**Real Estate Management System Documentation**

**System Design Overview**

The Real Estate Management System is designed to facilitate the buying and selling of properties, allowing users to register as customers or owners and manage their listings and offers. The system is built using Java and follows an object-oriented programming approach, leveraging design principles and patterns for maintainability and scalability.

**Key Components**

User Management:

User: Abstract class representing common attributes and methods for all user types.

Customer: Extends User, representing buyers.

Owner: Extends User, representing property owners.

Admin: Extends User, representing administrative users.

House Management:

A.House: Represents property details, including ID, owner ID, address, price, size, and status.

B.HouseService: Contains business logic related to houses, such as adding houses and retrieving available properties.

C.HouseDAO: Data Access Object (DAO) for database interactions related to houses.

Offer Management:

A.Offer: Represents an offer made by a customer on a house, including offer ID, house ID, customer ID, offer price, and status.

B.OfferService: Contains business logic for creating and retrieving offers.

C.OfferDAO: DAO for managing database interactions related to offers.

Applied Principles

A.Single Responsibility Principle (SRP): Each class has a single responsibility, promoting easier maintenance and testing.

B.Open/Closed Principle (OCP): The system is designed to be open for extension but closed for modification, allowing for new user types and functionalities without altering existing code.

C.Dependency Injection: Services are injected where needed, facilitating easier testing and reducing coupling.

D.Data Access Object (DAO) Pattern: Abstracts database interactions, promoting separation of concerns and keeping business logic separate from data access logic.

E.Encapsulation: All entity classes encapsulate their attributes, exposing only necessary getters and setters, ensuring data integrity.

Design Patterns

A.Singleton Pattern: Used in DAO classes (HouseDAO, OfferDAO) to ensure a single instance manages database connections, preventing resource waste.

B.Factory Pattern: Potential for future implementation to create user objects based on user type, promoting scalability.

C.Strategy Pattern: Could be implemented in the future for different pricing strategies for houses, allowing flexibility in pricing models.