COMP1010 Major Assignment

Your assignment is composed of three parts which contribute a total of 30% to your unit marks:

* **COMPONENT A**: 1/3rd (10%) of the marks will be based on your submission against the requirements that follow, including a responsibilities document that all group members sign-off on. This will include an estimate percentage of overall work that each student did. The percentage work will be used to calculate the final marks of components A and B.
* **COMPONENT B**: 1/3rd (10%) of the marks will be based on your program code style, importantly including delegation and class design.
* **COMPONENT C**: 1/3rd (10%) of the marks will be based on a live coding session.

**Group size**: The assignment must be submitted in groups of 4.

**Rationale**: While we understand working in a group has its own challenges, it is a work skill that students need to start preparing for starting in the first year. This involves scheduling management, work allocation, team management (compromises, negotiations, etc), and overall project management.

Group work is an important part of many industries, especially IT and related fields. Projects can get quite big in scope and understanding how to work with one another is a big part of success. Many organisations will be looking at whether you are a team-player or not, just as much as they are looking at your code. That said, a third of the mark is individually-assessed. We also recommend meeting face-to-face and setting clear expectations before allowing others to join your group.

Group self-enrolment will open on Sunday 1st September and close on Sunday 8th September. Anyone who doesn’t have a group of 4 will be paired by the staff after that, and can exchange contact information through assignment forum. That said, it won’t be the easiest situation, and to avoid that scenario entirely, we recommend you form/join a group of 4 by end of 8th September.

Marking Criteria

Your submissions will be assessed on the following basis:

* **COMPONENT A (Up to 10 marks)**: Implementation and testing
  + Unit testing (2 marks)
    - Whether you wrote good-quality (a diverse variety of data including edge or boundary cases) unit tests for your classes or not.
  + Functionality (3 marks)
    - What does your program achieve, and what can it be used for. The extent of services/features provided by your program. Think about the quality of features, not just quantity.
  + Scope (5 marks)
    - (1 mark) At least one class should contain object(s) of another user-defined class in the project (but not an ArrayList of objects)
    - (1 mark) At least one class should contain an ArrayList of object(s) of another user-defined class in the project
    - (2 mark) At least one satisfactory instance of recursive data structure being used. You should submit either,
      * A single zip file containing two projects: one using ArrayLists, and the other using a recursive data structure (note, the second project CAN contain ArrayLists too, it just needs to have an application of recursive data structures as well), or,
      * A single zip file containing a single project, that uses both ArrayLists and recursive data structures at different places.
    - (1 mark) File I/O. Example provided with the specs and demonstrated in week 7 lecture (**studyPlanner.zip**).
* **COMPONENT B (Up to 10 marks)**: Code style and documentation
  + Commenting, Variable names, Indentation
  + Delegation, Class Design
  + Providing **README.md** file (or even a word or pdf file **inside** your project) as a part of the zip file you submit, which describes the following,
    - What problem your application solves
    - A description of the structure of your program
    - The documentation should contain clear instructions on how to run the project, in bullet points, under a section heading, titled, “How to run the program”. A 1-mark penalty applies if these instructions are not included, no matter how simple the execution steps are.
  + **MajorAssignmentCodeStyleGuide.pdf contains the rubric**
* **COMPONENT C (Up to 10 marks):** Live-coding in the form of a CodeRunner quiz based on classes and objects, ArrayLists, and recursive data structures, analogous to the ones you would have solved during the course of the assignment.

## Submission details and Checkpoints

**GRADE NOTE 1**: A single submission for the group must be made in the form of a zip file (zipped Visual Studio Code folder containing the Java project, including the documentation file) that does not have any machine-based or path-based dependencies. That is, the project must be self-contained without external dependencies. You can add JUnit jar file in the *lib* subfolder of the project folder and refer to that. Example provided with the specs and demonstrated during week 7 lecture (**studyPlanner.zip**). Note, if it works on your computer, but not lab computers, it will be treated as an invalid submission, and be given a zero. Given this notice is being provided with the release of the assignment, no exceptions shall be made. Hence, it is critical that you download your submitted file on a lab computer, unzip the file, drag and drop the folder in Visual Studio Code and run it to ensure it works. If it doesn’t work on lab computers, you won’t get any marks for components A and B. It doesn’t matter if another group member submitted on behalf of your team, the entire team will be affected by this. If you decide to submit on the due date (Sunday), check it in the labs the next day, and if it doesn’t work, submit the correct version as assignments ARE marked with a 5% penalty per day or part of. We will not entertain students asking to be marked if this rule is broken.

**GRADE NOTE 2**: An estimate of the work distribution between different group members must be noted in the document that is a part of the final submission. We understand that not everyone will do the EXACT same amount of work. But at the same time, it is not fair for a student to have done only 15% work and the other student to have done 35% of the work, and they both get the same mark. To resolve this, we’ll use the following formula:

Individual marks for each of Component A and B will be limited to:

**1.25 \* 10 \* [CONTRIBUTION\_PERCENTAGE] / 25, capped to 10**

**SHORT STORY:** As long as you do 20% or more work in a group of 4, you qualify for the entire marks your group qualifies for.

**LONG STORY** : 1.25 is to provide some lenience and wiggle room.

10 are the maximum marks for Component A and B each.

/25 is based on default number of group members (4).

[CONTIRIBUTION\_ PERCENTAGE] is between 0 and 100.

For example, if someone does 15% of the work (60% of their allocation), their mark is capped at 1.25\*10\*15/25 = 7.5

However, if someone does 20% of the work (80% of their allocation), their mark is capped at 1.25\*10\*20/25 = 10

Thus, you can see that we have provided some leeway to students.

By default, number of group members will be 4. Only in exceptional cases, where we allow less than 4 members in a group, the formula will update (/25 will become /33.3 for 3 members).

I know this is a bit confusing, but don't worry about it too much. As long as you do 20% or more work in a group of 4, you qualify for the entire marks your group qualifies for. In exceptional circumstances, where group of 3 is allowed, that must be a minimum of 27% of work.

Submission timeline -

* Checkpoint 1: Sunday 15th September 23:55
  + Not compulsory, but highly advised if you want early feedback.
  + Submit a document containing UML diagram depicting the classes in your project. Maximum page limit: 2 pages (reasonably-sized diagrams).
  + You will receive feedback on whether it is feasible and if your project idea **CAN** meet the assignment requirements.
* Checkpoint 2: Sunday 6th October 23:55
  + Not compulsory, but highly advised if you want progress feedback.
  + Progress report, including project and responsibility document. Can be and md/ docx/ pdf file.
  + You will receive feedback on whether you have done sufficient groundwork or not, and further advice.
* Final submission: Sunday 27th October 23:55
  + Submit a single zip file as explained above
  + Refer to unit guide regarding late penalty rules
  + Grade based on rubric provided
  + Sample solutions provided
* Live coding: Week 13 sessions in which you can self-enrol

# TOPICS

Name of file submitted in week 12 should be **majorAssignmentTopic[Topic Number].zip** and when unzipped, the folder name should be **majorAssignmentTopic[Topic Number].** Each member of your group will get a 1-mark penalty if either condition is not met.

Choose one of the following 4 topics:

**TOPIC 1: Masters of MQ: Turn-Based RPG Combat**

(File name: **majorAssignmentTopic1.zip**)

A typical turn-based RPG involves two teams, each consisting of one or more characters, fighting until one team has had all their characters reach 0 "health points". On their turn, characters can perform two or more actions, such as attacking or defending. Often, characters cannot take actions if they do not have any health points left.

In most text-based games (like this one), information on the outcome of an action is delivered to the player via an easy-to-read, sometimes exciting string, such as "Aang attacked Deadpool! Deadpool took 5 points of damage!"

A character generally has a few base "stats" such as strength, intelligence and defence that dictate the outcome of an action. For instance, if character A attacks character B, compare A's strength against B's defence. “Dice rolls” or other randomness is often factored into an outcome to give the game some unpredictability and excitement. E.g., a number between 0-5 is added to attack damage, or a dice roll determines if the attack hits or not.

The order in which characters take turns may be team-by-team, back-and-forth, or determined using some sort of “initiative” stat and dice roll.

Characters themselves often have other elements that impact their stats or the potential outcome of an action. This may include equipment (armour, particular weapons, or enchanted rings), attributes based on race (Orc, Elf, Alien, Robot) and class (Rogue, Engineer, Barbarian, Mage), and even other “stat effects” (Poisoned, Wounded).

**TOPIC 2: Music Library Management**

(File name: **majorAssignmentTopic2.zip**)

In the problem, there are songs by artists (artists can be bands or solo artists), and these songs can be singles or belong to albums. You are required to create song catalogues or libraries or playlists with ability to go to the previous song or the next song, as well as shuffle play.

**TOPIC 3: BigMACS (Social network for students)**

(File name: **majorAssignmentTopic3.zip**)

Similar to MACS Discord, we’d like to set up a social network for students. They can subscribe to channels they are interested in, as well as communicate with one or more of the other students through private messaging. While you can take inspiration from the way MACS Discord is setup, you should use your own creativity to add/modify features.

**TOPIC 4: Soccer league management**

(File name: **majorAssignmentTopic4.zip**)

This is for all the fans of *the beautiful game*! In domestic leagues, there are N teams competing for the end-of-the-season championship. While the standard format is that every team plays every other team – TWICE – once at their own ground (home) and once at the other team’s ground (away), you are free to design your own tournament format as long as it’s simple enough (explained within 100 words)

## Samples of UML Class Diagrams using [yuml.me](https://yuml.me/)

[Fraction | +numerator : int; +denominator : int | +Fraction(int, int); +getValue() : double; +isPositive() : boolean]

[Rectangle | +length: int; +breadth: int | +Rectangle(int, int); +Rectangle(int); +area(): int]

[Date | +day : int; +month : int; +year : int | +Date(int, int int); +isInLeapYear() : boolean; +shift(int, int, int) : Date; +shift(int, int) : Date; +shift(int) : Date]

[Time | +hour : int; +minute : int; +second : int | +Time(int, int int); +shift(int, int, int) : Time; +shift(int, int) : Time; +shift(int) : Time]

[DateTime | +date : Date; +time : Time | +DateTime(Date, Time)]

A close-up of a paper

Description automatically generated