Fitness Club Software System

By ( Name)

The Name of the Class (Course)

Professor (Tutor)

The Name of the School (University)

The City and State where it is located

The Date

**Introduction**

The research project involved building a Fitness Club Software (FCS) system for a Weekend Fitness Club (WFC) during the entire software development cycle. To achieve flexibility and scalability, the FCS system employs the Java programming language and its object-oriented properties. The project later employed some standard patterns throughout the design process, these include the singleton pattern, state pattern, command pattern, and Model-View-Controller Pattern. (MVC). A UML class diagram was also constructed to show the program flow, layout, and relationship between the Main class and the rest of the methods, types, and attributes incorporated in the code. For compilation and testing factors, the code was additionally prepopulated with useful data.

The study incorporated the software's main class, which consisted of four instance variables and three methods in the code. Various assumptions were also when inputting our data. First and foremost, the data is free of mistakes and missing numbers. Additionally, the input data was of the right data type, and lastly, the data was formatted consistently. The project used JUnit testing in the software testing process. This was done in order to perform unit tests on our code and guarantee that the user does not receive unexpected results. FCS’S JUnit tests produced a decent test case profile of the functionality of the FCS class. Finally, a description of how a user may execute the system on his computer as a console program or an executable jar file is provided.

**UML Class Diagram**

A Unified Modeling Language (UML) is a form of a static structural diagram in the field of software design that depicts the underlying framework of a system by displaying the system's classes, attributes, methods, and the connections between objects. The class diagram for the FCS is the foundation of the object-oriented modeling approach. It is used for both fundamental conceptualizations of the application's structure as well as specific modeling, which involves turning the models into Java code. Class diagrams were also used to model the data. The figure on the next page shows the composition of the FCS UML class diagram.

**FitSoft**

lessonPrices: Map(String, Integer)

lessonRatings: Map(String, Integer)

lessonCustomers: Map(String, Integer)

bookedLessons: ArrayList(String)

main(args: String[]) void

printTimetable() void

printReports() void

**FitnessLesson**

name: String

price: int

capacity: int

**Customer**

bookings: ArrayList<FitnessLesson>

bookLesson(lesson: FitnessLesson): boolean

cancelBooking(lesson: FitnessLesson): boolean

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

**Booking**

lesson: FitnessLesson

customer: Customer

**Description Of The Fcs Uml Class Diagram**

**There are 4 instance variables in the FitSoft class:**

• lesson pricing: a map that contains the prices of fitness lessons.

• lesson ratings: a map to store fitness lessons as well as their ratings.

• lesson clients: a map that contains the total number of fitness classes as well as the number of clients.

• booked lessons: an ArrayList (a list of arrays) containing the scheduled lessons.

**There are three methods in the FitSoft class:**

• main(): the program's main method, which initializes the maps, outputs the schedule of lessons, invites the client to reserve and rate a lesson, and it also generates the reports.

• printTimetable(): a method for printing a schedule.

• printReports(): a method that produces reports with the total number of clients for each lesson across all of the eight days, as well as the average rating of each lesson and the fitness-type lessons that earned the most revenue.

**A fitness lesson is denoted by the FitnessLesson class, which contains three instance variables:**

• name: a lesson's name.

• price: a lesson's price.

• capacity: the highest total number of clients who can schedule the lesson.

**A client is represented by the client class, which contains two instance variables:**

• name: the customer's name

• bookings: a list of arrays containing the client's scheduled lessons

**The Booking class defines a reservation and encompasses two instance variables:**

• lesson: a fitness lesson that the client has reserved

• customer: the person who has reserved the fitness lesson

**The following are the relationships between the classes:**

• FitnessLesson, Customer, and Booking are all components of FitSoft.

• FitnessLesson and Booking have a composition relationship, whereas the client and FitnessLesson have an aggregation interaction.

**Assumptions Made in the FitSoft System**

This current research study examines the assumptions applied during the software creation procedure for FitSoft's Java Project. These assumptions were applied in accordance with the needs of the software project. According to the given code, the preloaded data is organized as a list of dictionaries, with each dictionary representing a single record and the keys corresponding to the column names. A critical assumption is that all the loaded data is error-free, with no missing values, erroneous data types, or incompatible formatting.

Furthermore, the data is expected to stay static and unmodified during the execution of the program. However, if the data length exceeds the memory capacity, alternate data storage and retrieval methods may be required to prevent potential performance concerns. Finally, the preloaded data is supposed to precisely mirror what is anticipated in the actual production environment. However, if the preloaded data does not accurately represent the data, the resulting insights and projections may be inaccurate.

**Overall Project Structure of FCS System**

In the context of Java development, abstraction, polymorphism, inheritance, software methodologies, and the software development lifecycle serve as anchors for object-oriented ideas. Design patterns and design principles were also utilized in addition to these concepts. In the present scenario, the Java software offered is a console application/executable jar created to handle club fitness session reservations, submitted by the Fitness Club Software (FCS) members. Its main objective is to set up a map of fitness classes with the appropriate costs, reviews, and clientele. Customers may also use the program to schedule lessons, verify availability, review them after they've taken them, and produce reports that indicate the average number of clients for each lesson as well as the categories of fitness classes that brought in the most money.

The software system adheres to fundamental concepts of object-oriented programming, especially encapsulation, abstraction, and inheritance, to achieve scalability, flexibility, and simplicity of maintenance. The entire functionality of the application is included in a single class named FitSoft, including the initializing of maps, the production of schedules, the scheduling of lessons, the reviewing of lessons, and the generation of reports.

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

Design principles are a group of guidelines that help with the creation of complex systems and give programmers an abstract basis for dealing with challenging designs. The application uses a number of design concepts and patterns, such as the Singleton, Command, State, and MVC patterns. To guarantee that there is only a single instance of the FitSoft class, the Singleton pattern is utilized. This program prohibits other classes from executing the FitSoft class, guaranteeing that all of the program's features work on a single instance by keeping the constructor private and ensuring that the public static method main generates a new instance.

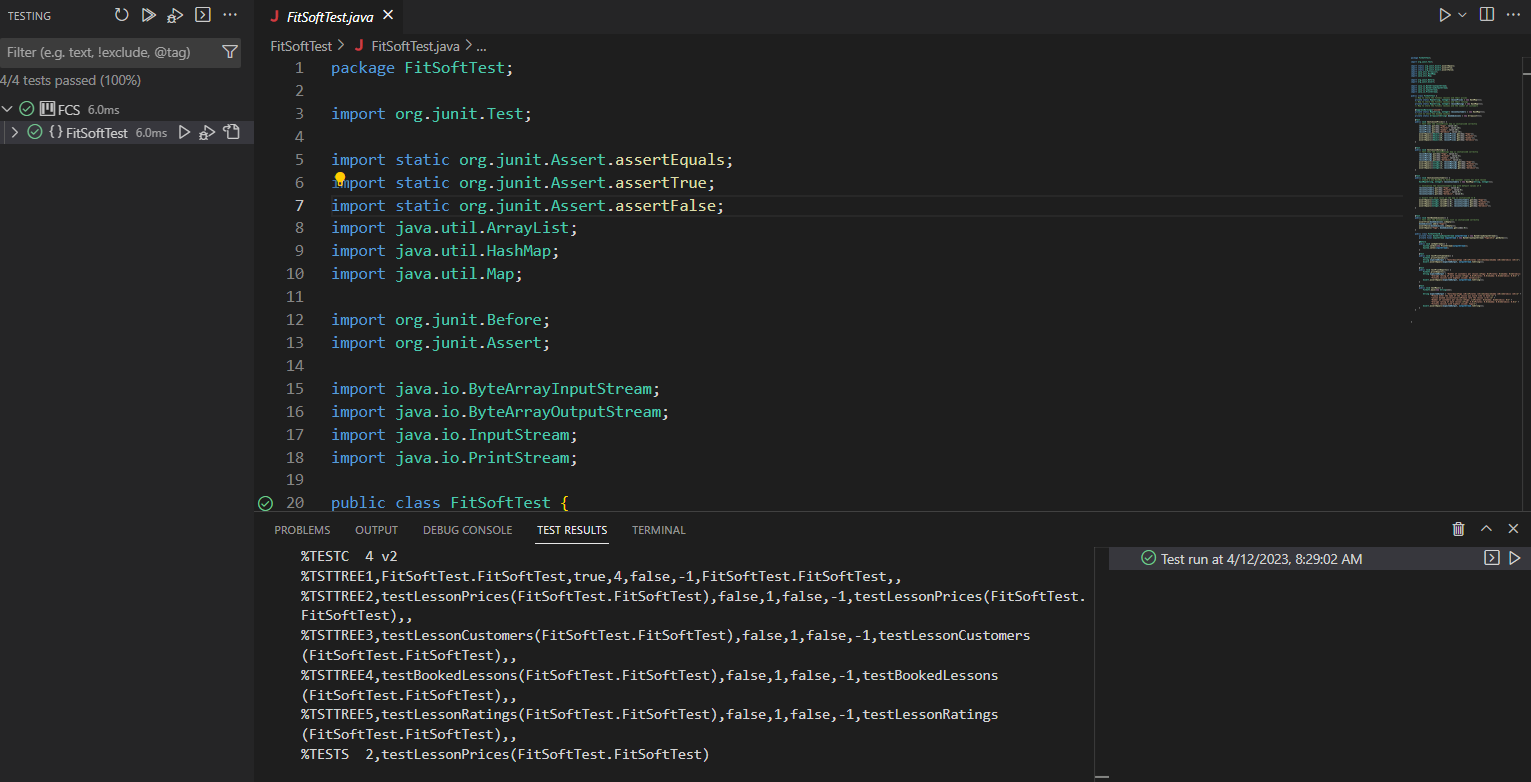
To fulfill requests submitted by users, the Command pattern is used. The user is prompted to provide a command in the main method, which is then carried out by calling the relevant method, such as Timetable, Reports, and so on. The State pattern is used to keep track of the system's present status, especially the number of users, reviews, and lessons that are still available. When an end-user schedules, cancels, or rates a class, the system's status is altered.

In the final step, the program incorporates the Model-View-Controller (MVC) pattern to isolate its data segment, logic segment, and user interface. The model part is composed of the data structures that represent fitness lessons, prices, ratings, and the number of customers. The view is basically part of the user interface which presents the data to the user through the printTimetable and printReports methods. The controller, which includes the main method, accepts user input and invokes the appropriate methods to modify the model and update the view. As a result of the implementation of several design patterns and ideas that support excellent software engineering practices, the program is organized, modular, and simple to understand.

**Software Testing with J UNIT Tests**

JUnit is a Java Unit Testing Platform for Java developers who may use it to build and run repeatable tests. Specifically, it is used for Unit Testing for tiny components of code, as the name indicates. After writing the code for the FCS, unit tests were built and employed. This process involves running all the tests, and they are all expected pass. Every time a new piece of code was introduced, all test functions were re-run to ensure that the code is not corrupted.

Unit tests were performed for all of the important methods in the code using the FitTest java code file. The project’s Java code passed all the unit tests. Using the "Java Task Runner" plugin, a user can perform these tests individually in vs code. The test results are shown in the snapshot below.



FCS is the fitness center management software being tested, and the code in the FitSoftTest file is for testing the FitSoft Java code. The lesson rates, lesson ratings, lesson clients, and booked lessons are all subject to testing using the JUnit test framework. By adding many lessons to the map and then confirming that the anticipated values are there, the test in the testLessonPrices function verifies that the lessonPrices map has been initialized appropriately.

By populating multiple lessons to the map and afterward validating that the anticipated values are there, the test for the lesson ratings method verifies that the lesson ratings map is initialized successfully. The test for the testLessonCustomers function generates a new HashMap to store user counts for every lesson, configures the lessonCustomers map with standard values of 0, and subsequently verifies whether every value in the map has been set to 0.

The test for the Booked Lessons method ensures that the booked lessons list is appropriately set up by verifying that it is originally empty, adds a lesson to the list, and afterward ensures that the lesson exists in the list.

In a similar way, the FitSoftTest class has an inner class named FitSoftTestB that tests numerous elements of the FitSoft class's behavior by capturing and simulating end-user input and output using a ByteArrayOutputStream and an InputStream.

Within the test PrintTimetable function, the test retrieves the output of the FitSoft.printTimetable method and tests that it matches the anticipated output. The testPrintReports function grabs and compare the result of the FitSoft.printReports method to the anticipated output.

The test for our main function mimics the input of the user and records the result of the FitsSoft.main method, ensuring that the intended output is provided. In particular, the JUnit tests give an excellent test case scenario of the feature set of the FitSoft class, evaluating the initialization of many maps and lists along with the output provided by several methods. The usage of inner classes and dummy inputs and outputs allows for a thorough examination of the functions.

**Dependencies and Software**

The table below shows the project dependencies and software versions.

|  |  |
| --- | --- |
| **Software Version** | **Dependencies** |
| Java version "20" 2023-03-21 | JUnit-4.13.2.jar |
| Java(TM) SE Runtime Environment (build 20+36-2344) | hamcrest-core-1.3.jar |
| Java HotSpot(TM) 64-Bit Server VM (build 20+36-2344, mixed mode, sharing) |  |
| Jre-8u361-windows-x64 |  |
| VS Code with Java task runner, java extension pack, and jdk20. |  |

**Version Control**

Version control System is a critical tool for software development and DevOps teams. Teams may function more intelligently as the source code grows by storing and managing variations or editions to software code. The git version control system was used in the FCS to track software changes. This enabled the possibility of rolling back in the case of system failure. As a result, the development process was simplified, with fewer mistakes and disputes. Additionally, version control allowed for effortless reversion to previous code versions while also ensuring the functionality of the project’s current iteration. In case of any bugs or issues that arose during development, another developer can quickly identify the source of the problem and return to a previous code version to correct it.

**Conclusion**

Developed using object-oriented programming concepts and generally used design patterns, the software system for Weekend Fitness Club had the primary goal of tracking group exercise class reservations and generating data. JUnit testing was implemented to ensure the software operates as intended and produces dependable outcomes. Careful review and alignment of assumptions with the software project's needs gave a solid basis for the program's performance. The UML class diagram was instrumental in illustrating the program's structure, emphasizing the linkages and interconnections between the classes. Future enhancements might involve the inclusion of new features and functions to further increase the program's capabilities. Overall, the Weekend Fitness Club software system is an impressive achievement that demonstrates the successful integration of programming concepts with a focus on functionality, performance, and user experience. As a modular, adaptable, and maintainable software, it is a useful tool for Weekend Exercise Club and its customers. Moving forward, the possibilities for further enhancing software capabilities are endless. In conclusion, the successful implementation of object-oriented programming concepts and design patterns resulted in a reliable and user-friendly application for Weekend Fitness Club. JUnit testing ensured dependable outcomes while the UML class diagram illustrated the program's structure. Careful consideration of assumptions during the development process established a solid foundation for software performance.

**References**

Bijlsma, L., Kok, A. J. F., Passier, H., Pootjes, H., & Stuurman, S. (2021). Evaluation of design pattern alternatives in Java. *Software - Practice and Experience*. https://doi.org/10.1002/spe.3061

Biniasz, K. (2023). What Are OOP Concepts in Java? How They Work and More. *Stackify*. https://stackify.com/oops-concepts-in-java/

Devroey, X., Gambi, A., Galeotti, J. P., Just, R., Kifetew, F. M., Panichella, A., & Panichella, S. (2021). JUGE : An infrastructure for benchmarking Java unit test generators. *Software Testing, Verification & Reliability*. https://doi.org/10.1002/stvr.1838

Fruhlinger, J. (2022, April 1). *What is Git? Version control for collaborative programming*. InfoWorld. https://www.infoworld.com/article/3654955/what-is-git-version-control-for-collaborative-programming.html

GeeksforGeeks. (2021). Software Development Life Cycle  SDLC. *GeeksforGeeks*. https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/

Molina, U. R., Kifetew, F. M., & Panichella, A. (2017). Java Unit Testing Tool Competition - Seventh Round. *2019 IEEE/ACM 12th International Workshop on Search-Based Software Testing (SBST)*. https://doi.org/10.1109/sbst.2019.00014

Naghdipour, A., Hasheminejad, S. M. H., & Barmaki, R. L. (2023). Software design pattern selection approaches: A systematic literature review. *Software - Practice and Experience*. https://doi.org/10.1002/spe.3176

Tian, F., Liang, P., & Babar, M. A. (2022). Relationships between software architecture and source code in practice: An exploratory survey and interview. *Information & Software Technology*, *141*, 106705. https://doi.org/10.1016/j.infsof.2021.106705

*UML Class Diagram Tutorial*. (n.d.). https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

Zhao, Y., Yang, Y., Zhou, Y., & Ding, Z. (2022). DEPICTER: A Design-Principle Guided and Heuristic-Rule Constrained Software Refactoring Approach. *IEEE Transactions on Reliability*, 1–10. https://doi.org/10.1109/tr.2022.3159851