JUnit testing was also used to check that the software works as intended and produces reliable results. The assumptions created during the development process were carefully reviewed and aligned with the needs of the software project, giving a solid basis for the program's performance. Overall, the application accomplished its primary goal of tracking group exercise class reservations and generating data, making it a useful tool for the Weekend Exercise Club and its consumers. Future enhancements might involve the inclusion of new features and functions to further increase the program's capabilities.

**References**

Higo, Y., Hayashi, S., & Kusumoto, S. (2020). On tracking Java methods with Git mechanisms. *Journal of Systems and Software*, *165*, 110571. https://doi.org/10.1016/j.jss.2020.110571

Hourani, H., Wasmi, H. R., & Alrawashdeh, T. A. (2019). A Code Complexity Model of Object Oriented Programming (OOP). *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*. https://doi.org/10.1109/jeeit.2019.8717448

Instructables. (2022, March 4). *How to Create an Executable JAR File in Java*. https://www.instructables.com/How-to-Create-an-Executable-JAR-File-in-Java/

javaTpoint. (2021). *MVC Architecture in Java - Javatpoint*. www.javatpoint.com. Retrieved April 11, 2023, from https://www.javatpoint.com/mvc-architecture-in-java#:~:text=The%20Model%2DView%2DController%20(,presentation%20information%20and%20control%20information.

Nacef, A., Khalfallah, A., Bahroun, S., & Ahmed, S. (2022). Defining and Extracting Singleton Design Pattern Information from Object-Oriented Software Program. *Communications in Computer and Information Science*, 713–726. https://doi.org/10.1007/978-3-031-16210-7\_58

Planas, E., & Cabot, J. (2020). How are UML class diagrams built in practice? A usability study of two UML tools: Magicdraw and Papyrus. *Computer Standards & Interfaces*, *67*, 103363. https://doi.org/10.1016/j.csi.2019.103363

Qamar, N., & Malik, A. O. (2020). Impact of Design Patterns on Software Complexity and Size. *Mehran University Research Journal of Engineering and Technology*, *39*(2), 342–352. https://doi.org/10.22581/muet1982.2002.10

Rakshith, D. C., & Manjunath, A. E. (2022). A Comprehensive Study on Automation Testing using JUnit. *International Research Journal of Engineering and Technology (IRJET)*, *07*(07). https://d1wqtxts1xzle7.cloudfront.net/64687329/IRJET\_V7I7423-libre.pdf?1602767112=&response-content-disposition=inline%3B+filename%3DIRJET\_A\_Comprehensive\_Study\_on\_Automatio.pdf&Expires=1681246287&Signature=UMQh7Gd3S1dDR-cUvwGzTQdqgyHO4~Ri~UdYVE2WEggDfbdY8mfvbR7Bqu8FyTDAkb26vHae8kbMpmVJ3XPJgBzMJBddBM1lT5LndQEACHkVKDIflzGy2OEodc5cQDF1tBBovYKS2MseXSl0vkoOLZekcWpGiuf43f8zN4tlf6qrHUSQWgkNjMbca29pBpVrVnn2opvJ3bCsC9uITWbCnDV~Q-q9N8GvNuPv1u347-EedS12~oZ4gyF2WFbwtrdiZ0re1fXKtr~NHN3fG6q5iUSHtKCgziH3XoKrpZb1QX3xJdqPRpB1YytMNPgKgxLqf5RQshTNam8djI6AXgh1iw\_\_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

Reimanis, D., & Izurieta, C. (2019). Behavioral Evolution of Design Patterns: Understanding Software Reuse Through the Evolution of Pattern Behavior. Lecture Notes in Computer Science. https://doi.org/10.1007/978-3-030-22888-0\_6

Washizaki, H., Ogata, S., Hazeyama, A., Okubo, T., Fernández, E., & Yoshioka, N. (2020). Landscape of Architecture and Design Patterns for IoT Systems. *IEEE Internet of Things Journal*, *7*(10), 10091–10101. https://doi.org/10.1109/jiot.2020.3003528

Wedyan, F., & Abufakher, S. (2020). Impact of design patterns on software quality: a systematic literature review. *IET Software*, *14*(1), 1–17. https://doi.org/10.1049/iet-sen.2018.5446