Core Class Structure

1. **Map** is a central class that:
   - Aggregates multiple MapObject instances
   - Aggregates multiple Command instances
   - Is extended by TrottierTown, UploadHouse, and DumbledoresOffice
2. **MapObject** is an interface that:
   - Defines player_entered and player_interacted methods
   - Is implemented by multiple classes including Door, Building, Sign, PressurePlate, SortingHat, Desk, Candle, Bookshelf, Book, Portrait, Pensieve, Phoenix, Rug, and ChatBot
3. **Command** is an interface that:
   - Is implemented by ListCommand, PullCommand, RegisterRepoCommand, UploadHouse, ChatCommand, TakeBookCommand, and SortCommand
   - Is aggregated by Map

Design Patterns Implementation

1. Observer Pattern

- **DumbledoresOffice** (Model) contains:
  - currentPlayer: Player
  - playerHouse: House
  - playerPosition: Coord
  - houseObservers: List<HouseObserver>
  - playerObservers: List<PositionObserver>
  - Methods for managing observers including add/remove/notify operations
- Two types of observers:
  1. **HouseObserver** interface:
     - Implemented by Rug
     - updateHouse(House): void method
  2. **PositionObserver** interface:
     - Implemented by SortingHat, Phoenix, Book, Bookshelf, Portrait, Pensieve, TextBubble
     - updatePosition(Coord): void method

2. Strategy Pattern

- **ConversationStrategy** interface:
  - Methods: startConversation(): String and getResponse(String): String
  - Implemented by PensieveConversationStrategy, PeevesConversationStrategy, HermioneConversationStrategy, DumbledoreConversationStrategy, SnapeConversationStrategy, and BookConversationStrategy
  - Used by Portrait, Pensieve, and Bookshelf classes

3. Singleton Pattern

- **ChatBot** is a singleton:
  - instance: ChatBot (static field)
  - getInstance(): ChatBot method
  - getResponse(ConversationStrategy, String): String
  - Used by Portrait, Pensieve, and ConversationStrategy implementations

4. Flyweight Pattern

- **Book** class implements the Flyweight pattern:
  - flyweightStore: List<Book> (static field to store shared book instances)
  - getBook(String): Book (static method to retrieve or create book instances)
  - Used by Bookshelf to efficiently manage multiple book instances

Key Relationships

1. **SinglePlayerDoor** extends Door and:
   - Depends on DumbledoresOffice to check if the room is occupied before allowing players to enter
2. **SortingHat** implements MapObject, PositionObserver:
   - Contains TextBubble for player interaction
   - Manages the sorting quiz and determines the player's House
3. **TextBubble** implements PositionObserver:
   - Used by interactive objects to display messages when player is nearby
   - Aggregated by objects like Portrait, Pensieve, SortingHat, Phoenix, Bookshelf, and Door
4. **House** is an enumeration with values:
   - GRYFFINDOR, HUFFLEPUFF, RAVENCLAW, SLYTHERIN
   - Used by SortingHat to assign houses
   - Used by Rug to display appropriate colors and emblems
5. **Rug** implements MapObject, HouseObserver:
   - Changes appearance based on the assigned house
   - Called by DumbledoresOffice when house changes
6. **Interactive Objects**:
   - **Portrait** has conversationStrategy (Strategy pattern) and TextBubble
   - **Pensieve** has conversationStrategy (Strategy pattern) and TextBubble
   - **Bookshelf** manages Book instances using Flyweight pattern
   - **Phoenix** has interact() method and TextBubble
7. **Commands**:
   - **ChatCommand** depends on ChatBot for processing chat requests
   - **TakeBookCommand** depends on Bookshelf to retrieve books
   - **SortCommand** likely used by SortingHat for the sorting quiz