Design Document Chutes And Ladders

TABLE OF CONTENT

1. Description of the solution

- Description of the package , main Interfaces, classes and Exceptions

package com.game.board.api.chutesladders
ChutesAndLadderGameBoardServiceApplication :
Class that is used to bootstrap and launch a Spring application from a Java main method.

package **com.game.board.api.chutesladders.model**
Board : A document to be saved as a mondoDB collection with name : “gameBoard"

package **com.game.board.api.chutesladders.repository**
LadderGameRepository : an interface to get the game board to play with based on the gameName from MongoDB extending Mongo specific org.springframework.data.repository.Repository interface.

package **com.game.board.api.chutesladders.api**
LadderGameApi : An interface that defines defines public API or services of Chutes and ladder game like creating the board details and playing with the same.

package **com.game.board.api.chutesladders.config**
ConfigUtility : Configuration class used to look up externalized values by injecting the Spring org.springframework.core.env.Environment from property files where properties are resolved through the Environment reside in one or more "property source" objects

package **com.game.board.api.chutesladders.service**
LadderGameService : a general purpose implementation of LadderGameApi interface, providing the necessary services.

package **com.game.board.api.chutesladders.controller**
LadderGameBoardApiController : RestController handling the appropriate HandlerMapping with RequestMapping(method = RequestMethod.GET) to get a new board and to play the game with the same.

TODO: User defined Exception Handling

2. Design decision and data structure

Game Board Initialization:

Configurable properties: classpath:gameSetup/boardSetup.properties

int numSquares - represents the total number of squares present in the board

```
chuteandladder.boardSetup.numSquares=12
```

List<List<Integer>> ladders and snakes - to make the ladder and snake square jumps.

```
chuteandladder.boardSetup.snakesFromTo=11,5;10,4
chuteandladder.boardSetup.laddersFromTo=2,6;7,9
```

Game Play:

Map<String, Integer> :

Used to store the Player and their current position while playing.

String

status :

Indicating the status of the game "New game", "${playerName} Turn over", "${playerName} won"

gameName :

Used to represent which game a player is playing.

3. Motivation and Consideration of the design pattern

MVC pattern (Model-View-Controller):
Model - represents an object or JAVA POJO carrying data to store in the Mongodb.
View - View represents the visualization of the data that model contains. The response object is in a JSON format giving the details of the Board.
Controller - It controls the data flow into model object and updates the view whenever game turn is played.

Requirements:
Initially Game board should be created in the mongodb database specified in the configurations in the application.properties running on MongoDB server (localhost: 27017 (default port)..)
The player takes turns or stops to play based on the status got in the response.
The game board can be created with user specified number of squares, ladders and snakes.

Assumptions:
1. Initially the game board details are present in the db or can be created by calling an endpoint : / games using REST API.
2. The player takes turn or stops playing by calling the endpoint : /gamesplay/{gameName}/ {playerName}/{diceRolled}, based on the status we get in the response.
3. The diceRolled provided in a URI as pathVariable should be from 1-6.
4. Only one document with gameName = ChuttesAndLadder should be present in db.gameBoard

Constraints:
BoardInitialization Issue:
1. If there is an issue while creating the new game with specified board details - NullPointerException
2. If there is an issue with processing (parsing, generating) JSON content - JsonProcessingException

Gap:
1. The round robin turn for players to play. The game played is based on the way the exposed apis are called.

Business Logic: actions to be implemented:
1. getChuttesAndLadderBoard:
Create only one board with initial values like snakes, ladders and numSquares obtained from the property file and save the same in the chutesLadder_db database as a document in a "gameBoard".
2. gamePlay:
Based on the gameName get the game board on which the player is playing and update the player with its new position on the board when the dice is rolled. i.e,
> get the current position from the DB mapped to the said player,
> add the same with the dice rolled number

> check if the jump leads to a ladder square or a snake square and make the final jump accordingly

> update the status to won if the new position is greater than or equal to the total number of squares present else indicate the current player's turn as over.

Benefits:

- The design uses MVC pattern, with services as an api interface where we can create reusable, small classes and are easier to read and test.
- Easier to Update the application.

Testing Strategy:

http://localhost:8080/games

```
{"id":"5db3d5a33845c0fd0f4c924a","numSquares":12,"ladders":[[2,6],[7,9]],"snakes":[[11,5],
[10,4]],"game_port":8080,"status":"New
Game","gameName":"ChuttesAndLadder","playerPositionMapping":null}
```

http://localhost:8080/gamesplay/ChuttesAndLadder/Nanditha/2

Jump from 2 to 6

```
{"id":"5db3d5a33845c0fd0f4c924a","numSquares":12,"ladders":[[2,6],[7,9]],"snakes":[[11,5],
[10,4]],"game_port":8080,"status":"Nanditha Turn
over.","gameName":"ChuttesAndLadder","playerPositionMapping":{"Nanditha":6}}
```

http://localhost:8080/gamesplay/ChuttesAndLadder/Kantha/3

```
{"id":"5db3d5a33845c0fd0f4c924a","numSquares":12,"ladders":[[2,6],[7,9]],"snakes":[[11,5],
[10,4]],"game_port":8080,"status":"Kantha Turn
over.","gameName":"ChuttesAndLadder","playerPositionMapping":{"Nanditha":6,"Kantha":3}}
```

http://localhost:8080/gamesplay/ChuttesAndLadder/Nanditha/1

Jump from 7 to 9

```
{"id":"5db3d5a33845c0fd0f4c924a","numSquares":12,"ladders":[[2,6],[7,9]],"snakes":[[11,5],
[10,4]],"game_port":8080,"status":"Nanditha Turn
over.","gameName":"ChuttesAndLadder","playerPositionMapping":{"Nanditha":9,"Kantha":3}}
```

http://localhost:8080/gamesplay/ChuttesAndLadder/Kantha/4

```
{"id":"5db3d5a33845c0fd0f4c924a","numSquares":12,"ladders":[[2,6],[7,9]],"snakes":[[11,5],
[10,4]],"game_port":8080,"status":"Kantha Turn
over.","gameName":"ChuttesAndLadder","playerPositionMapping":{"Nanditha":9,"Kantha":9}}
```

http://localhost:8080/gamesplay/ChuttesAndLadder/Nanditha/2

Jump from 11 to 5

```
{"id":"5db3d5a33845c0fd0f4c924a","numSquares":12,"ladders":[[2,6],[7,9]],"snakes":[[11,5],
[10,4]],"game_port":8080,"status":"Nanditha Turn
over.","gameName":"ChuttesAndLadder","playerPositionMapping":{"Nanditha":5,"Kantha":9}}
```

http://localhost:8080/gamesplay/ChuttesAndLadder/Kantha/3

```
{"id":"5db3d5a33845c0fd0f4c924a","numSquares":12,"ladders":[[2,6],[7,9]],"snakes":[[11,5],
[10,4]],"game_port":8080,"status":"Kantha
Won","gameName":"ChuttesAndLadder","playerPositionMapping":{"Nanditha":5,"Kantha":12}}
```

4. Flow Chart of use cases / Description of the methods

Pre-req. The data models should be Board entry in the Db based on the only one unique game name i.e ensure that the indexed fields do not store duplicate values
db.gameBoard.createIndex({"gameName" :1}, { unique: true })
```
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

1. Initialize the game board.

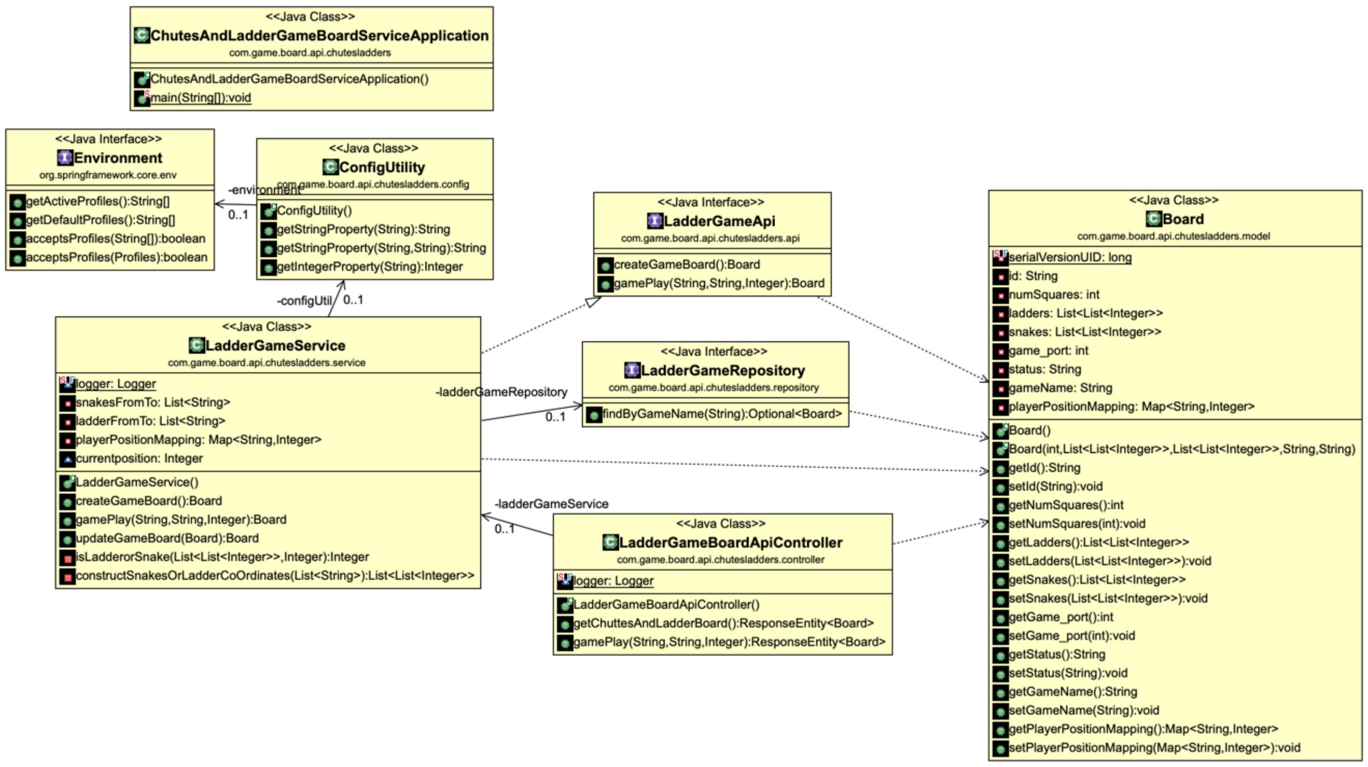
```
{"id":"5db3d5a33845c0fd0f4c924a","numSquares":12,"ladders":[[2,6],[7,9]],"snakes":
[[11,5],[10,4]],"game_port":8080,"status":"New
Game","gameName":"ChuttesAndLadder","playerPositionMapping":null}
```

2. Have a class constant variables used during the process
   numSquares, snakesFromTo, ladderFromTo, playerPositionMapping, currentposition, newposition

3. Actions implementations:
   Main methods:
   1. **public Board createGameBoard()**
      : create the Board details in the DB with new Board(numSquares, ladders, snakes, "New Game", "ChuttesAndLadder");
      : read the data from property file and map it as List<List<Integer>> entry in mongoldb.
private List<List<Integer>> constructSnakesOrLadderCoOrdinates(List<String> snakesOrLadderFromTo)
   2. **public Board updateGameBoard(Board board)**
      : To update the board in the database.
   3. **public Board gamePlay(String gameName, String playerName, Integer diceRolled)**
      : get the currentposition from the db,
      : isLadderorSnake - update the newposition based on dice rolled, and the type of square I.e whether it is snake jump or ladder jump to get the final position
   private Integer isLadderorSnake(List<List<Integer>> ladderOrSnake, Integer newposition)
      : setPlayerPositionMapping with the new position for the current player.
      : check if newposition is greater than or equal to getNumSquares(), and update the current player has won if true else update that current player's turn is over

   Helper methods:
   1. To read the property file
   public String getStringProperty(String pPropertyKey)
   public String getStringProperty(String propKey, String defaultValue)
   public Integer getIntegerProperty(String pPropertyKey)

5. UML Diagram


<<Java Class>>
ChutesAndLadderGameBoardServiceApplication
com.game.board.api.chutesladders
ChutesAndLadderGameBoardServiceApplication()
main(String[]):void
<<Java Interface>>
Environment
org.springframework.core.env
getActiveProfiles():String[]
getDefaultProfiles():String[]
acceptsProfiles(String[]):boolean
acceptsProfiles(Profiles):boolean
-environment
0..1
<<Java Class>>
ConfigUtility
com.game.board.api.chutesladders.config
ConfigUtility()
getStringProperty(String):String
getStringProperty(String,String):String
getIntegerProperty(String):Integer
-configUtil
0..1
<<Java Class>>
LadderGameService
com.game.board.api.chutesladders.service
logger: Logger
snakesFromTo: List<String>
ladderFromTo: List<String>
playerPositionMapping: Map<String,Integer>
currentposition: Integer
LadderGameService()
createGameBoard():Board
gamePlay(String,String,Integer):Board
updateGameBoard(Board):Board
isLadderorSnake(List<List<Integer>>,Integer):Integer
constructSnakesOrLadderCoOrdinates(List<String>):List<List<Integer>>
<<Java Interface>>
LadderGameApi
com.game.board.api.chutesladders.api
createGameBoard():Board
gamePlay(String,String,Integer):Board
-ladderGameRepository
0..1
<<Java Interface>>
LadderGameRepository
com.game.board.api.chutesladders.repository
findByGameName(String):Optional<Board>
-ladderGameService
0..1
<<Java Class>>
LadderGameBoardApiController
com.game.board.api.chutesladders.controller
logger: Logger
LadderGameBoardApiController()
getChuttesAndLadderBoard():ResponseEntity<Board>
gamePlay(String,String,Integer):ResponseEntity<Board>
<<Java Class>>
Board
com.game.board.api.chutesladders.model
serialVersionUID: long
id: String
numSquares: int
ladders: List<List<Integer>>
snakes: List<List<Integer>>
game_port: int
status: String
gameName: String
playerPositionMapping: Map<String,Integer>
Board()
Board(int,List<List<Integer>>,List<List<Integer>>,String,String)
getId():String
setId(String):void
getNumSquares():int
setNumSquares(int):void
getLadders():List<List<Integer>>
setLadders(List<List<Integer>>):void
getSnakes():List<List<Integer>>
setSnakes(List<List<Integer>>):void
getGame_port():int
setGame_port(int):void
getStatus():String
setStatus(String):void
getGameName():String
setGameName(String):void
getPlayerPositionMapping():Map<String,Integer>
setPlayerPositionMapping(Map<String,Integer>):void