Mancala Board Game

Introduction

Through this article, I am going to explian the technology stack which I used to create Mancala web game, I will explore some code snippts and the internal system archtict. Also, you will find a demo vedio about how you can play in the docs dir.

For more information about Mancala board game, visit https://en.wikipedia.org/wiki/Kalah.

1. Project Environment

- Spring Tool Suite 3.8.2.RELEASE
- JDK 1.8
- Apache Maven 3.3.9
- WebSocket (STOMP)

2. Project Package Structure

- src/main/java
  - com.backbase.mancala
    - MancalaApplication.java
    - MancalaApplication.java
  - com.backbase.mancala.controller
    - ChatController.java
    - GameController.java
  - com.backbase.mancala.domain
    - Pebble.java
    - Pile.java
  - com.backbase.mancala.dto
    - GameBoard.java
    - GameStatus.java (Enum)
    - Message.java
    - Winner.java (Enum)
  - com.backbase.mancala.service
    - IBoardGame.java
    - MancalaGame.java
- src/main/resource
  - static
    - app.js
    - index.html
    - main.css
- src/main/test
  - com.backbase.mancala
    - MancalaApplicationTests.java

3. Project Class Diagram

Mancala class diagram describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

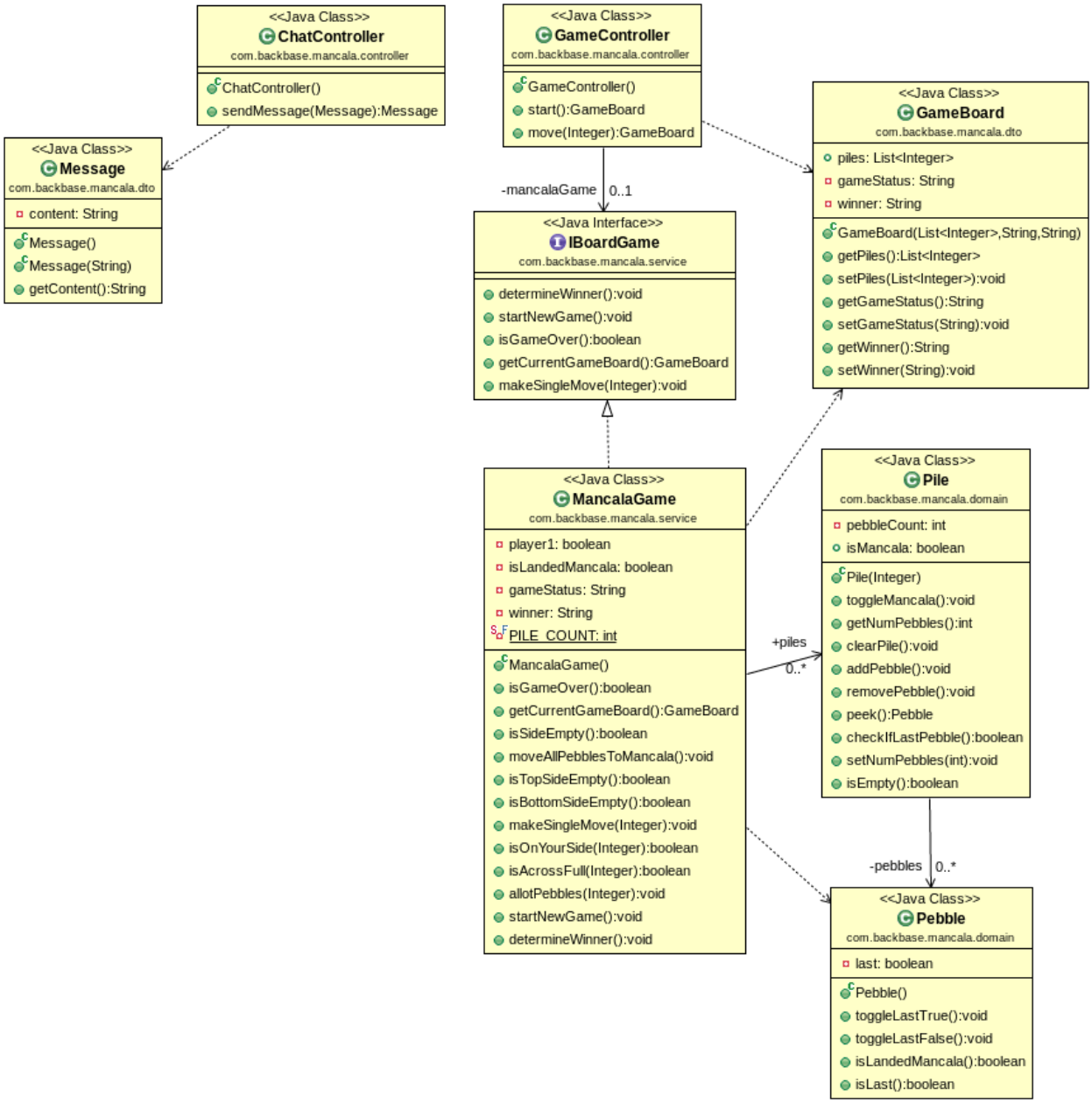

Figure 1: Mancala Class Diagram

4. Project Architecture

Mancala web game application architecture depends mainly on the messaging architecture using STOMP (Websockt Sub Protcol), Websockt provides full-duplex, two-way communication between client and server[1].

The following explains the message flow for the above architecture:

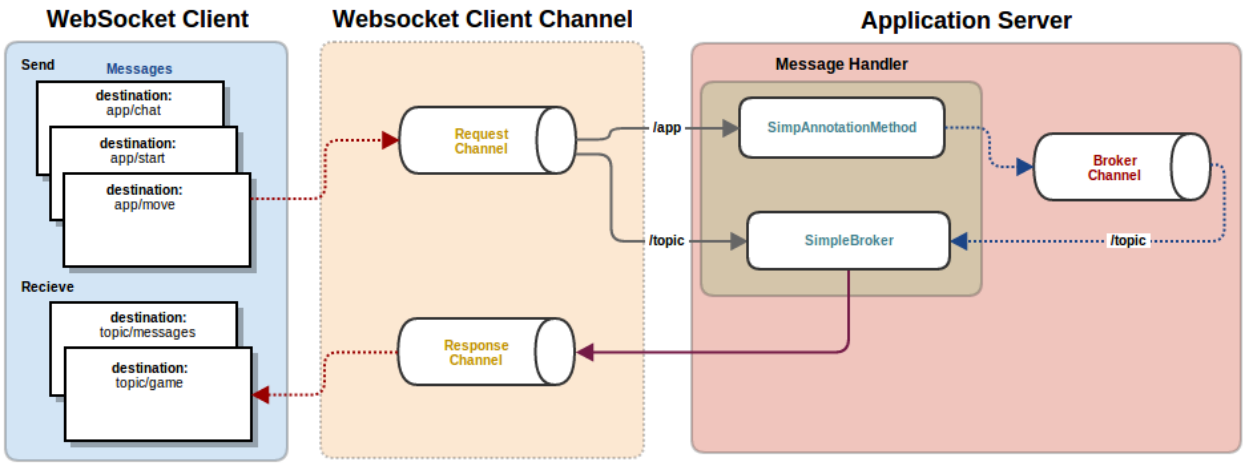

Figure 2: Mancala Application Architecture

- WebSocket clients connect to the WebSocket endpoint at "/websocket".
- Subscriptions to "/topic/game" pass through the "clientInboundChannel" and are forwarded to the broker.
- Messages sent to "/app/start" and "/app/move" pass through the "clientInboundChannel" then forwarded to the GameController.
- The GameController iniate a new game or make a move and return GameBoard through the "brokerChannel" as a message to "/topic/game" (destination is selected based on a convention but can be overridden via @SendTo).
- The broker in turn broadcasts messages to subscribers, and they pass through the "clientOutboundChannel".

Also, there's a chat feature which is working according to the same flow.

5. Project Testing/Running

There are some Junit cases to test the game functionalities, which are running during the project build phase. After building the project, go to the target dir and you will find an excutable deployable fat jar (mancala-game.jar) for the game which contains an embbeded tomcat application server.

To build the project, run the following mvn command:

```
mnv clean install
```

6. Future Enhancement

Actully, I did my best to build this beta version but still there are many enhancements (TODOs) to improve The current beta version and the followings are examples:

- We are using on a simple in-memory broker, we can StompBrokerRely with a dedicated message broker like ActiveMQ.
- The application can't handel more than one game but it's doable through a kind session managment mechanism.

Also, you can get a pull request or submit issues through project Git repository https://github.com/ashrafsarhan/mancala.

7. References

[1] Spring WebSocket Support http://docs.spring.io/spring/docs/current/spring-framework-reference/html/websocket.html