LAB211 Assignment

Type:	**Long Assignment**
Code:	**J1.L.P0035**
LOC:	**500**
Slot(s):	**N/A**

Title

Software Developer Management

Background

A software company (**DevCorp**) requires a system to manage the information of its developers. Each developer has personal and professional information, including their main programming language and salary.

Initial data is stored in two files: **developers.txt** (*basic information*) and **projects.txt** (*project assignments and duration*).

File: developers.txt	Description
DEV001, Nguyen Van A, [Java, C++], 5000	Information on a line: <Dev ID>, <Full Name>, <Programming Languages>, <Salary_USD> Note: Salary is monthly in USD.
DEV002, Tran Thi B, [Python, Scala], 4500	
DEV003, Le Van C, [C#, Golang, React], 5500	

File: projects.txt	Description
PROJ01, DEV001, e-Commerce, 12, 01/01/2026	Information on a line: <Project ID>, <Dev ID>, <Project Name>, <Duration_Months>, <Start_Date> Note: Dev ID must exist in the developers list.
PROJ02, DEV001, CRM System, 6, 02/01/2026	
PROJ03, DEV002, Data Analysis, 8, 01/02/2026	

Constraints

1. **Constraints on Developers:**
   - Developer ID must be a 6-character string, starting with "DEV" followed by 3 digits (e.g., DEV001), and must be **unique**.
   - Full Name cannot be empty and must contain at least two words.
   - Programming Languages cannot be empty (e.g., Java, Python, C#).
   - Salary must be a positive integer, at a minimum of 1000 USD.
2. **Constraints on Projects:**
   - Project ID must be **unique**.
   - Dev ID must exist in the Developer list.
   - Project Name cannot be empty.
   - Duration must be a positive integer in months, minimum 1 month.
   - Start date must be a date in future.

Program Specifications

Students are required to build a Java console application using **the Object-Oriented Programming (OOP)** approach to manage developers and projects in the company.

Menu Functions

1. List all Developers
2. Add a new Developer
3. Search for a Developer by ID
4. Update a Developer's salary by ID
5. List all Developers by Language
6. Add a new Project
7. List all Projects by Developer (Grouped)
8. Calculate Total Experience by Dev ID
9. Remove a Developer by ID
10. Sort Developers by Salary
11. Save data to files
12. Quit program

Features and Assessment

No.	LOCS	Function Description
1	**20**	**List all Developers:** Display data in a **formatted table** with column headers (ID, Name, Language, Salary).
2	**50**	**Add a new Developer:** Accept ID, Name, Language, and Salary. Add to the list only if all constraints are satisfied. Display success/failure message.
3	**20**	**Search for a Developer by ID:** If the Developer does not exist, show: **"Developer ID does not exist!"**. Otherwise, display Developer information.
4	**50**	**Update a Developer's salary by ID:** If the Developer does not exist, show: **"Developer ID does not exist!"**. Otherwise, allow updating **Salary** (must be >= 1000$).
5	**50**	**List all Developers by Language:** Accept a language input (e.g., "Java"); display all developers who list that language in their skills.
6	**50**	**Add a new Project:** Accept Project ID, Dev ID, Project Name, Duration and Start Date. **Dev ID must be selected from a menu of existing developers**. Validate all constraints.
7	**50**	**List all Projects by Developer (Grouped):** Display the list of Developers. Under each developer, list all Projects they are currently or have been assigned to, showing Project ID, Duration, and Start Date.
8	**50**	**Calculate Total Experience by Dev ID:** Accept Dev ID; calculate and display the total duration (in months) the developer has spent on all registered projects.
9	**30**	**Remove a Developer by ID:** Accept Dev ID. If the Dev does not exist, show: **"Developer ID does not exist!"**. If the Dev has associated Projects, show: **"Cannot delete: Developer is assigned to projects."**. Otherwise, remove the record.
10	**30**	**Sort Developers by Salary:** Display the list of Developers **sorted by Salary in ascending order**.
11	**50**	**Save data to files:** Save both Developer and Project lists into **developers.txt** and **projects.txt** files, respectively. Show a confirmation message.

No.	LOCS	Function Description
12	**50**	**Quit program:** Check for unsaved changes. If changes exist, prompt the user: **"Do you want to save the changes before exiting? (Y/N)"**. Exit the program after handling the save/discard choice and **reload data on program start.**
	500	

Note for Students

- **OOP Principles:** Follow OOP principles strictly (***encapsulation, inheritance, polymorphism, abstraction***). Design appropriate classes (*Developer, Project, etc.*).
- **Computational Thinking:** Apply Decomposition, Pattern Recognition, Abstraction, and Algorithm in your design.
- **Input Validation:** Input validation must strictly follow all defined constraints. Show appropriate, helpful messages for invalid or missing data.
- **Data Persistence:** The program must **load** data from the files upon startup and **save** data upon existing (*if changes occurred*) or when function 11 is selected.

Submission

- Source code
- Class Diagram
- Flow-chart