**Design Manual**

Introduction: The Design Manual serves as a comprehensive technical document delineating the architectural framework and design rationale of the Interactive Floor Plan Designer software. Intended for developers, architects, and technical stakeholders, it offers a detailed blueprint to facilitate software development, maintenance, and scalability.

1. Architecture Overview: The software's architecture encompasses a modular structure comprising distinct components orchestrated to achieve seamless functionality. At its core, the application consists of three primary layers: the Presentation Layer, Business Logic Layer, and Data Access Layer.

1.1 Presentation Layer:

* Responsible for rendering the user interface and facilitating user interactions.
* Implemented using Java Swing to provide an intuitive graphical interface for floor plan design.

1.2 Business Logic Layer:

* Governs the application's core functionality, including floor plan creation, manipulation, and saving/loading.
* Encompasses classes and modules for managing design elements, user actions, and application logic.

1.3 Data Access Layer:

* Manages data persistence, enabling the storage and retrieval of floor plan designs.
* Utilizes file I/O operations to save designs to disk and load them for future editing.

2. Design Patterns: The software incorporates several design patterns to enhance maintainability, extensibility, and code reuse.

2.1 MVC (Model-View-Controller):

* Employed to separate concerns and achieve a clear separation between data, presentation, and user interaction.
* Models represent floor plan elements and their properties, views render the graphical interface, and controllers handle user input and application logic.

2.2 Command Pattern:

* Facilitates undo/redo functionality by encapsulating user actions as command objects.
* Enhances modularity and extensibility by decoupling invokers (user interface) from receivers (application logic).

3. Component Descriptions: Each major component of the software is meticulously documented to elucidate its purpose, functionality, and interactions.

3.1 FloorPlanManager:

* Coordinates floor plan creation, editing, and persistence.
* Manages the collection of design elements and orchestrates their manipulation and rendering.

3.2 DesignElement:

* Abstract base class representing generic design elements such as walls, doors, windows, and furniture.
* Defines common properties and behaviors shared by all design elements.

3.3 Wall:

* Concrete implementation of the DesignElement class representing walls within the floor plan.
* Stores properties such as length, thickness, and position.

3.4 Door:

* Concrete implementation of the DesignElement class representing doors within the floor plan.
* Includes properties such as width, height, and swing direction.

3.5 Window:

* Concrete implementation of the DesignElement class representing windows within the floor plan.
* Contains attributes like size, style, and position.

4. Standards and Conventions:

4.1 Coding Standards:

* Follow the Java coding conventions outlined in the Oracle Code Conventions for the Java Programming Language.
* Utilize meaningful variable and method names to enhance code readability and maintainability.
* Adhere to consistent indentation and formatting practices for improved code clarity.

4.2 Naming Conventions:

* Class names should be nouns, written in camel case (e.g., FloorPlanManager).
* Method names should be verbs or verb phrases, written in camel case (e.g., renderFloorPlan).
* Variable names should be descriptive and reflect their purpose (e.g., wallLength, doorWidth).

4.3 Java Swing Guidelines:

* Follow the Java Swing documentation and best practices for building user interfaces.
* Utilize Swing components such as JFrame, JPanel, JButton, etc., for constructing the graphical interface.
* Implement event handling using ActionListener, MouseListener, or other appropriate interfaces for user interactions.

4.4 Exception Handling:

* Implement robust error handling mechanisms to gracefully handle unexpected situations and provide informative error messages to users.
* Follow Java's exception handling best practices, including using try-catch blocks to capture and handle exceptions appropriately.