School of Computing and Information Technology
University of Wollongong

Session: **Autumn 2025**
Lecturer: **Dr Thanh Le**

CSIT213 Autumn 2025

Assignment 2

Due: Friday 06 June 2025, 11:30 pm
Total marks: 20 marks

Scopes

This assignment is related to UML classes diagrams, Java classes definitions and implementations, polymorphism, collectors, file input/output, and JavaFX.

Please read the assignment specifications below carefully.

1. General Java coding requirements

- Create your programs with good programming style and form using proper blank spaces, indentation, and braces to make your code easy to read and understand.
- Create identifiers with sensible names.
- Add proper comments to describe your code segments where they are necessary for readers to understand what your code intends to achieve.
- Logical structures and statements are properly used for specific purposes.
- In **every source file** you submit for this assignment, you must include the following header:

```
/*-----------------------
Student name:
Student number:
Subject code: CSIT213
------------------------*/
```

2. Submission and marking procedures

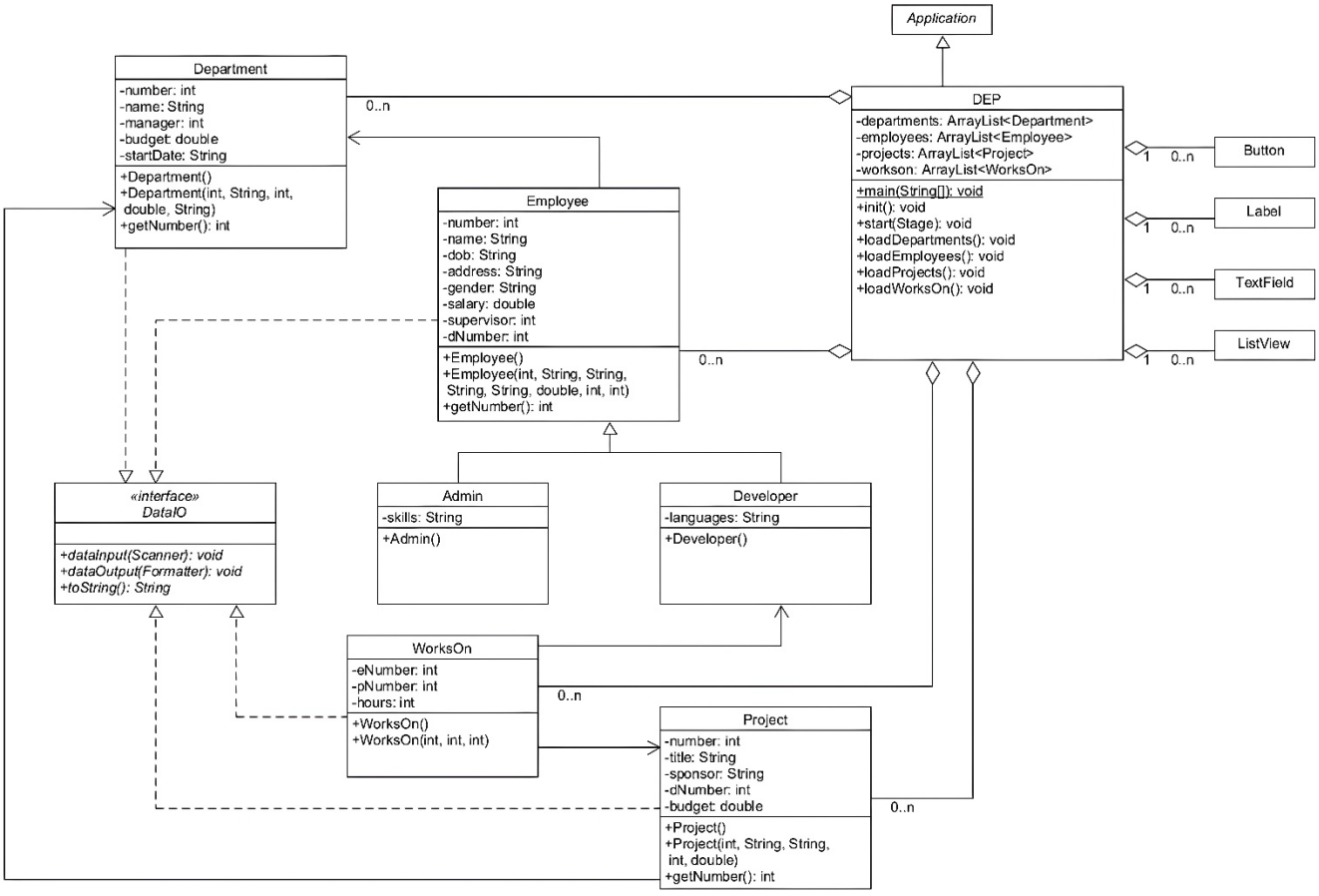
- A submission procedure is explained at the end of this document.
- A submission marked by Moodle as "Late" is treated as a late submission no matter how many seconds it is late. The policy regarding late submissions is detailed in the Subject Outline.
- An implementation that **does NOT compile** due to one or more syntactical or processing errors scores **NO marks**.
- All tasks must be solved individually without any cooperation from the other students. If you have any concerns, please consult your lecturer or tutor during lab classes or consultation hours. Plagiarism will result in a **failure** grade being recorded for the assessment task.
- The use of GenAI (e.g., ChatGPT or Microsoft Co-pilot) is **NOT permitted** for the assessment tasks.
- It is recommended to solve the problems before attending the laboratory classes to efficiently use supervised lab time.

Assignment Task (20 marks)

In this assignment, you are required to design and implement a Department, Employee, and Project (DEP) System in Java. This system will help a company efficiently manage its departments, employees, projects, and developers working on those projects.

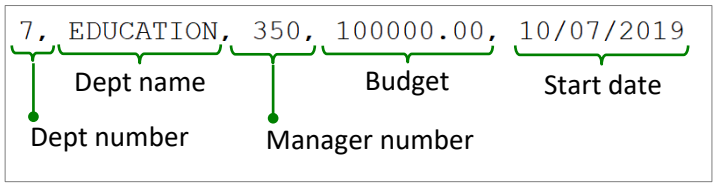
1. Specifications

- The UML class diagram for the DEP system is shown below. You may add new classes, methods and attributes to the UML class diagram; however, you must **NOT** modify or remove any existing classes, attributes, or methods. Your Java implementation must be consistent with the UML class diagram.

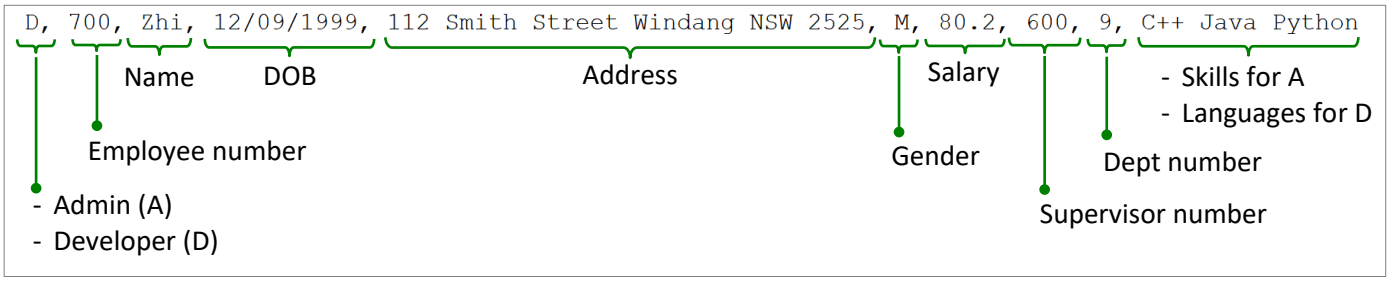


- The DEP program initially loads the data for departments, employees, projects, and works-on from **four text files**: *departments.txt*, *employees.txt*, *projects.txt*, and *workson.txt*, respectively. The data is then stored in the containers by calling the methods *loadDepartments(), loadEmployees(), loadProjects()*, and *loadWorksOn()*. The application then displays the GUI and handles the user events.

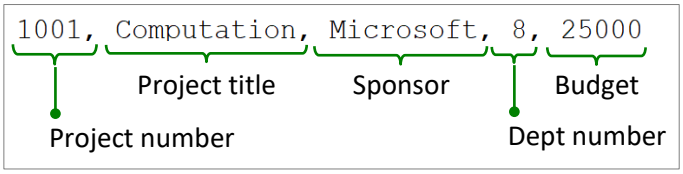
- Each field in the text file is separated by a comma "," and a space " ". The format of the file *departments.txt* is as follows:



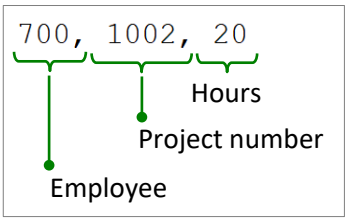
- The format of the file *employees.txt* is as follows:



- The format of the file *projects.txt* is as follows:



- The format of the file *workson.txt* is as follows:



Hint: You may open a text file, and use the method `useDelimiter(", |\r\n|\n")` of a `Scanner` object before reading input data from the text file.

2. Functionalities

The DEP class must include JavaFX components (e.g., Label, TextField, Button, and ListView) in the GUI to enable user interaction for the following functionalities:

2.1. Displaying data

- When the application starts, it displays the lists of 1) department numbers, 2) employee numbers, 3) project numbers, 4) works-on information, and 5) a pie chart of department budgets. A sample GUI is shown in Fig. 1.
- When a user clicks an item in a list, its details appear in the text fields below (Fig. 2).

2.2. Adding a new works-on record

Below is the user interaction for adding a new works-on information:

- The user first selects an employee number and a project number from the lists.
- When the user clicks the '*Add*' button, the application validates the selected employee and project:
  - If the selected employee is **NOT** a developer, display a notification message '*Please select a developer'* in the text field (Fig. 3).
  - Otherwise, check if the developer has been working on the project. If yes, display a notification message '*Employee [ID] has already worked on Project [ID]'* in the text field (Fig. 4). If no, a dialog pops up to input the working hours. A new works-on record is then added to the '*workson'* container and the '*workson'* list-view in the GUI (Fig. 5).

2.3. Deleting a works-on record

When the user clicks the '*Delete*' button, the application pops up a confirmation dialog and deletes the selected works-on record if confirmed (Fig. 6). A notification message '*The selected works-on record has been deleted'* is displayed in the text field.

2.4. Saving all works-on records to a file

When the user clicks the '*Save'* button, the application saves the works-on information from the '*workson'* container to a text file, named '*workson.txt*'. A notification message indicating how many records have been saved will be displayed in the text field (Fig. 7).

Note: The application must be able to handle exceptions during user interaction. These exceptions may include invalid information when loading departments, employees, projects, and works-on.

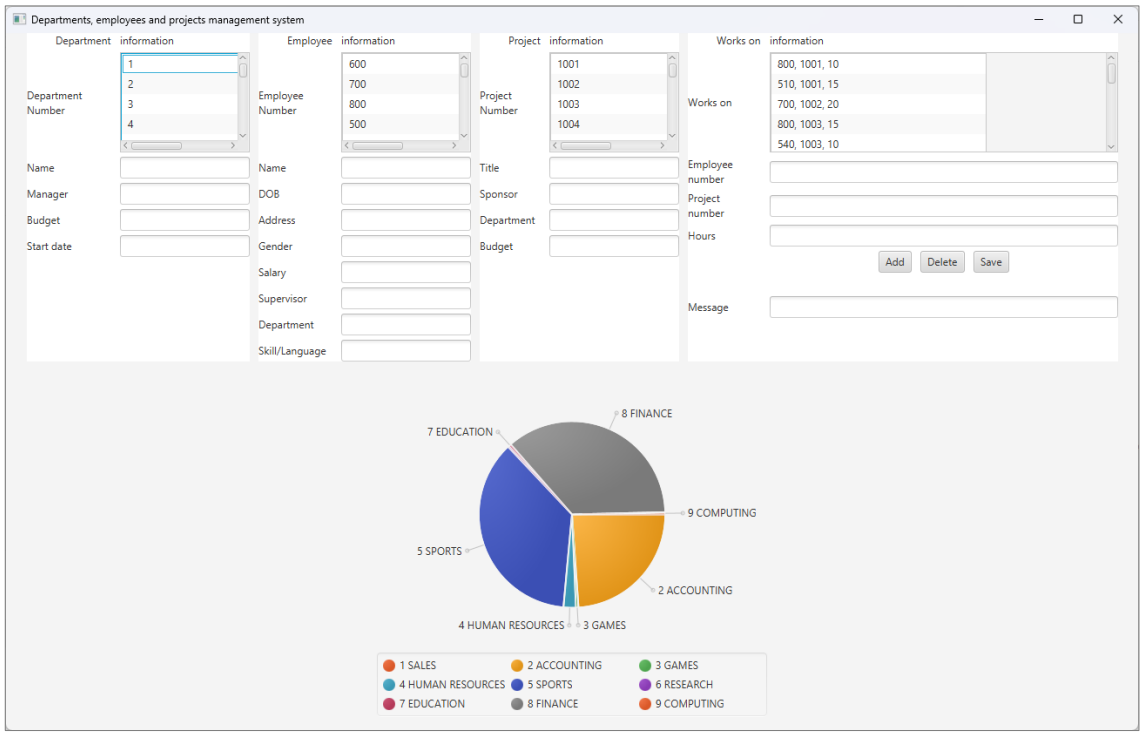


Figure 1: Initial GUI displaying the data from the text files.

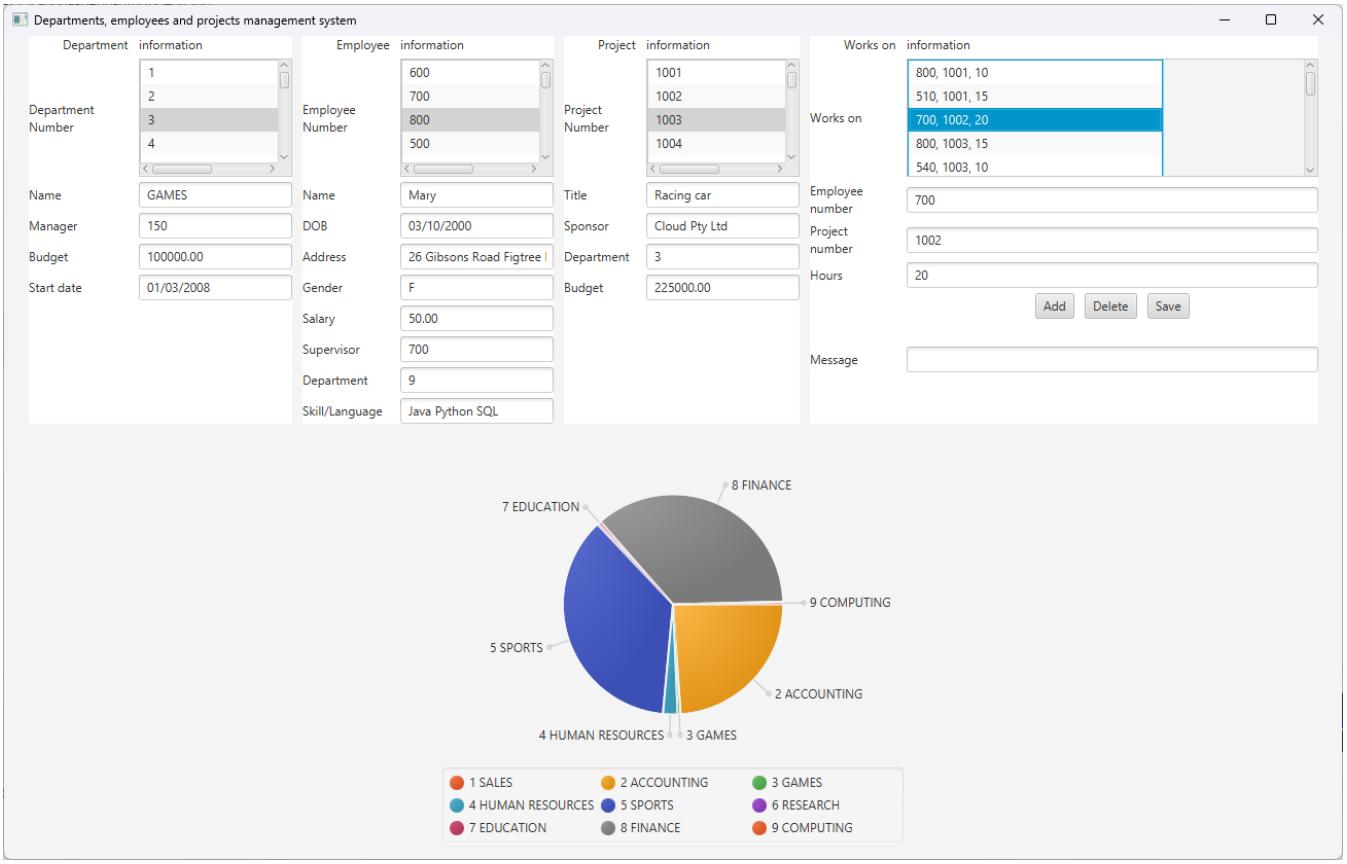


Figure 2: Details of the item displayed in the text fields when selected.

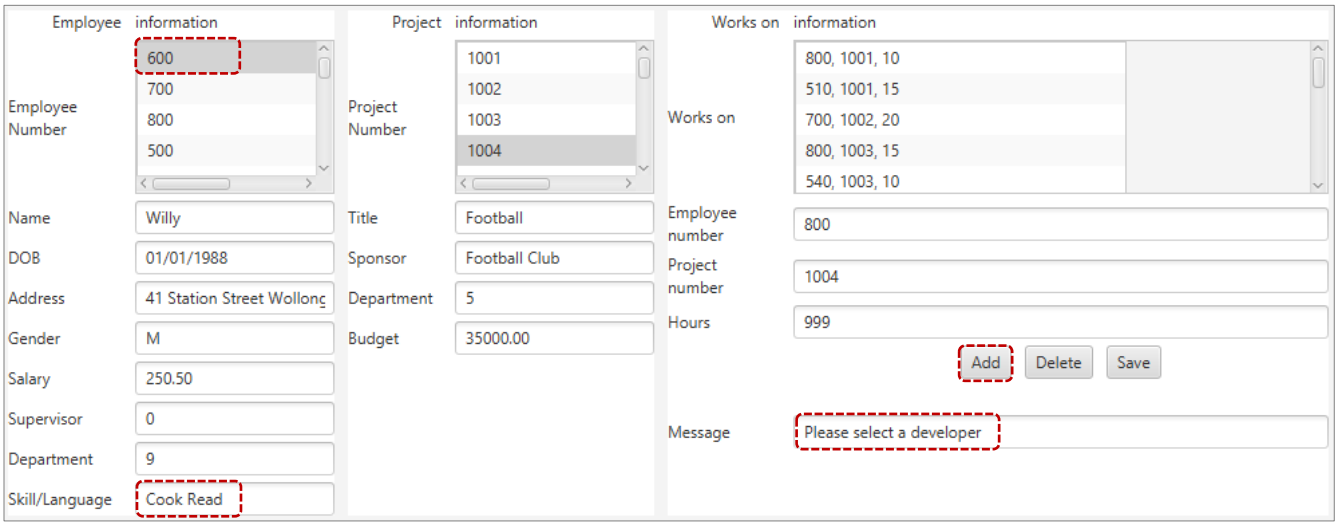


Figure 3: Notification displayed when a non-developer employee is selected for adding a new work-on.

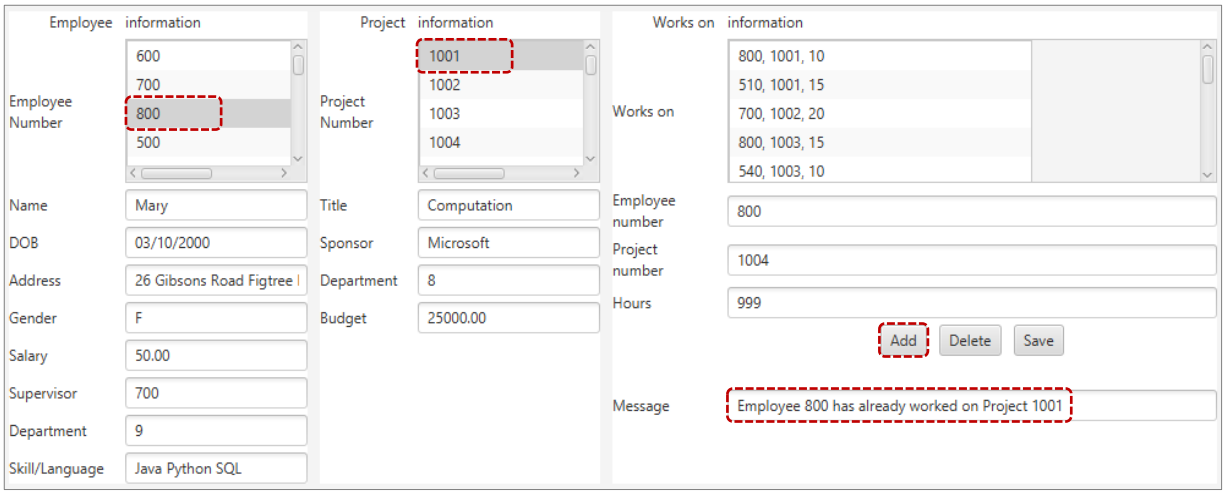


Figure 4: Notification displayed when an existing works-on record is found.

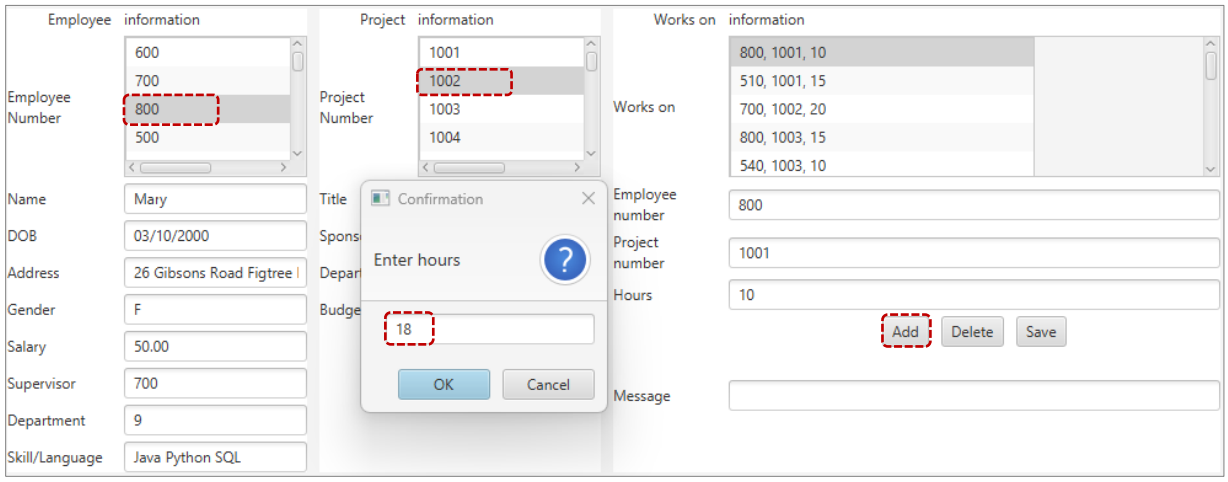


Figure 5: A pop-up dialog displayed when adding a valid works-on record.

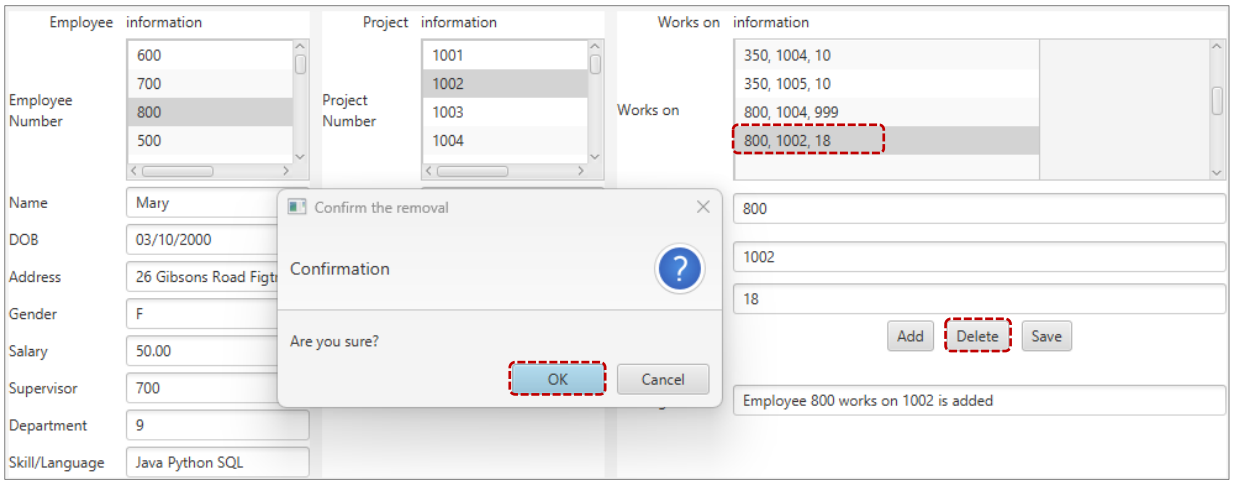


Figure 6: A confirmation pop-up dialog displayed when deleting the selected works-on record.

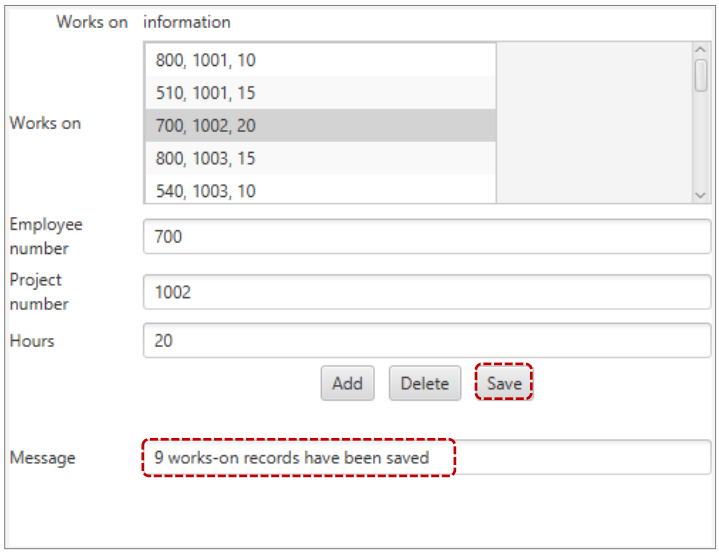


Figure 7: Saving the works-on records to the text file.

3. Deliverables

- **Six Java files** (or more if needed): *DEP.java, DataIO.java, Department.java, Employee.java, Project.java*, and *WorksOn.java*. **(16 marks)**

  Your program must:

  - be consistent with the UML class diagram;
  - follow the conventions for naming all classes, variables, and methods;
  - provide sufficient comments;
  - use proper blank spaces, indentation, and braces to make your code easy to read and understand; and
  - follow the specified implementation steps.

- **A PDF file** named *Assignment_2.pdf* containing: (1) the UML class diagram, and (2) the compilation and testing results. **(4 marks)**

  - Remember to use the CSIT213 palette in the UMLet tool.
  - Use the option *File->Export* as to export the UML class diagram into a file in BMP format.
  - Insert the BMP file into the document.

- Create **a ZIP file** named *Assignment_2.zip* containing all the Java files and the PDF file for submission.

4. Submission

Submit your compressed file through Moodle:

- Access the Moodle at *http://moodle.uowplatform.edu.au*.
- Log in and select the site *"CSIT213 (S125) Java Programming"*.
- Scroll down to the section "*Assignments and Submissions"*.

- Click on the link “*Assignment* 2 - *Submission”*.
- Click on the button “*Add Submission”*.
- Upload the compressed file *Assignment_2.zip* into the box.
- Click on the button *“Save changes”*.

***** END*****