Object Oriented Programming
Project (20%) (CA6)

Database-Driven Client-Server Application
2023
Stage 2 – Semester 2

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

The OOP Semester 2 Project is worth 20% of the year-long module marks (equivalent to 40% of a semester-based module). Refer to Moodle for deadlines and upload links.

This Project is to be completed individually.

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)
- Design and implement relevant unit tests
- Demonstrate the **ongoing** use of version control
- 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 must 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 to discuss your approach with your 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**. 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

Week Beginning	Deliverable Milestones/Deadlines
Week 9 Monday 20th March	Deadline 1: Sunday 26th March Features 1 - 4 (set up project on shared Repo make available to lecturer)
Week 10 Monday 28th March	Deadline 2: Thursday 30th March Features: 5 - 6
Easter Reading 1 Monday 7 March	
Easter Reading 2 14 – 16 March	
Week 11 Monday 17th April	Deadline 3: Thursday 20th April Features: 7-8
Week 12 Monday 24th Apr	Feature 9: (Note: Practical Lab test also takes place this week. Open Book)
Week 13 Monday 1st May	Deadline: Thursday 3rd May Feature 10 - 12 Demo/Interviews Final Submission of all requirements
Study Week	
Terminal Exams	(Refer to DkIT Terminal Written Examinations Scheduel)

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 of 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 following options:
   (Commit and push each feature to your remote Repo as you develop them (2-3 commits at least per feature). Marks will be lost if a consistent and spaced history of Repo commits & pushes are not in evidence.

   Feature 1 – Find all Entities e.g. *findAllPlayers*() to return a List of all the entities and display that list.
   Feature 2 – Find and Display (a single) Entity by Key e.g. *findPlayerById(id)* – return a single entity and display its contents.
   Feature 3 – Delete an Entity by key e.g. *deletePlayerById(id)* – remove entity from database
   Feature 4 – Insert an Entity in the database using DAO (gather data from user, store in DTO object, pass into method *insertPlayer (Player)*, return new entity including assigned auto-id.
   Feature 5 – List entities using a filter e.g. *findPlayerUsingFilter(playerAgeComparator)*
   Feature 6 – Create a Cache using a *HashSet* that will maintain the **id**s of all players and refactor the *findPlayerById*() method so that it checks for the existence of a player id before it makes a query to the SQL database.

4. Create two DAO methods that will return data in JSON format:
   Feature 7 - Retrieve all Entities as a JSON String e.g. *findAllPlayersJson()*
   Feature 8 – Find a single Entity by Key as a JSON String e.g. *findPlayerByIdJson(id) [consider what to return if there is no match?]*

5. Test your features using **JUnit** tests.

Client-Server Features

The following features will allow you to integrate the components that you have completed previously and the client-server material that has been provided in class.

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. Find Student by Student 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, use an appropriate DAO (on server side) to retrieve the entity from the database, convert the entity into JSON representation, and return the JSON via a socket to the client. 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 to retrieve all entities, convert to JSON format, and return the JSON data to the client. The client will parse the 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 JSON formatted request and send the JSON request to the server. The server will add the entity to the database using a relevant DAO, and will send a response to the client. The response will return the Entity object data incorporating the newly allocated ID (if the ID was auto generated). The client will display the newly added entity, along with its auto generated ID. If the insert fails, and appropriate error should be returned and displayed.

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 to the server to undertake the delete, and display an appropriate message on the client.

Upload Requirements

Upload to contain:

1. **Screencast** showing your app working AND a walkthrough of your code identifying the core functionality and the data structures used. Please state your name 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 : "Lastname_Firstname_CA6"
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.

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 72hour 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.