The Applications Of Artificial Intelligence For Video Game Non-Player-Characters

Academic Year	**Module**	**Assessment Number**	**Assessment Type**
2025	5CS019/HJ1: Object-Oriented Design and Programming (Herald College, Kathmandu, Nepal)	final	Code / Report

Student Id: 2431342
Student Name: Swoyam Pokharel
Section: L4CG26
Module Leader: Subash Bista
Tutor: Subash Bista

Submitted on: 12-2-2025

Table Of Contents

1. Introduction

This coursework is focused on developing a system that manages competitor information, including their personal details and scores. The objective of the assignment was to design and implement a Competitor class that interacts with a MySQL database, handles score calculations, and produces reports based on this data.

2. Methodology

I followed the Waterfall Software Development Lifecycle (SDLC) for this project. The project started with requirements gathering, followed by system design, implementation, and testing. The tools used during the development include:

- Java for the application logic.

- MySQL for the database.

- JDBC for connecting the application with the database.

- Neovim as the editor for coding.

3. Implementation

3.1. Competitor Class Development

The Competitor class was developed to handle a competitor's personal details such as name, level, age, and an array of scores. The main logic of this class includes:

- getOverallScore(): Calculates the overall score of a competitor based on the mean of the scores array.

- getFullDetails() and getShortDetails(): Return the competitor's details in full or short format.

- Name handling: The name is stored in a `Name` class, which splits the full name into first and last names.

- The attribute `scores` was chosen to store the competitor's performance in various rounds. The class ensures that a maximum of 5 scores are stored.

3.2. MYSQL and Arrays

The Competitor class was updated to handle an array of scores with a fixed size of 5. The `getOverallScore()` method calculates the mean score from this array.

- The database schema for the **Competitors** table includes:

  - CompetitorID (Primary Key)

  - Name, a string

  - Level, a string (either Beginner or Intermediate or Expert)

  - Age an int between 10 and 100

  - Scores (score1, score2, score3, score4, score5) array representing user's scores.

3.3. MySQL Integration and Reports

The integration between the classes and MySQL database was handled by the DB_API class. The class uses JDBC to interact with the MySQL database, allowing CRUD operations.

Reports are generated by the CompetitorList class, which stores a list of competitors and formats their data into reports. The data is fetched from the database and displayed using methods like `getFullDetails()` and `getShortDetails()`.

3.4. Error Handling

Error handling was implemented to catch potential issues with invalid inputs and MySQL connection problems:

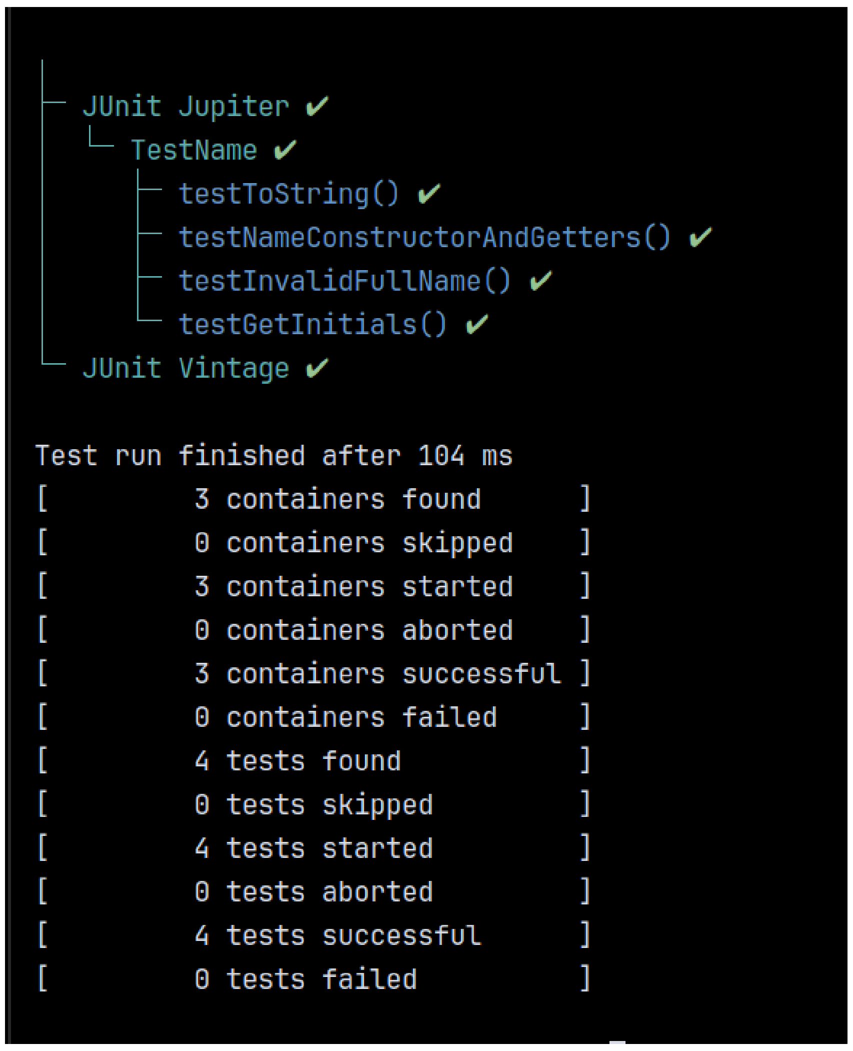
- Input validation is done to check for invalid or empty fields in the competitor's data.

- Database connection issues are caught with try-catch blocks, ensuring the application handles SQL exceptions gracefully.

3.5. Testing

Unit tests were written for the following methods:

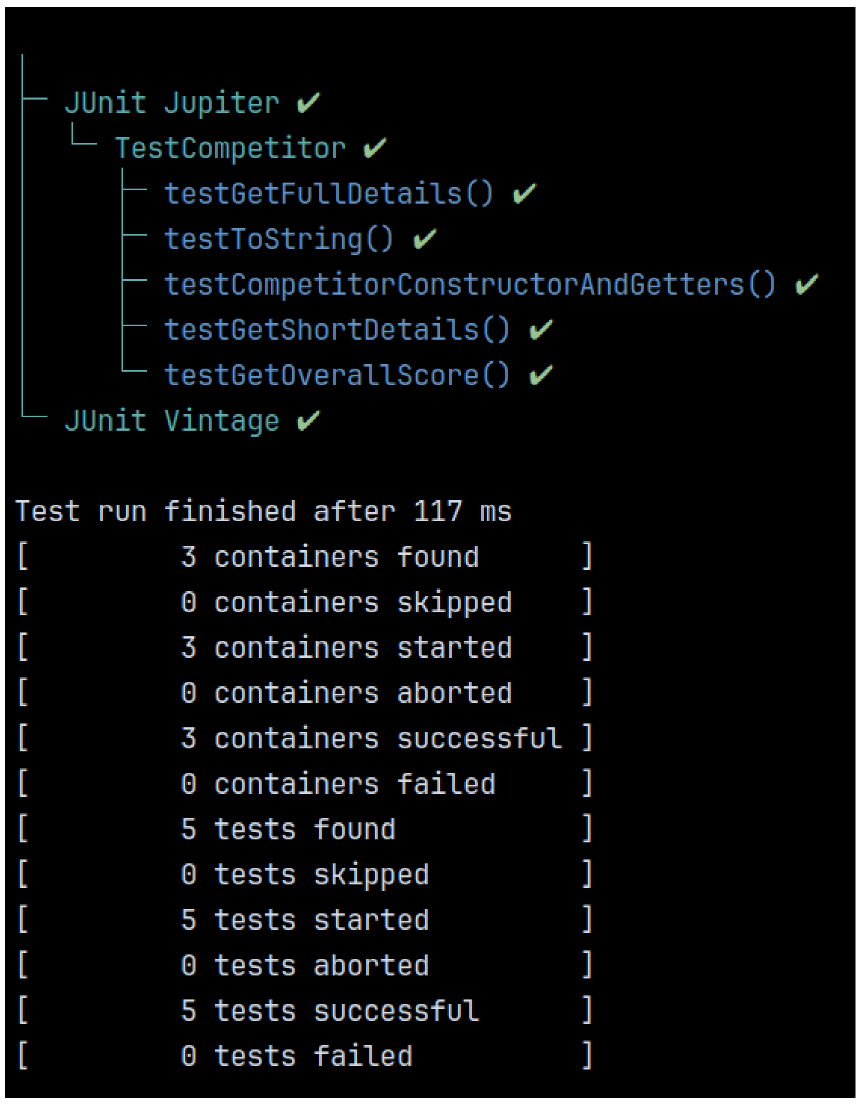
For `Name`:



```
├─ JUnit Jupiter ✔
│  └─ TestName ✔
│     ├─ testToString() ✔
│     ├─ testNameConstructorAndGetters() ✔
│     ├─ testInvalidFullName() ✔
│     └─ testGetInitials() ✔
└─ JUnit Vintage ✔

Test run finished after 104 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         4 tests found           ]
[         0 tests skipped         ]
[         4 tests started         ]
[         0 tests aborted         ]
[         4 tests successful      ]
[         0 tests failed          ]
```

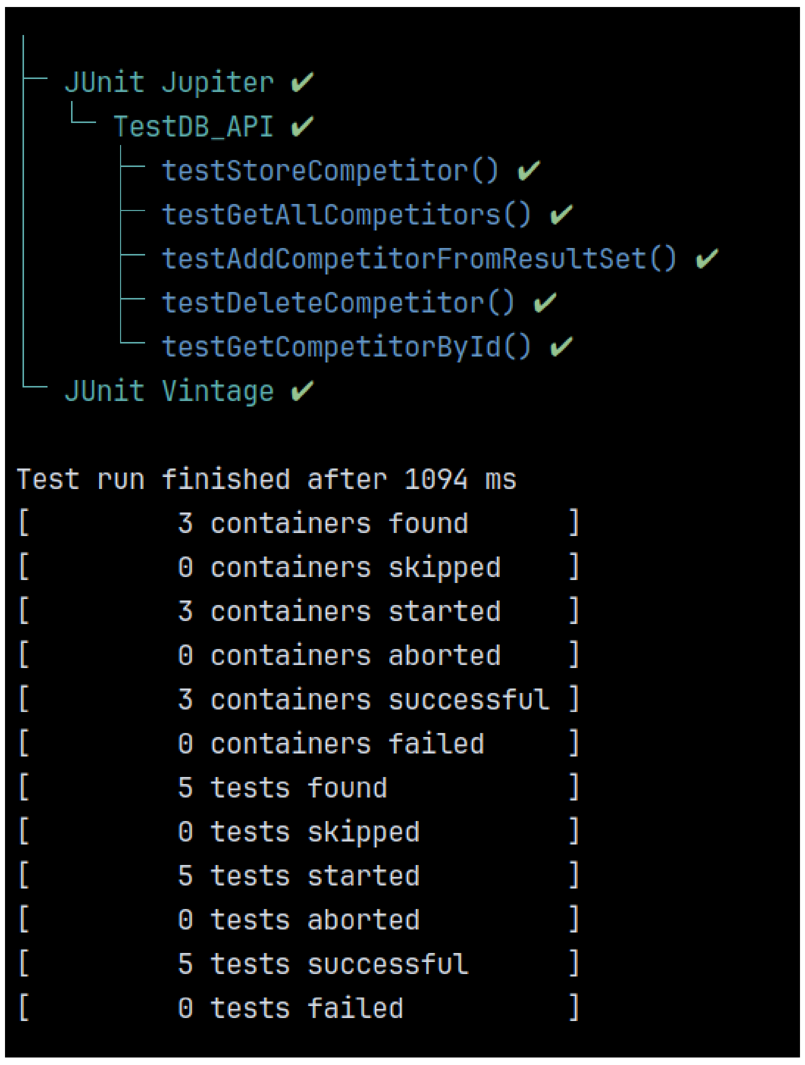
For `Competitor` and `CompetitorList`:



```
├─ JUnit Jupiter ✔
│  └─ TestCompetitor ✔
│     ├─ testGetFullDetails() ✔
│     ├─ testToString() ✔
│     ├─ testCompetitorConstructorAndGetters() ✔
│     ├─ testGetShortDetails() ✔
│     └─ testGetOverallScore() ✔
└─ JUnit Vintage ✔

Test run finished after 117 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         5 tests found           ]
[         0 tests skipped         ]
[         5 tests started         ]
[         0 tests aborted         ]
[         5 tests successful      ]
[         0 tests failed          ]
```

For `DB_API`

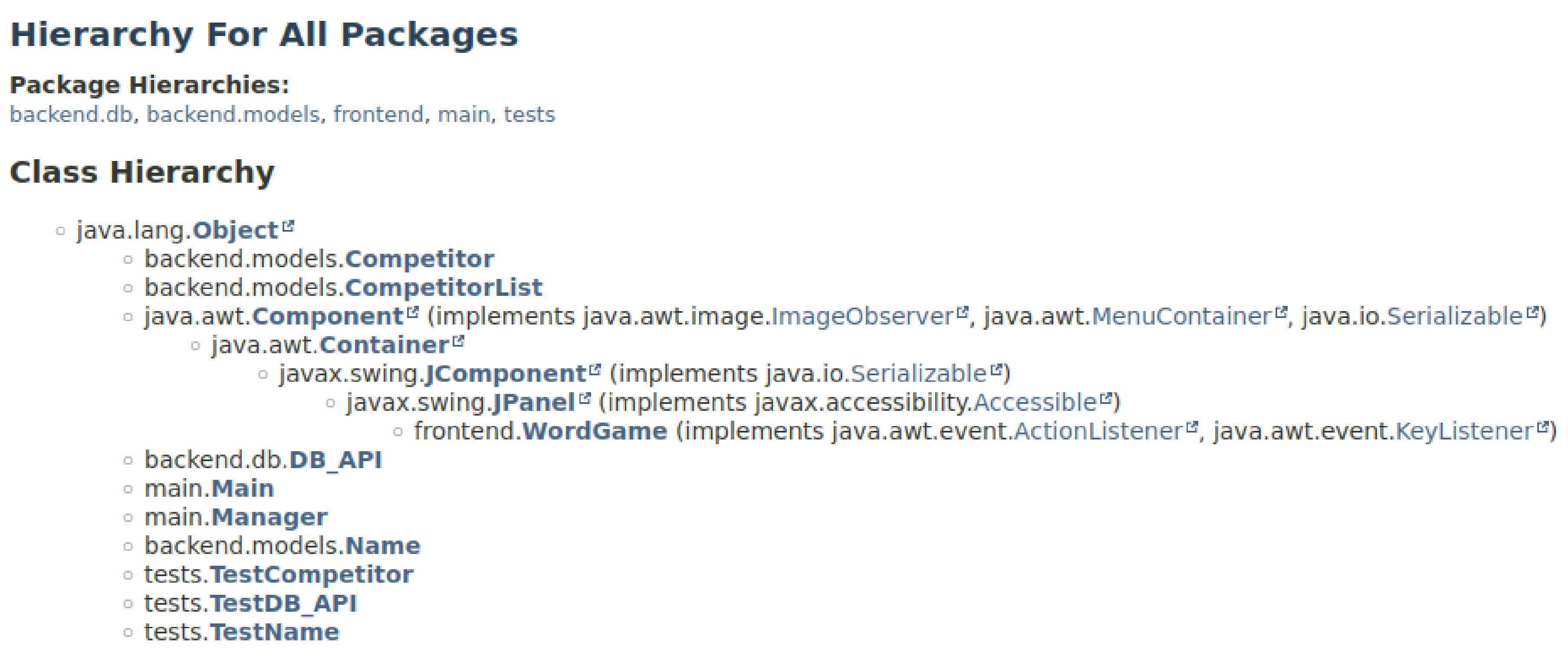


```
├─ JUnit Jupiter ✔
│  └─ TestDB_API ✔
│     ├─ testStoreCompetitor() ✔
│     ├─ testGetAllCompetitors() ✔
│     ├─ testAddCompetitorFromResultSet() ✔
│     ├─ testDeleteCompetitor() ✔
│     └─ testGetCompetitorById() ✔
└─ JUnit Vintage ✔

Test run finished after 1094 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         5 tests found           ]
[         0 tests skipped         ]
[         5 tests started         ]
[         0 tests aborted         ]
[         5 tests successful      ]
[         0 tests failed          ]
```

4. Javadoc Comments

Javadoc comments were written for the `CompetitorList` class, providing clear documentation for each method, including details on parameters and return types. This helps future developers and maintainers to understand the class's purpose and functionality.



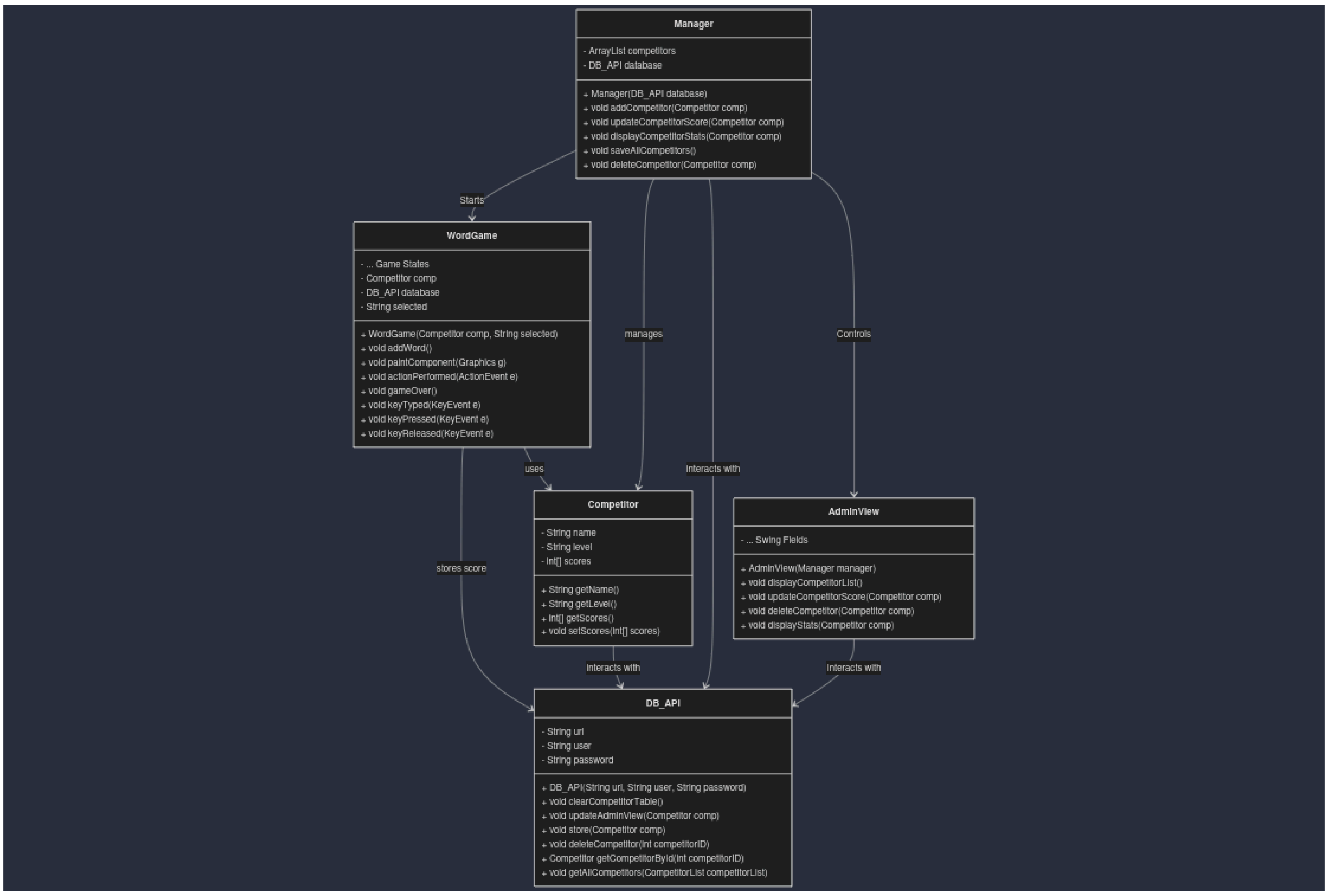
Hierarchy For All Packages

Package Hierarchies:
backend.db, backend.models, frontend, main, tests

Class Hierarchy

- java.lang.**Object**
  - backend.models.**Competitor**
  - backend.models.**CompetitorList**
  - java.awt.**Component** (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - java.awt.**Container**
      - javax.swing.**JComponent** (implements java.io.Serializable)
        - javax.swing.**JPanel** (implements javax.accessibility.Accessible)
          - frontend.**WordGame** (implements java.awt.event.ActionListener, java.awt.event.KeyListener)
  - backend.db.**DB_API**
  - main.**Main**
  - main.**Manager**
  - backend.models.**Name**
  - tests.**TestCompetitor**
  - tests.**TestDB_API**
  - tests.**TestName**

5. Class Diagram



Manager
- ArrayList competitors
- DB_API database
+ Manager(DB_API database)
+ void addCompetitor(Competitor comp)
+ void updateCompetitorScore(Competitor comp)
+ void displayCompetitorStats(Competitor comp)
+ void saveAllCompetitors()
+ void deleteCompetitor(Competitor comp)
Starts
WordGame
- ... Game States
- Competitor comp
- DB_API database
- String selected
+ WordGame(Competitor comp, String selected)
+ void addWord()
+ void paintComponent(Graphics g)
+ void actionPerformed(ActionEvent e)
+ void gameOver()
+ void keyTyped(KeyEvent e)
+ void keyPressed(KeyEvent e)
+ void keyReleased(KeyEvent e)
manages
Controls
uses
Interacts with
Competitor
- String name
- String level
- int[] scores
+ String getName()
+ String getLevel()
+ int[] getScores()
+ void setScores(int[] scores)
AdminView
- ... Swing Fields
+ AdminView(Manager manager)
+ void displayCompetitorList()
+ void updateCompetitorScore(Competitor comp)
+ void deleteCompetitor(Competitor comp)
+ void displayStats(Competitor comp)
stores score
Interacts with
Interacts with
DB_API
- String url
- String user
- String password
+ DB_API(String url, String user, String password)
+ void clearCompetitorTable()
+ void updateAdminView(Competitor comp)
+ void store(Competitor comp)
+ void deleteCompetitor(int competitorID)
+ Competitor getCompetitorById(int competitorID)
+ void getAllCompetitors(CompetitorList competitorList)

6. Status Report

The application is **fully functional** and meets the specifications provided. All required features, including competitor management and report generation, have been successfully implemented.

7. Known Bugs and Limitations

- Limited to 5 scores: The system assumes exactly 5 scores per competitor; more dynamic score management might be a future enhancement.

- Current Test implementation for `DB_API` isn't mocked. This includes making calls to an actual database underneath, which is not industry standard.

8. Conclusion

The project successfully implements the core functionalities of competitor management, from storing data to generating reports. The use of MySQL and JDBC allows for smooth data interaction, and the error handling ensures robustness. For future improvements, I would enhance input validation and support for more flexible score management.