JAVA PROGRAMMING

Tutorial 08

Activity 01: Person

In this exercise, you will design and implement class **Person** which contains the information about id, name, date of birth (format: ddmmyyyy), email, phone number, and address. The following table is a table of the domain constraints that apply to the attributes of the class.

Class	Attribute	type	mutable	optional	length	min	max
Person	id	Integer					
	name	String					
	dateOfBirth	String					
	email	String					
	tel	String					
	address	String					

Complete the following tasks:

- Complete the domain constraints in the table, using your practical understanding of the application.
- Create a Java package called "**tut08.person**" which contains all implemented classes in this exercise.
- Design and implement the **Person** class with robust validation, using ***Regex*** for validation where applicable.
- Develop helper classes that implement the **Comparator** interface and are able to sort instances of the **Person** class based on name, age, or both in ***ascending*** order.
- Create a list of **Person** instances and demonstrate the sorting functionality.
- Write a test program to validate the correctness of the implemented functionality.

Activity 02: Student

In this exercise, your task is to design and implement a Java class named **Student**. The **Student** class inherits from the **Person** class and includes additional attributes such as studentID, enrolledCourses (a list of **Course** objects), and gpa (grade point average, max = 4.0). Complete the following tasks:

- Define a comprehensive table of domain constraints applicable to the attributes of the **Student** class.
- Design and implement the **Student** class within the "**tut08.person**" package.
- Develop a helper class that implements the **Comparator** interface and is able to sort instances of the **Student** class based on GPA in ***ascending*** order.
- Create a list of **Student** instances and demonstrate the sorting functionality.
- Write a test program to validate the correctness of the implemented functionality.

Activity 03: Course

In this exercise, your objective is to design and implement a Java class named **Course**. The **Course** class will contain attributes such as courseID, name, credits, department (use an ***Enum***), and semester (use an ***Enum***). Additionally, you'll implement an interface named **StudentManageable** for managing enrolled students on a course. Complete the following tasks:

- Define a comprehensive table of domain constraints applicable to the attributes of the **Course** class.
- Design and implement the **Course** class within the "**tut08.course**" package.
- Develop helper classes that implement the **Comparator** interface and are able to sort instances of the **Course** class based on name, credit, department, and semester in ***ascending*** order.
- Implement the **StudentManageable** interface with methods such as ***addStudent***, ***removeStudent***, ***updateStudent***, ***getEnrolledStudent***, etc.
- Create a list of **Course** instances and demonstrate the sorting functionality.
- Write a test program to validate the correctness of the implemented functionality.

Activity 04: Learning Management System

In this exercise, your mission is to design and implement a Java class named LearningManagementSystem. This program simulates a simplified **Learning Management System (LMS)** using Java. It utilizes a **Map** to store a list of students and a list of courses. Complete the following tasks:

- Design and implement the LearningManagementSystem class within the "**tut08.lms**" package. Ensure the class has at least two attributes: a **HashMap<Integer, Student>** to store students and a **HashMap<Integer, Course>** to store courses.
- Implement methods to dynamically add students or courses to the LMS. Ensure that the system generates unique IDs automatically using a specified format, such as "**SP2024**" followed by an **auto- increased ID**, where "**SP2024**" represents the current season (**Sp**ring, **Su**mmer, **Au**tumn, **Wi**nter) and year.
- Develop functionalities to remove specific students or courses based on their respective IDs.
- Implement search methods allowing users to find students or courses based on specified criteria. These methods should return the ID.
- Create methods to return ascending sorted lists of students or courses based on certain criteria.
- Write a test program to validate the correctness of the implemented functionality.

Activity 05: Word Occurrence

In this exercise, students will work with a text file named "**input.txt**", which contains paragraphs of text, potentially sourced from a story, book, or any other text-based content. The goal of this exercise is to develop a program that reads the file, analyzes its contents, and counts the occurrences of each individual word. Here, a word is defined as a sequence of characters without *whitespace*, *tabs*, *line breaks*, or *special characters* like commas, dashes, or periods, etc. To achieve this, complete the following tasks:

- Create a file named "**input.txt**" which contains paragraphs of text.
- Design and implement the **WordOccurrence** class within the "**tut08.word_occurrence**" package. Ensure the class uses a **Map** data structure for efficient storage of word counts. The result should be sorted in ascending order of the words.

- After processing, the program should generate an output file named "**output.txt**" with the following format:
    - The first line should indicate the ***total number of words***.
    - Subsequent lines should list each ***word*** and ***its occurrence count***.
- Create some more input files to validate the correctness of the implemented program. Here are some test cases for the text file: File does not exist, Empty file, File containing only one word, File containing two words, File containing words with punctuation (e.g., comma, period, hyphens, etc.), File containing multiple occurrences of the same word, File containing words with different capitalizations (e.g., "Word" vs. "word"), File containing words with special characters, File containing words with leading/trailing whitespace, Large file, etc.

Submission

Submit a zip file containing all Java programs to this tutorial's submission box in the course website on FIT Portal.