Object Oriented Programming
Project (10%) (CA5)

Multi-User Client-Server Application
2024
Stage 2 – Semester 2

BSc (Hons) in Computing in Games Development
BSc (Hons) in Computing in Software Development

This Project is to be completed in the designated groups.

Objectives

To complete the following:

- Design, implement and test a database-driven client-server application
- Design and implement a Data Access Layer for database access
- Implement a Multithreaded Server
- Design, develop and document a communications protocol using a sequence diagram showing all messages that pass between client and server
- Implement the exchange of data between client and server using the JSON format, and to serialize and deserialize the data as required (using GSON Parser)
- Implement the exchange of binary stream of data
- Design and implement relevant unit tests
- Demonstrate the **ongoing** use of version control (this is mandatory)
- Incorporate relevant code design patterns (such as Factory, Singleton, Command and/or Decorator)
- Incorporate Collection classes where appropriate
- Apply DRY program design principles. (DRY=Don't Repeat Yourself)
- Draw an annotated high-level Architecture diagram for your system (one page)

All of the above items should be incorporated in the implementation of the features described below and will form part of the criteria for grading.

Requirements

You are required to select a theme/topic of interest to you and to identify one or more entities/classes that are relevant to your theme. (For example – if you select Premiership Soccer as a theme, then the following entities may be relevant – Player, Match, Ground, Manager etc. Once identified, you can develop classes to represent these entities. Below, a number of required **Features** are specified. You are required to implement those features for your specific theme (system) and you are required to apply best practices in software engineering design, implementation and testing, and use of collections and structures that provide the best-practice solution when implementing a feature.

An important learning outcome for you is the completion of features by a specified deadline. Hence, there are strict deadlines for delivery at each milestone. Another important aspect in grading your work is an assessment of your understanding of the code that you have presented. You must demonstrate a clear understanding - and be able to explain your design and implementation choices in all aspects of your code. As we will not be specifying detailed implementation approaches, you should discuss your proposed approach with your team/colleagues and/or ask your lecturer for guidance as you proceed.

You must create a Repository (Repo) on GitHub or GitLab and make the Repo available to your lecturer. You must push your work to your Repo at regular intervals during the development of **each** feature and at the completion of each feature, using meaningful commit descriptions. **A poor Commit History will severely reduce your grade (see Grading Rubrik below).** Features must also be demonstrated to your lecturer in lab-based progress demo/interviews. No marks will be awarded in the absence of an interview. If a milestone/deadline is missed – the maximum available mark for the component due at that milestone will become 40%. Specific penalties described below are applicable to the Final Deadline.

Schedule of Deliverables

Active Week	**Deliverable Milestones/Deadlines**
Week 7	Deadline 1: Sunday 10th March Features 1 - 4 (Individual Repo - make available to lecturer)
Week 8	Deadline 2: Sunday 17th March Features: 5 - 6
Week 9	
Easter Reading 25-31 March	
Easter Reading 01-07 April	Deadline 3: Sunday 7th April Features: 7 & 8
Week 10	Deadline 4: Sunday 14thth April Features: 9,10,11
Week 11	Deadline 5: Sunday 21st April Feature: 12,13,14 and all other requirements. Final Submission of all requirements
Week 12	Demo/Interviews in lab.
Week 13	OOP Written Test
Terminal Exams	(Refer to DkIT Terminal Written Examinations Schedule)

Feature Specifications

Select **one** of the main entities (classes/tables) for your proposed system. (In this example we use a *Player* class, but you will name your own table appropriately). It must be a class that has **at least** one integer field, one floating point field and one String field. You class must have a field called **"id"** and it must be an **autoincrement** field of a suitably sized Integer type (INT ?).

1. Create and populate a MySQL Database Table to store your main entity. i.e. one table is required with at least 10 rows of data. You must create a ***mysqlSetup.sql*** file with the SQL required to create and populate the database table so that it can be easily recreated. When you have completed features 1-7, you can add additional related tables to improve the quality of your system and the degree of difficulty of your system.

2. Create a corresponding Data Access Layer with a DTO, a DAO and corresponding Interfaces that allow access to your database table using the structure provided in the Data Access Layer code sample. (See Features below)

3. Crete a simple **menu system** that will allow you to select the options stated below: (Commit and push each feature to your remote Repo as you develop them (2-3 commits at least per feature would be expected). Marks will be lost if a consistent and spaced history of Repo commits & pushes are not in evidence.

Basic Features (DAO Methods)

Feature 1 – Get all Entities (assuming Player entities in this example)
e.g. `getAllPlayers()` - return a **List** of all the entities and display the returned list.

Feature 2 – Find and Display (a single) Entity by Key
e.g. `getPlayerById(id)` – return a single entity (DTO) and display its contents.

Feature 3 – Delete an Entity by key
e.g. `deletePlayerById(id)` – remove specified entity from database

Feature 4 – Insert an Entity
(gather data, instantiate a Player object, pass into DAO method for insertion in DB)
e.g. `Player insertPlayer(Player p)`
return new entity (Player DTO) that **includes** the assigned auto-id.

Feature 5 – Update an existing Entity by ID
e.g. `Player updatePlayer(int id, Player p)` – executes specified updates.

Feature 6 – Get list of entities matching a filter (based on DTO object)
e.g. `findPlayersUsingFilter( playerAgeComparator )`

4. Create two methods (in a JsonConverter Class) that will convert data into **JSON** format:
   Feature 7 - Convert List of Entities to a JSON String
   e.g. `String playersListToJson( List<Player> list )`
   Feature 8 – Convert a single Entity by Key as a JSON String
   e.g. `String playerToJson( Player p )`

5. Test your features by creating a number of meaningful **JUnit** tests. (These tests form part of your submission)

Client-Server Features

The following features will allow you to integrate the components that you have completed previously with the client-server material that has been provided in class. Your Data Access Layer (DAOs) will reside on the server, and your Menu will reside on the client.

1. **Feature 9 - "Display *Entity* by Id"**
   Implement a **client-side menu** item that allows a user to select the option "Display *Entity* by ID" where the Entity is your selected entity class (e.g. Display Player by ID). The client will send a request (command) to the server, along with the user inputted identifier (ID), in accordance with your specified protocol. The server will process the request, extract the ID, and call an appropriate DAO method (on server side) to retrieve the entity by ID from the database, and instantiate a Player object. The Player object will then be serialized into JSON representation, and sent from server to client via a socket stream. The client will receive the JSON data, parse the data and instantiate and populate an entity object with the data. The data will then be retrieved from the entity object and displayed on the client screen in a neatly formatted manner.

2. **Feature 10 - "Display all *Entities*"**
   Implement a client-side menu option "Display all Entities" that will send a request to the server to obtain a list of all entities. The server will process the request (command) and will use a DAO method to retrieve all entities. This list will be passed to a method that will convert it to JSON format (array of objects), and return the JSON String. The JSON String will be sent from server to client via socket. The client will parse the received JSON data and use it to populate a List of entities. All entities will be displayed and neatly formatted from the list of entity objects.

3. **Feature 11 – "Add an Entity"**
   Implement a client-side menu item that will allow the user to input data for an entity, serialize the data into a JSON formatted request and send the JSON request to the server. The server will extract the received JSON data and add the entity details to the database using a relevant DAO method (INSERT), and will send a success/failure response to the client. On successful insertion, the response will

return the Entity object (as JSON) data incorporating the newly allocated ID (if the ID was auto generated). This will be sent from server to client, and the client will display the newly added entity, along with its auto generated ID. If the insert fails, and appropriate error (as JSON) should be returned to the client and an appropriate client-side message displayed for the user.

4. **Feature 12** – **"Delete Entity by ID"**
   In a manner similar to above, provide a menu item that will delete an entity by ID, send a command (as JSON) to the server to undertake the delete, and display an appropriate message on the client.

5. **Feature 13 – "Get Images List"**
   The client menu will provide a menu item that will allow the user to request a list of image file names available for download from the server. This list of image file names is transferred in JSON format. The client user can then select one image and a request for this image will be sent to the server. The server will read the request and then opens the file and sends the file as a binary stream to the client. The client reads the binary stream, using buffering, and writes the data to a local file (on client side). [Note, we can't display the image as we are using only console display. JavaFX could be used to create a Client GUI].

6. **Feature 14 – Exit**
   Notify the server that this client is quitting and terminate this client.

Git Commits and GitHub

- Each student must maintain their own GitHub Repo for the project.
- Use a repo name in this format "**CA5-OOP-Surname-Firstname"**
  (A shared GitHub repo may be used by team for development purposes)
- When a feature/code is written by a student it must be identified as the work of the student by a comment inserted before the code. Secondary contributors can be added also (see below)
- 'Feature' commits must be made before the relevant deadlines.
- The history of Git Commits must be available on GitHub for your lecturer to inspect.

```
/**
 * Main author:             Alex Kane
 * Other contributors:      Bill Knight
 *
 */
```

This means that each student's codebase will contain code that they wrote **and** code that their teammates wrote. However, each student must understand ALL of the code in their codebase and be able to answer interview questions relating to the code to demonstrate their understanding.

Upload Requirements

Upload to contain:

1. **Screencast** showing your app working with two clients AND a walkthrough of your code identifying the core functionality and the data structures used. Please state your names and class at the beginning. Screencast to be no longer than 5 minutes.

2. Zipped file containing all **source code** and **data files.** (Repo URL alone **is not acceptable**)
   ZIP the project folder for upload. Please name your zipped project folder :
   “CA5_Lastname_Firstname”
3. **Completed CA cover sheet**, sign and upload an electronic copy with your project.

Upload completed project as a single zipped file to Moodle by the deadline.

Students will demonstrate their work and each student will be interviewed on ALL aspects of the project.

Late Submission Penalties

The following late submission penalties will apply:

- The marks awarded to the assessment element will be reduced by 20% for material submitted after the deadline and within 24hrs of the deadline.
- The marks awarded to the assessment element will be capped at 40% for material submitted after (the deadline + 24hrs), and before 72hrs after the deadline.
- Assessment material submitted more than 72hours after the deadline will be given a mark of zero.

In cases of **plagiarism**, zero marks will be awarded in this assignment and standard DkIT policies will apply.

Standard DkIT policies will apply in relation to legitimate verifiable absence from assessment or late submission of assessment.