The University of Newcastle

SENG1110 - Programming Assignment 2 – Semester 2, 2025

Due Date: By electronic submission (Canvas) by 11:59pm on Fri 30 May 2025

1 Overview

This assignment extends the **Project Management System** developed in Assignment 1. You will enhance the system by incorporating **arrays** for managing collections of projects and tasks, implementing **file input/output** for data persistence, and introducing basic **exception handling** for robustness. The program will continue to allow users to create and manage projects and their associated tasks.

For this assignment, you must only use concepts covered in Weeks 1–12. ArrayLists and other collection frameworks are explicitly prohibited. Submissions violating these constraints will result in marks being deducted.

Carefully read the specification below. Make sure you have all the information necessary to start writing the program. If you are uncertain of something, do not make assumptions. Instead, post your questions to the "Programming Assignment 2" discussion board forum and check it regularly.

Note that you can complete this assignment either individually or in pairs (if you worked as a pair for Assignment 1, you must continue with the same partner. If you worked individually, you must continue individually, unless explicit permission is granted by the lecturer). Your solution must be your own work. You may be required to participate in an oral exam for quality assurance.

2 Program Structure Requirements

You must implement a **menu-driven console application (TIO)**. The core classes from Assignment 1 will be modified to use arrays. The following is a recap of the minimum required attributes for each of the three required classes.

2.1 Task Class

The ‘Task‘ class structure remains largely the same as in Assignment 1.

- **Attributes:**
  - `taskId` (int) – A unique identifier for the task (within its project)
  - `description` (String) – A brief description of the task
  - `completed` (boolean) – Indicates whether the task is completed
  - `taskType` (char) – Represents the task nature: 'A' (Administrative), 'L' (Logistics), 'S' (Support)
  - `taskDuration` (int) – Estimated or actual time spent on the task (in hours)

2.2 Project Class

- **Attributes:**
  - `projectId` (int) – A unique identifier for the project

- projectName (String) – Name of the project
- tasks (Task[]) – An array of Task objects. The size of this array is determined by projectType.
- projectType (String) – Controls the maximum number of tasks allowed:
  - 'Small' – Allows 1 task (array size 1)
  - 'Medium' – Allows up to 2 tasks (array size 2)
  - 'Large' – Allows up to 3 tasks (array size 3)

2.3 UserInterface Class

Implements the **menu-driven system**. All I/O handling and exception handling related to user input must be done exclusively within this class.

- **Attributes:**
  - projects (Project[]) – An array to store Project objects (e.g., up to a maximum of 10 projects, initially all null).
  - scannerInput (Scanner) – To handle input from the user.

3 Functionality Requirements

The program must provide the following functionality, extending Assignment 1:

- **Project and Task Management (Array-based)**
  - **Creating a Project:**
    - The user can create projects up to the **maximum limit defined in** UserInterface **(e.g., 10 projects)**. If the array is full, display an error.
    - Each project must have a **unique** projectId across all existing projects. If the user enters a duplicate ID, display an error message, and assign a new, randomly generated unique ID.
    - The projectType (Small/Medium/Large) is case-insensitive and normalised.
  - **Adding a Task to a Project:**
    - Tasks are added to the ‘tasks‘ array of a specific project (identified by projectId).
    - Tasks require a **unique** taskId *within that project*. If a duplicate task ID is entered for that project, reject with an error message and assign a new, randomly generated unique ID for that task within the project.
    - The taskType is case-insensitive and normalised.
    - Task cannot be added if the project's ‘tasks‘ array is full (based on ‘projectType‘).
  - **Removing a Project:** The user can delete a project by projectId. This involves removing it from the ‘projects‘ array. If the ID is invalid, display an error.
  - **Marking a Task as Completed:** Same as Assignment 1, but operates on tasks within the array structure.
  - **Removing a Task:** The user can delete a task from a project's ‘tasks‘ array. This involves removing it from the array.
- **Project Display**
  - **Displaying All Projects Details:** Iterate through the ‘projects‘ array. Format similar to Assignment 1.

  - **Displaying Completed Tasks for a Project:** Iterate through the specified project's 'tasks' array. Format similar to Assignment 1.
  - **Filtering Tasks by Type:** Iterate through all projects and their 'tasks' arrays. Format similar to Assignment 1.

- **Project Task Duration Summary**
  - Functionality remains the same as Assignment 1, but data is sourced from the array structures.
  - **Handling No Data:** If no projects or tasks exist, display appropriate messages.

- **File Input/Output** (New Functionality)
  - **Load Projects and Tasks from File:**
    - On startup, or via a menu option, the program should be able to load project and task data from a user-specified text file (e.g., 'ProjectData.txt').
    - Assume the file format is correct if the file exists. A suggested format (one line per project, subsequent lines for its tasks, comma-separated):

      ```
      PROJECT_ID,PROJECT_NAME,PROJECT_TYPE
      TASK_ID,TASK_TYPE,TASK_DURATION,COMPLETED_STATUS
      TASK_ID,TASK_TYPE,TASK_DURATION,COMPLETED_STATUS
      ... (more tasks for this project)
      NEXT_PROJECT_ID,PROJECT_NAME,PROJECT_TYPE
      ... (tasks for the next project)
      ```
    - Handle all potential I/O exceptions gracefully.
  - **Save Projects and Tasks to File:**
    - Via a menu option, the program should save all current project and task data to a user-specified text file using the same format as for loading.
    - Handle all potential I/O exceptions gracefully.

- **Exception Handling** (New Requirement)
  - Implement `try-catch` blocks for robust input:
    - When expecting numerical input (IDs, duration), catch relevant exceptions if non-numeric input is given. Re-prompt the user.
  - File operations (load/save) must handle relevant exceptions , displaying user-friendly error messages.

- **Menu System**
  - **Options:** Extend existing menu from Assignment 1 with "Load from File" and "Save to File".
  - **Input Handling:**
    - Normalise string/char inputs as in Assignment 1.
    - Handle duplicate project/task IDs gracefully as described above.
    - Use exception handling for non-integer inputs for IDs/durations, unrecognised types, and menu options, re-prompting the user.
  - The menu re-displays after each action until the user exits.

4 Submission Instructions

Submit a single compressed .zip file named **c9999999_A2.zip** (where c9999999 is your student number) via Canvas.

4.1 Source Code Files

- `Task.java`
- `Project.java`
- `UserInterface.java`

4.2 Report Document (PDF, max 4 pages)

As you design, write and test your solution, report on the following:

- Keep track of how much time you spend designing, coding, testing, and correcting errors.
- Keep a log of what proportion of your errors come from design errors versus coding/implementation errors.
- Screenshots of the program running key functionalities (creating projects/tasks, display options, file load/save operations, and an example of error handling like invalid input).
- **Discussion on Inheritance:** Describe how OOP inheritance *could be accommodated* for the `Task` class and for the `Project` class. Also, discuss the potential benefits/drawbacks.
- (Only if disallowed topics outside Weeks 1-12 were used) Justification for the use of disallowed topics.

4.3 Late Submission and Adverse Circumstances

Note that your mark will be reduced by 10% for each day (or part day) that the assignment is late. This applies equally to week and weekend days. You are entitled to apply for special consideration if adverse circumstances have had an impact on your performance in an assessment item. This includes applying for an extension of time to complete an assessment item. See https://www.newcastle.edu.au/current-students/learning/assessments-and-exams/adverse-circumstances for more details.

4.4 Working as a Pair (two students)

If you worked as a pair for Assignment 1, you must continue with the same partner for Assignment 2. New pairings are not permitted without lecturer approval. Both students receive the same mark. Contributions should be equitable.

4.5 Use of Generative AI/LLMs

According to Student Conduct policies, all submitted work must be your own original work. This means that the use of Generative AI tools like ChatGPT or similar platforms is not permitted for writing or drafting any assessment, including both code and documentation. Any violation of this rule may result in a report to SACO and a zero mark. The only exception is using AI tools for proofreading your submitted PDF report, such as correcting grammar or refining wording.

Appendix I - Marking Criteria (Indicative)

This guide is to provide an overview of Programming Assignment 2 marking criteria. It briefly shows the mark distribution. This guide is subject to adjustment without notice.

Program Correctness (75%)

- **Task Class** (5%): Attributes, methods, design compliance.
- **Project Class** (10%): Array for tasks, methods for task management within array, design compliance.
- **UserInterface Class** (25%): Array for projects, menu system, input validation (including exception handling for format errors), file I/O logic, correct output format.
- **Core Functionality** (35%):
  - Project/Task creation, addition, removal using arrays (ID uniqueness, capacity checks).
  - Display/filtering operations on array data.
  - Project task duration summary.
  - Correct File Load/Save operations.
  - Appropriate Exception Handling(input mismatch, file not found etc.).

PDF Report (15%)

- Content completeness (all requirements addressed, especially the inheritance discussion).
- Clarity, professional formatting, and depth of reflection.

Code Quality & Comments (10%)

- Readability, documentation (comments for complex logic, class/method headers).
- Adherence to constraints (Weeks 1-12 topics, mandatory use of arrays, no ArrayLists).
- User-friendly input/output design.

Deductions

- **Late submission**: -10% per day.
- **Disallowed topics** (e.g., ArrayLists, advanced collections): -20 to -50 marks depending on severity.
- **Not using arrays for project/task collections**: -30 marks.
- **Lack of basic exception handling for I/O or input**: -10 to -15 marks.
- **Incorrect class filenames**: -2 per error.
- **Compilation errors**: -3 per error (up to a cap, may result in 0 for functionality if un-runnable).
- **Runtime errors (unhandled exceptions crashing program)**: -3 per distinct error type/scenario.