Assignment I3: Saving Program State

Published | Edit | ⋮

This is an individual assignment. This assignment will be graded as 0 to max points. Scroll down for submission instructions at the end.

Assignment 3 builds on Assignment 2. Please make sure you have completed Assignment 2 before starting this assignment. In this assignment, you will be adding more features to the original tic-tac-toe game. This assignment focuses on providing robustness to the application. **You do not have to use the user interface provided in I1. Please use postman or JUnit tests to test your application.** You are supposed to complete this assignment in Java programming language.

1. Setup instructions

This assignment builds on the previous assignment. If you have not already completed assignment 2, then please do so before starting assignment 3.

In this assignment, you will be using **SQLite database** (https://www.sqlite.org/index.html) . As the name suggests, it is a lightweight database system. You will need Java Database Connectivity (JDBC) in your application to connect to the database and perform operations. See tutorial **here** (https://www.tutorialspoint.com/sqlite/sqlite_java.htm) .

Add the following to your pom.xml file:

```
<dependency>
    <groupId>org.xerial</groupId>
    <artifactId>sqlite-jdbc</artifactId>
    <version>3.32.3.2</version>
</dependency>
```

2. Tasks

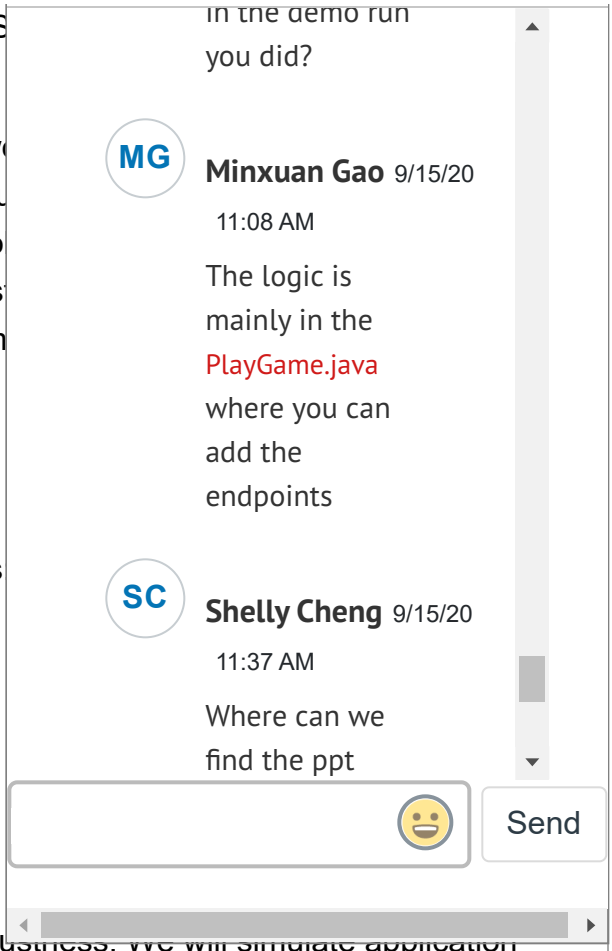
You are required to complete the following programming tasks to complete the assignment:

1. Add database storage to the tic-tac-toe game.
2. Read and write game data to and from the database.
3. Write JUnit test cases for the application.

Course Chat

4. Perform style checks (CheckStyle) and static analysis (S
   bugs.

The game will support only one game at a time, i.e., only tw
be crash tolerant, meaning if the application crashes, it shou
resume the game (if applicable). One way to achieve this ro
every move. You must create a table(s) in the database to s
evaluate you on completing the game using proper program
among other things.



3. Grading Scheme

We will evaluate your submission on four criteria, as follows

1. Robustness Tests
2. Game Logic Tests
3. Check Style compliance
4. Spotbugs compliance

3.1 Robustness Tests

We have several tests to validate the tic-tac-toe game's robustness. We will simulate application crashes and your application must be able to recover the game state before crash. Your game must follow these as described below:

1. Every time a new game starts, the database table(s) must be cleaned.
2. If the application crashes after a move, the application must reboot with the game board's last move.
3. If the application crashes after the game was a draw, but no new game started, the application must reboot to show the draw game board.
4. If the application crashes after the game has ended with a winner, but no new game started, the application must reboot to show the corresponding game board.
5. If player 1 had started a game and the application crashed, then the application must be able to reboot with player 1 as part of the game board.
6. If player 2 had joined a game and the application crashed, then the application must be able to reboot with player 2 as part of the game board with the corresponding game board status.
7. If a new game was created and the application crashed, it should reboot to a new game board.
8. If the application crashed after a player made an invalid move, the application must reboot with the last valid move state.
9. If the game crashes between a request to an endpoint and dispatch of the application's response, the data should **NOT** get persisted in the database.

We do not require you to write JUnit tests for the above rules. However, if you write tests, you can identify bugs and fix them.

3.2 Game Logic Tests

You must write JUnit tests for each method in your application. Tests must cover all conditional branches within a method. Make sure that you write tests to validate database operations. All previous test cases must pass:

1. A player cannot make a move until both players have joined the game.
2. After game has started Player 1 always makes the first move.
3. A player cannot make two moves in their turn.
4. A player should be able to win a game.
5. A game should be a draw if all the positions are exhausted and no one has won.

3.3 Check Style Compliance

In this assignment, we will follow **Google Java Style (https://google.github.io/styleguide/javaguide.html)** . Please go through the document to understand the recommended style and make the necessary changes to your application's code.

(https://github.com/Programming-Systems-Lab/4156-PublicAssignmentBackground/blob/master/Part-2/README.md#53-spotbugs-compliance) 3.4 Spotbugs Compliance

We require that you resolve all issues identified by Spotbugs.

You can ask for help with any of the technologies involved in this assignment by posting questions or problems in this course's individual_project folder on **piazza, (https://courseworks2.columbia.edu/courses/104335/external_tools/1456)** on relevant community forums, on stackoverflow, etc. However, do not ask for help writing the code, this is a violation of the university's academic honesty policy.

Submission instructions: Submit a zip file containing hw1 directory. You may resubmit repeatedly until the deadline. If you were unable to complete this assignment for any reason, submit anyway. In that case you should include a text explaining how far you got and what went wrong.

Points	20
Submitting	a file upload
File Types	zip

Due	For	Available from	Until
Oct 8, 2020	Everyone	Oct 1, 2020 at 12:01am	Oct 23, 2020 at 11:59pm

Assignment 3 Rubric

Criteria	Ratings		Pts
Robustness Tests We have created 9 test cases that tests the program state. You will be getting one point for each passing test case.	**9 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	9 pts
Spotbugs Your application must pass Spotbugs check. We will deduct one point for any Spotbugs warning or error.	**3 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	3 pts
Game Logic Tests Your test cases must cover all methods and conditional branches within methods. We will deduct 0.5 points for every missed method or conditional branch. 1 point will be deducted for each failing test.	**5 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	5 pts
CheckStyle Compliance 1 point deduction per 2 errors (other than indentation and tabs)	**3 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	3 pts
			Total Points: 20