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 the instruction repository and ELE for any additional guidance/instructions that may be provided to assist you in your tasks.

1. Scope

This assignment supposes 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 user interface is being developed separately and does not need to be coded by you.

However so that the back end can be utilised the client has specified the structure of an interface containing the methods you must implement exactly so that it is compatible with the user interface.

In this assignment your task is to design and code 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, so that it can form a complete system when integrated with the corresponding front end that interacts in accordance with the interface design.

Your company have requested you to collaborate with a partner on this project and complete all coding using pair programming.

2. Pair Programming

In addition to developing and demonstrating the skills in java and OOP techniques you have been learning in ECM1410, this assessment is designed to give you experience of pair programming, helping develop wider skills in collaboration and communication.

This is a popular development approach, primarily associated with agile software development, and is used across the software industry. In pair programming, two software developers work together, at the same time, on a single machine, to generate the solution to a given problem. One (the driver) physically writes the code while the other (the observer) reviews each line of code as it is generated. Normally, the pair would be side-by-side in the same physical machine, but when this is not practical it is possible to work remotely by sharing the IDE screen.

During development the driver/observer roles should switch frequently. The aim of the split role is for the two programmers to concern themselves with different aspects of the software being developed, with the observer considering the strategic direction of the work (how it fits with the whole, and the deliverables), and the driver principally focused on tactical aspects of the current task at hand (block, method, class, etc.), as well as allowing useful discussion between the developers regarding different possible solutions and design approaches.

Research into pair programming has indicated that it leads to fewer bugs and more concise (i.e., shorter) programs than development by a single programmer, and achieves this in a shorter period of time (albeit with slightly more total developer time, due to two developers being involved at once).1 Additionally, it also facilitates knowledge sharing between developers, which can be crucial for a software house, and is important for the continued learning of the developers involved. Often in industry pairs are generated by cycling through developers from a larger team, so everyone is eventually paired with everyone else at some point.

Notes:

- 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 cooperate 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.

Raising Issues
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 by 5pm on March 4th.***

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.

3. Overview of Task

Context

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 the ease with which users could share their game result and compete within their social circles.

In this coursework you must develop a working Java **gamesleague** package to demonstrate and trial a system that allows groups of players to set up, manage and participate in competitive leagues, where the results of daily games are logged, and collated into league tables.

Each league will be associated with a daily solo challenge/task which each player can complete each day with the outcome being a game score for each player.

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

Once all player results in a league have been registered with your system, the system will allocate league points and collate the results. Player results will be stored so that the following league results can be obtained:

- round scores and points (i.e. daily results for the league)
- weekly league points (ranked by total points from Mon to Sun)
- monthly league points (ranked by total points over calendar month)
- yearly league points (ranked by total points from 1st Jan to end of the year).

The complete **gamesleague** java package will comprise of:

i. Front end code. (You do not need to code this!)
This is the code to run a user interface for creating player accounts, managing leagues and viewing results.

ii. Back end code. (This is the task assigned to you and your partner!)
This will implement a specified interface (see **GamesLeagueInterface.java**) by which the front end can enter and retrieve system data.
Your back end should include a **GamesLeague** class that implements the interface, and defines appropriate OOP object classes to managing, store and process the system data.

Use-case

It is envisaged that this demo system will be set up to run on a single computer in the Computer Science staff coffee area, so that staff players can log into the game application, play their daily games, and view their results and league tables.

This means that for the purposes of this task you **do not need** to consider the situation where multiple simultaneous users/operations are possible and the associated issues of system integrity.

Game Info

For the purposes of this demonstration you should allow the leagues to be associated from one of two games DICEROLL and WORDMASTER.

This assessment does not require you to code the actual games (although you will have the opportunity to do this as a workshop task) or indeed have any knowledge of the gameplay.

However you do need to take into account that:

- in the DICEROLL game the player with the lowest game score wins (i.e. ranking is in order of score low to high). The player(s) in the group with the lowest game score are awarded 3 league points, and the player(s) in the group with the next lowest score are awarded 1 league point.

- in the WORDMASTER game the player with the highest score wins (i.e. ranking is in order of score high to low). For each round (day) players are awarded a number of league points matching their game score.

For the purpose of this coursework you should work to develop a system in accordance with OOP principles, so that the GamesLeague class stores system data is in collections (e.g. ArrayLists) of appropriate objectswhich can be serialised to/from disk so the system state can persist when the application is closed and reopened.

In addition to meeting the requirements and functionality the following additional considerations will be considered when grading the OO design.

- is the program structured such that the code base can be extended and rebuilt with additional game types or league scoring systems.

- does the program run efficiently such that the weekly/monthly/yearly league results do not need to be recalculated each time the league data is requested.

4. Repository Resources and Submission

Resources

You will be provided with a GitHub classroom assignment repository that is preconfigured with the project structure. **Only one student should accept the GitHub classroom assignment, and they can then invite their partner as a collaborator on the repository.** The repository provided will contain a source code directory that contains the initial code for the **gamesleague** package, including java interface definition **GamesLeagueInterface**. You must not edit this file, and you must implement it as a class called **GamesLeague**.

Prior to the submission deadline you will be given access to resources for testing and validating your code structure. The purpose of these is ensure that you can be confident your code is compatible with that used to perform automatic grading of your submission. However these tests will not test the full functionality of your package and you should develop and make use of your own test application code that checks your coded functionality implements fully the methods and requirements specified in the **GamesLeagueInterface** documentation.

Submission

You must ensure that at the coursework deadline you have finalised your repository, committed the latest version and pushed it to the GitHub classroom. For marking you do not require to upload anything to ELE, as the markers will have access to a snapshot of your repository at the timepoint of the coursework deadline. Should you wish to make a late submission or are applying for mitigation you must communicate this to the coursework organiser (p.lewis2@exeter.ac.uk via email).

The following must be included in your repository for grading:

- **CoverSheet.md** file that lists you and your partners candidate numbers (do not include your actual names) and a short summary declaring any usage of GenAI
- PDF named **GamesLeagueClasses.pdf** that displays a diagram of the class structures you have desgined (see marking criteria for more information)
- **docs** folder that contains the javadoc **.html** files documenting your package
- **src** folder containing your **.java** files (one file per class) including the unaltered resource files provided (i.e. GamesLeagueInterface.java and exception files).
- **bin** folder containing the java **.class** binaries compiled from your .java files
- **res** folder containing any resources your code needs to run
- **build** folder containing the jar file collecting your **gamesleague** package code

Guidance and commands for building and running the java code when organised in this structure will be provided with the Week 7 lectures.

5. Academic conduct and Use of GenAI

The usual regulations for student academic conduct (e.g. with regards to plagarism or contract cheating) apply to this assignment

The assignment has been designed so it can be completed without GenAI and it is the opinion of the coursework organiser that minimising your use of GenAI is the best way to ensure that you develop as programmers and ensure your grade for this and your future academic work is maximised.

However we acknowledge that AI is ever more integrated with IDEs and search engines it is not practical to run assignments as being fully AI prohibited.

Therefore **this assignment is classed as AI-supported** – where ethical and responsible use of GenAI tools in the development of an assessment is allowed.

Students are expected to self report how they as a pair made use of GenAI tools in a short statement in the submission CoverSheet.md file.

Appropriate usage might include using GenAI:

- as a chatbot for accessing information on java coding syntax, libraries and examples.
- generating trivial boiler plate code such as templating the getter and setter methods for a class.
- explaining errors you are experiencing with your code.

Inappropriate usage would include:

- pasting parts of the report brief / specification interface into GenAI to generate solution code.
- utilising non-trivial code sections authored by GenAI without attribution (either in the code file, or in their GenAI statement).
- making use of GenAI with the end result that you cannot understand/explain/justify your code design and operation.

It is your responsibility to ensure that the code you and your partner submit reflects your coding abilities and understanding. Where a marker has cause to feel that students have submitted code they may not understand and/or GenAI has been used inappropriately students may be invited to a viva to demonstrate their understanding, and marks for design and operation may be reduced if students cannot explain their work satisfactorily.

6. 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 formatting. 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, code testing and good time management (e.g. evidence in commits of 2-3 coding sessions per week over a 2-4 week period); documentation of development (i.e. commit history with useful commit messages); correct privacy settings (you should not change repository permissions to make it publicly viewable).

OO design and implementation (/30)
Marks will be awarded based on following good OO design principles that are appropriate to the project, and how these are implemented in your code files. Overly complex structures will be penalised. **You must submit a diagram detailing the class structure of the objects you have created. This should be drawn so that the marker can easily understand the components of your system and how they are related.** This can be in UML either constructed from your code automatically or drawn manually. For each class, all attributes and public methods must be included. 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 (/40)
The degree to which the class operates as required, as supported by the package members. Prior to the submission deadline you will be given a testing script which will allow you to self-test that your class is compatible with the automated testing program, Submission of a jar file that cannot be compiled and utilised by the automated testing program (due to e.g. the interface definition being changed, required package members missing, etc.) may receive an operation mark of 0, and it is your responsibility to test your repository files before submission.

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 store results if players have not been created.

Penalties

- Use of non-permitted packages (-10) *ask coursework organiser if you are unsure*
- Non-submission of (-5)
- Making your submission publicly available on GitHub (-10)

7. Timeline

The following is a suggested schedule of work.

Week 7 (Feb 25th - Mar 3rd) - analysis and initial work
Break the problem down with your partner, consider the different aspects of the problem and the objects you will need, and what their attributes and methods should be. Agree on the development environment to use, and what your regular pair-programming slots will be and block out these times in your calendars going forward.

Think about what supporting object classes need to be designed and how the **GamesLeague** class methods will make use of them. You can try to start working on writing and testing the supporting classes.

In the lectures you will cover interfaces, packages, jar files, design by contract and assertions, so by the end of the week you should have the knowledge required to complete the majority of the assignment.

Week 8 (Mar 4th - Mar 10th) - begin implementation
Continue to develop the supporting classes and test their functionality. Start work on the **GamesLeague** class implementation. As you are working include in your repository test code e.g. TestGamesLeagueApp.java that you can use to validate the code you write functions as expected. As a target aim to complete the methods for creating and managing leagues.

In the workshop you will be practise working, organising and packaging java projects, and you should practise constructing the jar file for the coursework repository, and check its contents.

In the lectures you will cover Serialisation and IO, and can start then start work on the interface methods to load/save the system state.

Week 9 (Mar 11th - Mar 17th) - complete implementation
Continue with implementing the specified interface methods in your GamesLeague class. After completing the methods for managing players and leagues (and testing your code) work on the code that accepts and processes the scores and league points. Check you can run the commands and correctly building and save the documentation.

Ensure that when testing your code you include tests that run using the defined interface functions than your own class methods, as the automated marking code will test functionality via these.

When the above is complete work on the final step of serialisation to save/load the system state to/from a file.

Week 10 (Mar 18th - Mar 24th) - test system, complete supporting documentation
When your code is nearing completion and the deadline is approaching make sure you commit regularly.

Produce a final class diagram. Final testing and fixes. Fill in the necessary annotations and comments for building the java documentation ensuring submission in good time for the deadline. Build and test your final code. Build your jar file, check its contents, and that it is compatible with the testing script provided.

8. Advice:

- 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.
- Check your objects behave as you intend — use a testing application that tests the operation of your code.
- 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.
- 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.
- 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.
- Practice creating the jar file if you leave this until the last minute and cannot submit a file with valid structure the automated test code will not be able to load and run, and you will get a 0 for the operation criteria.