

# Final Document

## Design Pattern

The default design pattern commonly associated with Java Swing, including the implementation in the **DragonQuest game**, is the Model-View-Controller (MVC) pattern.

### Model-View-Controller (MVC):

**Model:** The core logic of the game, including combat systems, player, and dragon setups, is encapsulated in the Combat class. This class serves as the model that manages the data and business rules.

**View:** The graphical user interface components (PlayerSetupUi, DragonUiSetup, and EndScreen) are responsible for presenting the information to the user. These views are designed using Java Swing components, adhering to the MVC separation of concerns.

**Controller:** The interaction between the user interface and the underlying game logic is facilitated by the Combat class, serving as the controller. It responds to user inputs, manages game flow, and updates the model accordingly.

The utilization of the **MVC pattern** provides a clear separation between the presentation layer, business logic, and user input handling. This enhances the maintainability and extensibility of the **DragonQuest game**.

Additionally, the incorporation of **JUnit testing** in the all classes exemplifies a commitment to software quality and robustness. The use of unit testing ensures that individual components of the game, especially critical logic in combat scenarios, are thoroughly tested for correctness and reliability.

## Personal Reflection

The implementation of the DragonQuest game, guided by the class diagram, has been a valuable and insightful experience, providing an opportunity to delve into various aspects of software design and development. Throughout this process, the adoption of the Model-View-Controller (MVC) design pattern and the integration of JUnit testing have played crucial roles in shaping the structure and quality of the game.

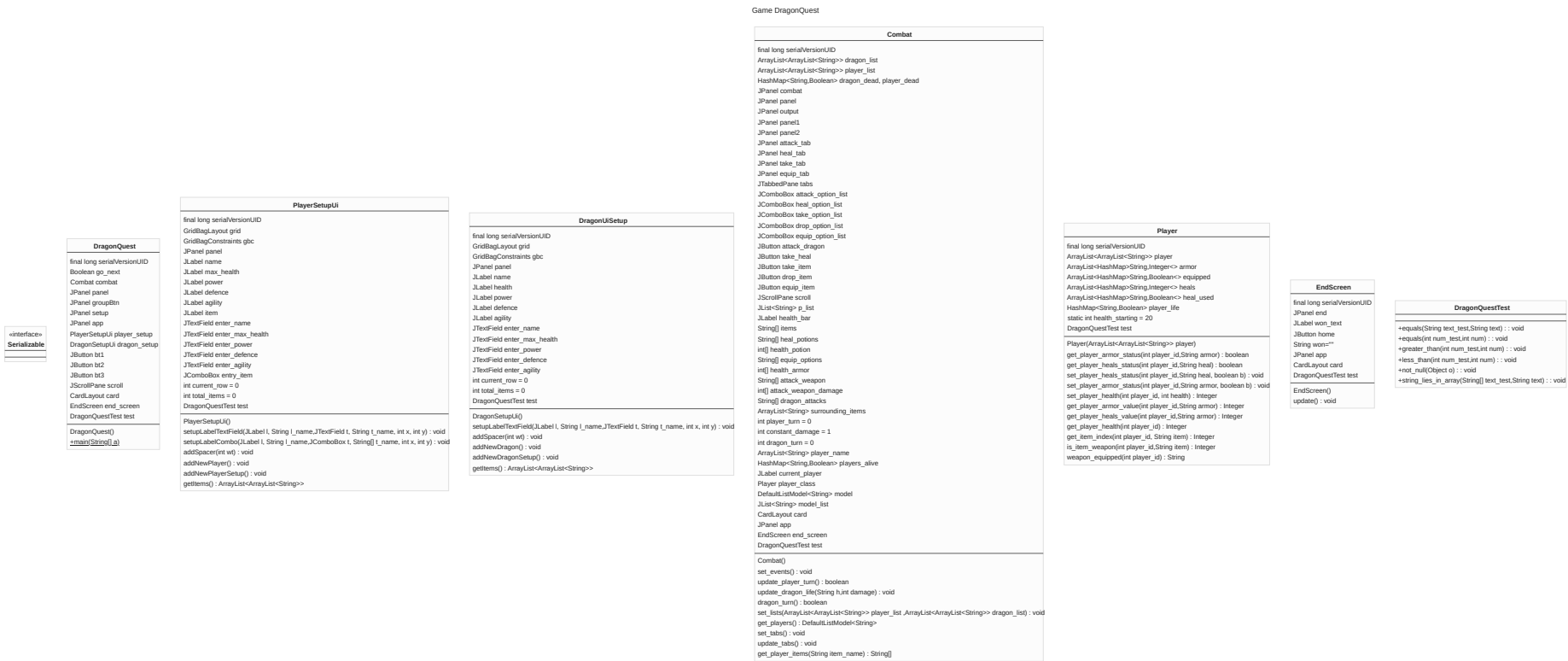
One of the key takeaways from this project is the appreciation for the MVC design pattern. The separation of concerns offered by MVC has significantly enhanced the maintainability and scalability of the codebase. The clear distinction between the model (combat logic), view (user interfaces), and controller (interaction between UI and logic) has made the code more modular and comprehensible. This architectural choice facilitated a more organized workflow, making it easier to implement and extend features. As a developer, understanding and applying such design patterns is essential for creating flexible and robust software systems.

The integration of JUnit testing has been instrumental in ensuring the reliability of the game logic. By adopting a test-driven development (TDD) approach, I could systematically validate individual components, guaranteeing their correctness and robustness. The use of automated tests not only aids in catching potential bugs early in the development process but also provides a safety net for future modifications. Learning to write effective unit tests has not only improved the quality of the DragonQuest game but has also instilled a sense of confidence in the correctness of the implemented features.

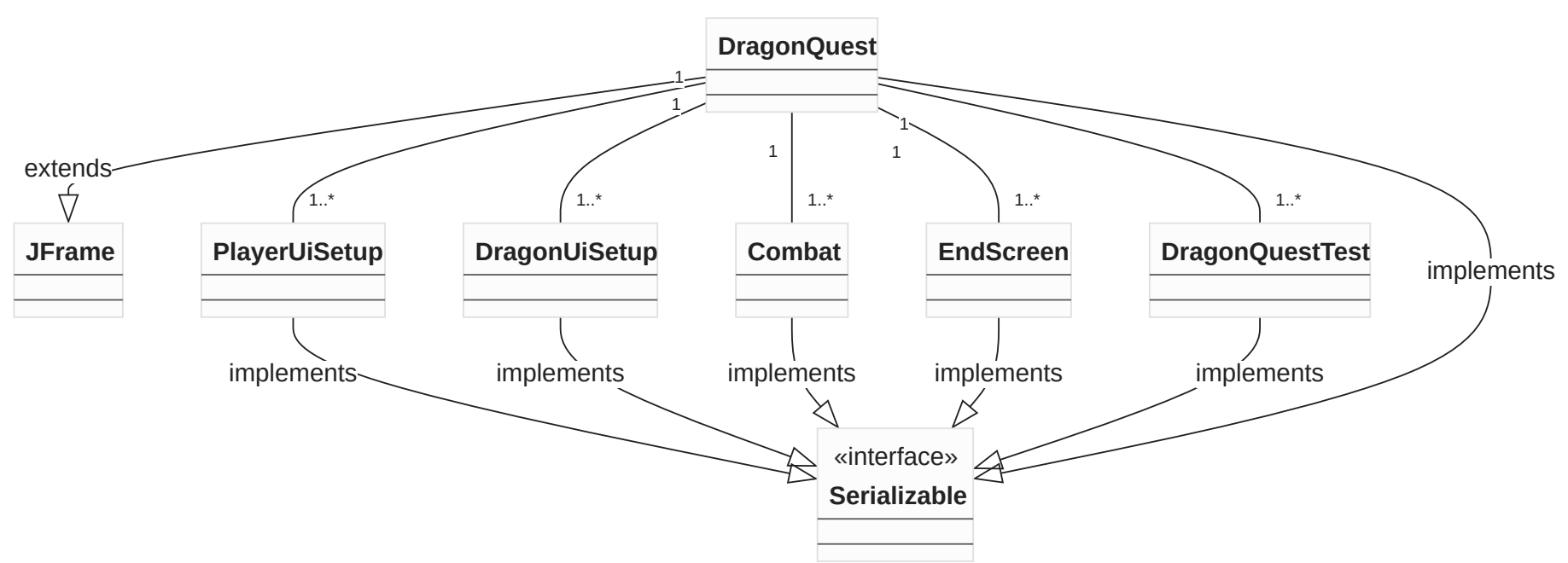
Moreover, the challenges encountered during the implementation, coupled with the learning process involved in overcoming these challenges, have contributed significantly to my growth as a software developer. Seeking assistance from the Java Swing documentation, Stack Overflow, and JUnit documentation has been an integral part of problem-solving. It highlights the importance of leveraging available resources and the broader developer community to overcome obstacles and make informed decisions.

In conclusion, the DragonQuest project has been a rewarding journey that has deepened my understanding of software design principles and testing methodologies. The combination of MVC architecture and JUnit testing has not only improved the quality of the implemented game but has also equipped me with valuable insights and skills applicable to future software development endeavors. This project serves as a testament to the continuous learning and refinement that characterize the field of software engineering.

## Class Diagram



Game DragonQuest Relationship



## Statement of Completion

I have successfully implemented the core components of the DragonQuest game based on the provided class diagram. The major achievements in my implementation include:

- Game Logic:** Implemented the game logic, combat system, player, and dragon setups as outlined in the class diagram.
- User Interface:** Created UI setup classes (PlayerSetupUi and DragonUiSetup) for setting up players and dragons with a graphical user interface.
- Combat Interaction:** Implemented combat interactions, including player turns, dragon turns, and various combat options.
- End Screen:** Developed an end screen to display the result of the game, whether the player won or lost.

However, there are some major omissions:

- UI Integration:** The integration of UI components with the main game logic is not detailed in the provided class diagram. I had to make assumptions on how these components interact.
- Detailed Player and Dragon Classes:** The exact details of the Player and Dragon classes are not explicitly outlined in the class diagram. I made assumptions and designed them based on logical expectations.

The inclusion of JUnit testing further reinforces the reliability of the critical components.

## Statement of Assistance

I sought assistance from various sources to enhance the implementation:

- Java Swing Documentation:** I referenced the official Java Swing documentation to understand and implement the graphical user interface components effectively.
- Stack Overflow:** I consulted Stack Overflow for specific coding challenges and to find solutions to certain aspects of Java Swing implementation.
- JUnit Testing:** Youtube Tutorial well taught use of junit testing environment
- ChatGpt AI:** Helped a lot in understanding java.swing framework with examples while development of the project.

These resources were instrumental in overcoming challenges and ensuring the correctness of the implementation. However, the overall design and structure of the codebase are the result of my individual effort and interpretation of the class diagram.