**ECM1410 Pair Programming Coursework**

**(70% of module grade)**

**deadline: 12 midday 24th March 2025**

This assessment covers the use and implementation of a range of object-oriented concepts using the Java programming language as covered in ECM1410.

The assignment is summative. Please ensure you read the entire document before you begin the assessment, and monitor ELE for any additional guidance/instructions that may be provided to assist you in your tasks.

## 1. A Note on Pair Programming

In addition to developing and demonstrating the skills in java and OOP techniques you have been learning in this module, this assessment is designed to give you experience of pair programming, helping develop wider skills in collaboration and communication.

You are expected to code all parts of your submission together, and each pair will be awarded a single mark. As in a workplace you are expected to respect your partner and

co-operate to solve and overcome any issues that may arise.

The module leader reserves the right to split pairs where one student is not engaging with the coursework. The coordinator also reserves the right to assign non-contributing students a mark of 0. In the rare situation that you are paired with a student who is not contributing (e.g. not replying to emails and/or not meeting up for pair-programming sessions) you must inform the teaching staff of the situation within one week of release.

Both parties of a split pair will be assigned an individual variant of the coursework (however, if there are multiple pairs in this situation it may be possible to reform pairs consisting of participating students, and of non-participating students). It is not permitted for a student working on the individual variant of the coursework to collaborate with any other student.

## 2. Scope

This assignment is imagines you work for a company who have accepted a contract to write the back end of a Java application.

The system's required functionality has already been determined, and the front end that manages the UI is being developed separately and does not need to be coded by you.

The client has specified the structure of an interface containing the methods needed by the user interface to operate the system. Your job is to design the backend in java using appropriate object orientated programming principles, and the corresponding methods as defined in the interface.

You must package and build your backend into a jar file, which could be used by a front end that communicates in accordance with the interface design to form a fully functional solution.

**3. Task**

The phenomenon of Sudoku and Wordle show that there is a strong appetite amongst the public for recreational puzzle solving, and such games have proved increasingly popular with the public who use their mobile devices for social and casual gaming.

One of the key drivers for the viral growth of Wordle was that users are encouraged to share results and compete within their social circles.

In this coursework you must develop a Java application to demonstrate and trial a system that allows social groups to set up, manage and participate in competitive leagues to keep track of the scores of games played.

Each league will be associated with a daily solo challenge/task which each user can complete each day. Once all player results have been registered with your system, the system will collate the results for presentation in a leagues tables.

Users should be able to view and navigate:

i) the results of each round (i.e. browse daily results of all players in the league)

ii) the weekly league table (scored from Monday to Sunday)

iii) the Monthly league table (scored by calendar month)

iv) the Yearly league table (scores added from 1st Jan to the end of the year).

Players may be members of multiple leagues, but each league is associated with a single daily game.

It is envisaged the system will be able to manage multiple games-types such as Wordle, Nerdle or a daily crossword.

You should allow each league to make use of different scoring options for a given game, for example for the defined games DICEROLL and WORDMASTER, DICEROLL should rank players by lowest score, and WORDMASTER rank by highest score.

A real world implementation (e.g. as an Android phone application) might use java for local processing but interface with an online database to retrieve and save data. However this is beyond the scope of the module and should not be implemented here (although you may wish to consider how best to structure the code so that it could be adapted into this form).

For the purpose of this coursework you should instead work in a simple system whereby we store our objects (relating to user/league/game data) locally in memory as lists (e.g. `ArrayList` or `LinkedList`).

Methods should be in place to serialise the system data to load/save the system state to a local file, so that on opening the program the current data can be loaded, and the updated data can be saved to disk when the user selects to terminate the program.

You should structure the system to make use of OOP (object orientated programming) principles such that the code base can be extended and rebuilt with additional game types.

**3. Marking Criteria**

**Documentation and commenting ( /15 )**

In all code files you should ensure that through commenting and organisation your code is sufficiently readable so that a fellow UG student taking the module would be able to follow your work. Where you have made use of code snippets taken from an internet source you should add a comment showing the web link.

All classes and methods should be annotated so that documentation can be automatically generated by javadoc, and you should build this documentation, placing it into a docs folder in your project repository. It must not be publicly available.

**Java conventions ( /5 )**

The degree of adherence to Java naming conventions and format- ting. See lecture notes and e.g. https://google.github.io/ styleguide/javaguide.html

**Repository & Project Management ( /10 )**

Your GitHub repository will be examined. Marks will be awarded for: code organisation ; evidence of regular commits and good time management; documentation of development (i.e. commit history); correct privacy settings (you should not change repository permissions to make it publicly viewable).

**OO design and implementation (/20)**

Marks will be awarded based on following good OO design practises.

To facilitate this you must submit a diagram detailing the class structure of the objects you have created. This can be in UML either constructed from your code automatically or made manually. For each class, include all attributes and public methods. There is no need to include the interface or exception classes. You can use any software to create your diagram word, powerpoint, drawio, etc. but must export it to PDF format and commit to your code repository.

**Operation (/20)**

The degree to which the class operates as required, as supported by the package members. Submission of a jar file that cannot be compiled in with the test code (due to e.g. the interface definition being changed, required package members missing, etc.) will receive an operation mark of 0. Operations are assessed by the delivered functionality. For instance, if the system fails to create a player it will necessarily fail most other functionalities as one would not be able to create leagues nor results.

**Demonstration (/30)**

You should include in your repository a program DemoApp.java which demonstrates the software in action. This should be sufficient to test all classes and methods that have been requested. You should include instructions to run the DemoApp.java in your projects README.md file. The grade for the demonstration will be based on the coverage of methods demonstrated and the requirement that the program prints a sufficient output to allow the marker to follow the stages in the demonstration.

**Penalties**

- Use of non-permitted packages (-10)

- Non-submission of coversheet (-5)

-Making your submission publicly available on GitHub (-10)

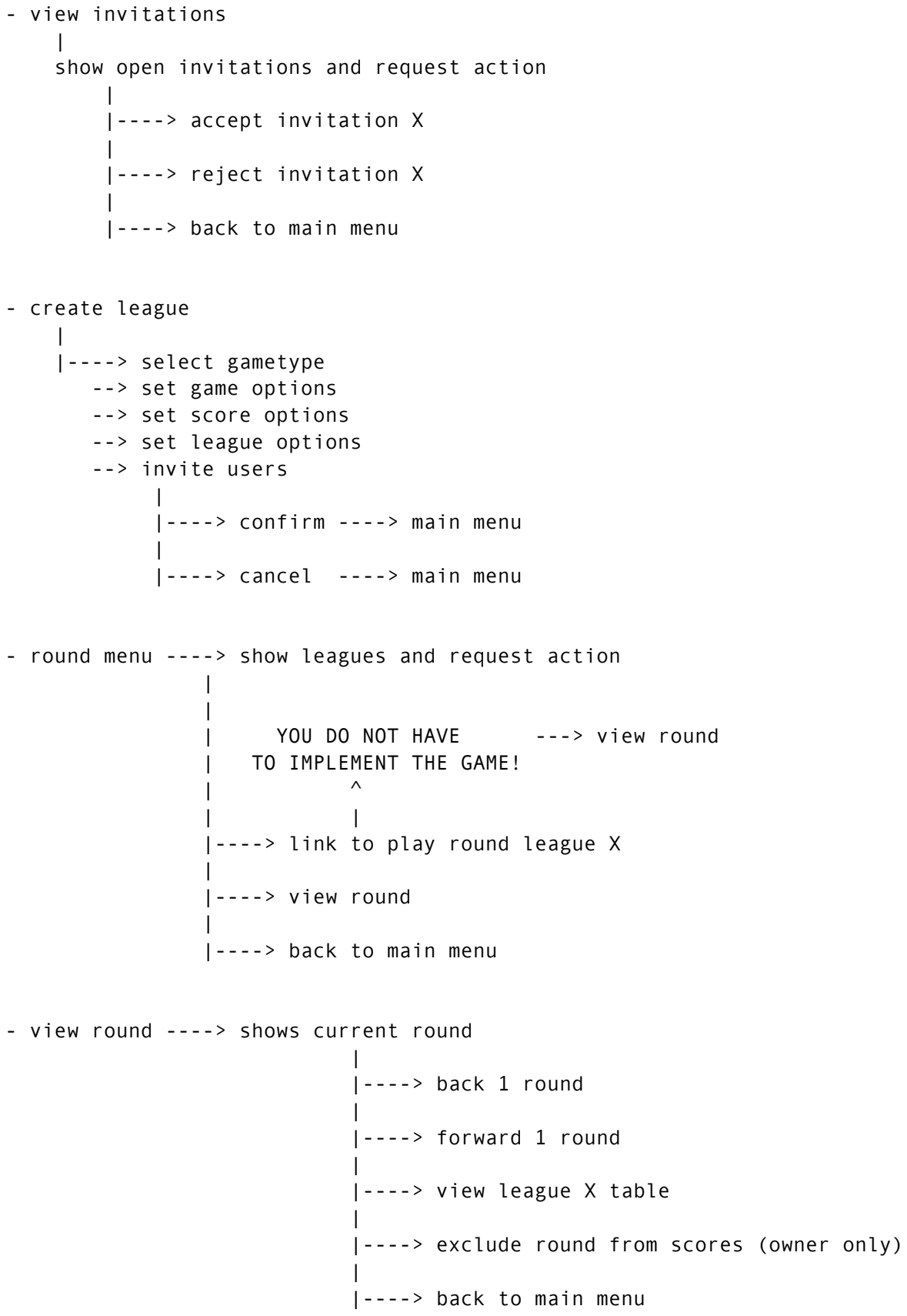
**4. Requirements**

You must develop and design an suitable set of classes and methods using OOP principles in line with the exact design specified in file GameLeaguesApp.java.

A UI (detailed in section 5) must be able use this interface without changes. (Otherwise you will not achieve all marks in the operations)

## 5. User Interface diagram

**(You do not need to implement this but it is to provide an idea of how the overall system will function).**


- view invitations
show open invitations and request action
|----> accept invitation X
|----> reject invitation X
|----> back to main menu
- create league
|----> select gametype
--> set game options
--> set score options
--> set league options
--> invite users
|----> confirm ----> main menu
|----> cancel ----> main menu
- round menu ----> show leagues and request action
YOU DO NOT HAVE ---> view round
TO IMPLEMENT THE GAME!
|----> link to play round league X
|----> view round
|----> back to main menu
- view round ----> shows current round
|----> back 1 round
|----> forward 1 round
|----> view league X table
|----> exclude round from scores (owner only)
|----> back to main menu

```
- view league X table ---> shows current league table
          |
          |----> back
          |
          |----> forward
          |
          |----> toggle timespan (week / month / all-time)
          |
          |----> view league X (current round)
          |
          |----> back to main


- league management ----> select league (show all owned)
                                |
                                |----> start league
                                |
                                |----> finish league
                                |
                                |----> reset league
                                |
                                |----> send invite
                                |
                                |----> remove player
                                |
                                |----> add owner
                                |
                                |----> delete owner
                                |
                                |----> delete league
                                |
                                |----> back to main


- account management
    |
    |----> change display name
    |
    |----> change email
    |
    |----> change privacy setting
    |
    |----> leave league
    |
    |----> delete account
    |
    |----> back to main
```

**5 Advice**

1. Do not jump straight into the coding: take time to consider the design of your solution first. Think about the objects that you will use, the data they will contain, what the methods they should provide are (in addition to those mandated via the interface), how they relate to one another, etc. Once you are happy with your design, then start programming. Don't be afraid to reassess your design as you go through, but check on the implications of making a changes on all the other objects in your system that use the changed part.
2. Check your objects behave as you intend — use a testing application and use assertions.
3. Slowly fill out functionality — it is far better to submit a solution that supplies most but not all of the required operations correctly, rather than one that doesn't provide any/doesn't compile, as a submission which does not provide any correct functionality at all will get a 0 for the operation criteria. Start off with a class that compiles and slowly (incrementally) add functionality.
4. Learn how to work with git to manage versions. Keep copies of your working code. If the worst happens and you had a version that worked on 50% of the operations and you've made changes that seem to have broken everything, it is useful to be able to 'roll-back' to the earlier version and try again.
5. Do not change the interface and classes defined in the requirements. If you change them, the markers will not be able to compile my codebase with your submission, and you will receive an 'Operation' component mark of 0, as the interface will not be able to connect to the front-end of the system.
6. Practice creating the jar file