New Features

UML Class Diagrams:

The UML class diagrams for the new features is in the file:

- ../misc/diagrams/RefactoredCode/UMLClassDiagrams/NewFeature.pdf

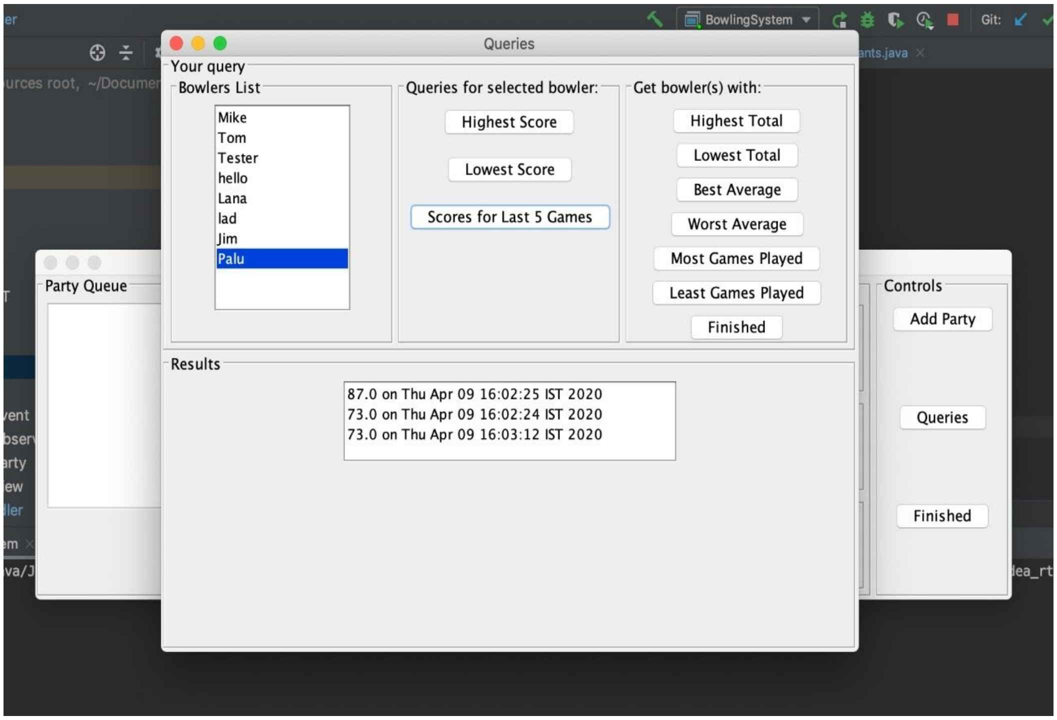
Sequential Class Diagram:

The sequential diagrams for the new features are in the file::

- ../misc/diagrams/RefactoredCode/SequentialDiagram/NewFeature.pdf

Features Implemented:

Feature 1: Ad-hoc queries



```
87.0 on Thu Apr 09 16:02:25 IST 2020
73.0 on Thu Apr 09 16:02:24 IST 2020
73.0 on Thu Apr 09 16:03:12 IST 2020
```

A new feature was implemented for performing ad-hoc queries on the existing database. This gives the user options to select from a variety of queries:

- Highest score for a selected bowler
- Lowest score for a selected bowler
- Scores for the past 5 games for a selected bowler
- Bowler with highest total score over all games
- Bowler with lowest total score over all games
- Bowler with the best average score per game
- Bowler with the worst average score per game
- Bowler with most games played
- Bowler with least games played

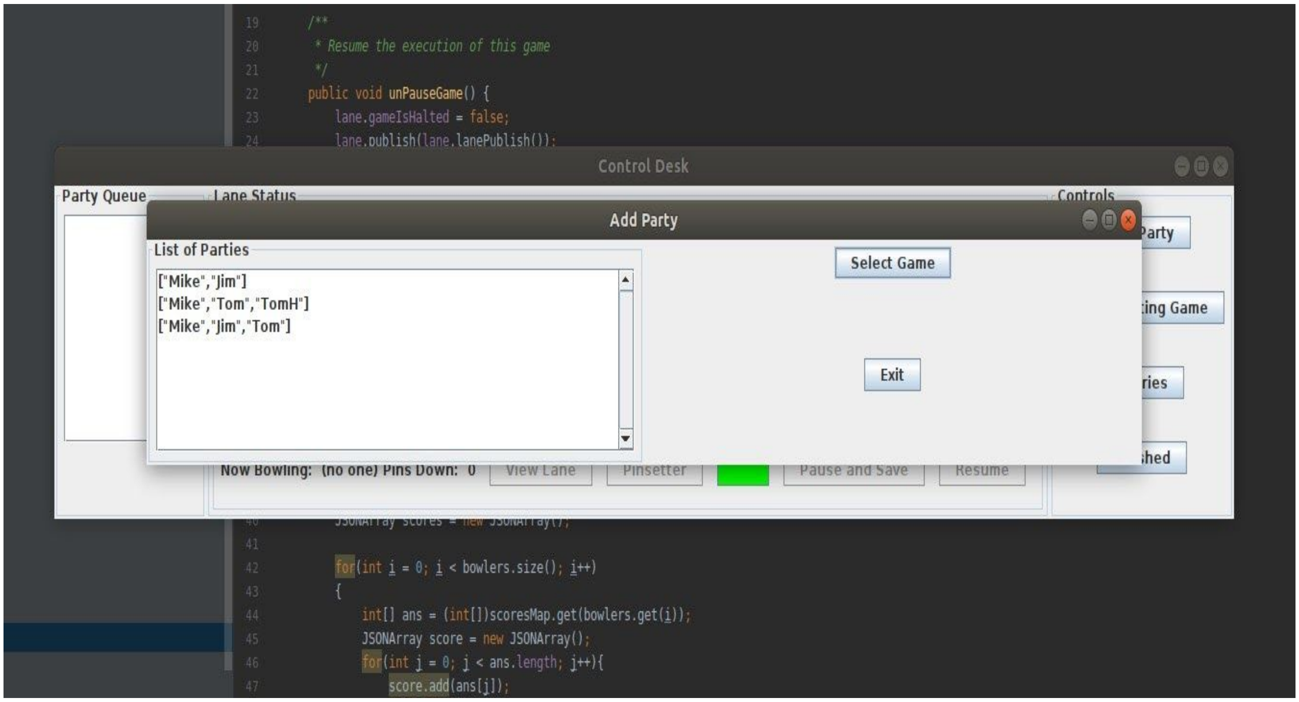
The results for the queries about selected bowlers are displayed along with the date of the game.

The results for the queries for the Bowler with … show the bowler's name along with the related statistic.

Related Classes implemented:

Class Name	Description
JsonIO	Reads and writes JSON objects using the json.simple library
QueriesView	Creates the view for querying and manages the selection of bowlers
QueriesHandler	Takes queries given by the view and returns the appropriate results
DatabaseHander	Writes game report data to the database and manages the database

Feature 2: Pause/Resume and Load Saved Game

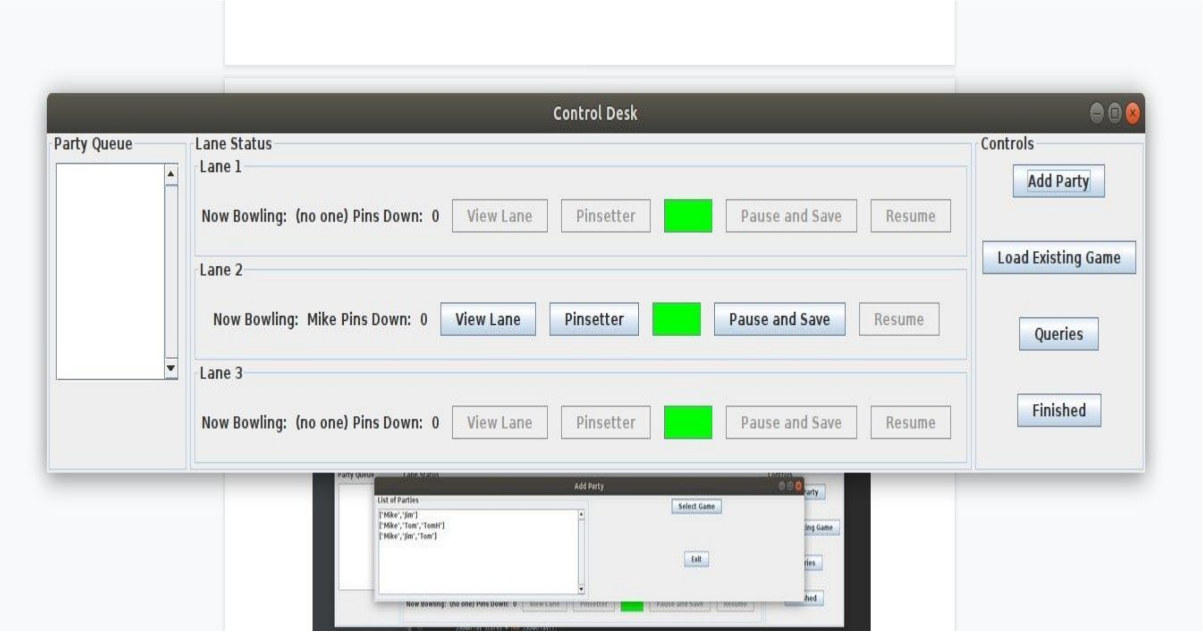


```
/**
 * Resume the execution of this game
 */
public void unPauseGame() {
    lane.gameIsHalted = false;
    lane.publish(lane.lanePublish());
```

```
    JSONArray scores = new JSONArray();

    for(int i = 0; i < bowlers.size(); i++)
    {
        int[] ans = (int[])scoresMap.get(bowlers.get(i));
        JSONArray score = new JSONArray();
        for(int j = 0; j < ans.length; j++){
            score.add(ans[j]);
```

```
["Mike","Jim"]
["Mike","Tom","TomH"]
["Mike","Jim","Tom"]
```



Lane 1 — Now Bowling: (no one) Pins Down: 0
Lane 2 — Now Bowling: Mike Pins Down: 0
Lane 3 — Now Bowling: (no one) Pins Down: 0

This feature allows users to pause their game and continue it again in future even after the game is closed .
To run a stored game there should be an empty lane ,if there are none then user must wait until a lane is available .

Related Classes implemented:

Class Name	Description
LaneCurrentStatus	This class holds the current state of a lane
LaneRunningStatus	This class take the current state of and creates JSON object
MessageDisplayWindow	This window is displayed whenever there is no available lane to run a stored game.
JsonConverter	Reads JSONObjects and converts them into the required form for saving and loading games
LoadExistingGameView	Creates a view window for selecting and loading an existing game
LoadGameFrame	Creates the frame for LoadExistingGameView and has utility methods for managing it
GameLoader	Provides static methods for using JsonConverter and get variables required for loading an existing game

Feature 3: Multiplayer and Player Storage

The multiplayer feature was implemented by just changing the maxPatronsPerParty variable to 10, which enabled each Lane to have more than 6 players. The player storage feature was already implemented by the system but was refined by removing deprecated methods.