LABORATORY 05-07

OBJECTIVES

- Learn how to program using object-oriented programming concepts
- Learn how to leverage layered architecture
- Learn how to implement your own exception classes
- Learn how to propagate exceptions through program layers
- Learn to specify and test your code using **pydoc** and **PyUnit**.

REQUIREMENTS

- You will be given one of the problems below to solve.
- Use simple feature-driven software development process.
- Use object oriented programming for the program from the first iteration. Implement classes that represent the entities in the problem domain, as well as the application's layers, including the user interface.
- The program must provide a console-based user interface based on a menu system. Exact implementation details are up to you.
- Iterations are scheduled for three successive labs:
    - **Iteration 1 (deadline is week 6, 25% of grade):**
        - Implement features 1 and 2.
        - Have at least 10 items in your application at startup.
        - Provide specification and tests for all classes and methods except those of the UI.
    - **Iteration 2 (deadline is week 9, 25% of grade):**
        - Also implement features 3 and 4.
        - Provide specification for all classes and methods except those of the UI.
        - Unit tests must be implemented using **PyUnit**
    - **Iteration 3 (deadline is week 10, 50% of grade):**
        - All required features must be implemented
        - Have at least **100** items in your application at startup.
        - This iteration will be tested by your lab professor ☺
- Data validation - when the user enters invalid input values, they will be notified about the mistake.

BONUS POSSIBILITY (0.1P)

- Have 80% code coverage using PyUnit tests, for all modules except the UI. Install the coverage module, examine and improve your test code coverage until you reach this threshold. Deadline is **week 9**.

BONUS POSSIBILITY (0.2P)

- In addition to the menu-based user interface required, also implement a graphical user interface (GUI) for the program.
- To receive the bonus, both user interfaces (menu-based and graphical) must use the same program layers. You have to be able to start the application with either user interface.
- The deadline for this bonus is **week 11**.

PROBLEM STATEMENTS

1. STUDENTS REGISTER MANAGEMENT

A faculty stores information about:

- Student: *<studentID>, <name>.*
- Discipline: *<disciplineID>, <name>*.
- Grade: *<disciplineID>, <studentID>, <grade_value>*.

Create an application which allows to:

1. Manage the list of students and available disciplines. The application must allow the user to add, remove, update, and list both students and disciplines.
2. Grade students at a given discipline. The program must allow only those students who are enrolled at a given discipline to receive any number of grades. Deleting a student also removes their list of grades. Deleting a discipline deletes all the grades at that discipline.
3. Search for disciplines/students based on their ID or name/title. The search must work using case-insensitive, partial string matching, and must return all matching disciplines/students.
4. Create statistics:
    - All students enrolled at a given discipline, sorted alphabetically or by descending order of average grade.
    - All students failing at one or more disciplines (students having an average <5 for a discipline are considered to be failing)
    - Students with the best school situation, sorted in descending order of their aggregated average (the average between their average grades per discipline).
    - All disciplines at which there is at least one grade, sorted in descending order of the average grade received by all students enrolled at that discipline.
5. Unlimited undo/redo functionality. Each step will undo/redo the previous operation performed by the user. Undo/redo operations must cascade and have a memory-efficient implementation (no superfluous list copying).

2. STUDENT LAB ASSIGNMENTS

Write an application that manages lab assignments for students at a given discipline. The application will store:

- Student: *<studentID>, <name>, <group>.*
- Assignment: *<assignmentID>, <description>, <deadline>, <grade>.*
- Grade: *<assignmentID>, <studentID>, <grade>*.

Create an application that allows to:

1. Manage the list of students and available assignments. The application must allow the user to add, remove, update, and list both students and assignments.

2. Give assignments to a student or a group of students (unless already given). In case an assignment is given to a group of students, every student in the group will receive it. In case there exist students who were previously given that assignment, it will not be assigned again.
3. Grade student for a given assignment. When grading, the program must allow the user to select the assignment that is graded, from the student's list of ungraded assignments. A student's grade for a given assignment cannot be changed. Deleting a student also removes their assignments. Deleting an assignment also removes all grades at that assignment.
4. Create statistics:
    - All students who received a given assignment, ordered alphabetically or by average grade for that assignment.
    - All students who are late in handing in at least one assignment. These are all the students who have an ungraded assignment for which the deadline has passed.
    - Students with the best school situation, sorted in descending order of the average grade received for all assignments.
    - All assignments for which there is at least one grade, sorted in descending order of the average grade received by all students who received that assignment.
5. Unlimited undo/redo functionality. Each step will undo/redo the previous operation performed by the user. Undo/redo operations must cascade and have a memory-efficient implementation (no superfluous list copying).

3. MOVIE RENTAL

Write an application for movie rental. The application will store:

- Movie: *<movieId>, <title>, <description>, <genre>*.
- Client: *<clientId>, <name>*.
- Rental: *<rentalID>, <movieId>, <clientId>, <rented date>, <due date>, <returned date>.*

Create an application which allows the user to:

1. Manage the list of clients and available movies. The application must allow the user to add, remove, update, and list both clients and movies.
2. Rent or return a movie. A client can rent an available movie until a given date, as long as they have no rented movies that passed their due date for return. A client can return a rented movie at any time. Only available movies are available for renting.
3. Search for clients or movies using any one of their fields (e.g. movies can be searched for using id, title, description or genre). The search must work using case-insensitive, partial string matching, and must return all matching items.
4. Create statistics:
    - Most rented movies. This will provide the list of movies, sorted in descending order of the number of times they were rented or the number of days they were rented.
    - Most active clients. This will provide the list of clients, sorted in descending order of the number of movie rental days they have (e.g. having 2 rented movies for 3 days each counts as 2 x 3 = 6 days).
    - All rentals. All movies currently rented.

- Late rentals. All the movies that are currently rented, for which the due date for return has passed, sorted in descending order of the number of days of delay.

5. Unlimited undo/redo functionality. Each step will undo/redo the previous operation performed by the user. Undo/redo operations must cascade and have a memory-efficient implementation (no superfluous list copying).

4. LIBRARY

Write an application for a book library. The application will store:

- Book: *<bookId>, <title>, <description>, <author>*.
- Client: *<clientId>, <name>.*
- Rental: *<rentalID>, <bookId>, <clientId>, <rented date>, <due date>, <returned date>.*

Create an application which allows the user to:

1. Manage the list of clients and available books. The application must allow the user to add, remove, update, and list both clients and books.
2. Rent or return a book. A client can rent an available book until a given date. A client can return a rented book at any time. Only available books can be rented.
3. Search for clients or books using any one of their fields (e.g. books can be searched for using id, title, description or author). The search must work using case-insensitive, partial string matching, and must return all matching items.
4. Create statistics:
   - Most rented books. This will provide the list of books, sorted in descending order of the number of times they were rented or the number of days they were rented.
   - Most active clients. This will provide the list of clients, sorted in descending order of the number of book rental days they have (e.g. having 2 rented books for 3 days each counts as 2 x 3 = 6 days).
   - Most rented author. This provides the list of book authored, sorted in descending order of the total number of rentals their books have.
   - Late rentals. All the books that are currently rented, for which the due date for return has passed, sorted in descending order of the number of days of delay.
5. Unlimited undo/redo functionality. Each step will undo/redo the previous operation performed by the user. Undo/redo operations must cascade and have a memory-efficient implementation (no superfluous list copying).

5. NAME AND ADDRESS BOOK MANAGEMENT

The following information may be stored in a name and address book:

- Person: *<personID>, <name>, <phone number>, <address>*
- Activity: *<activityID>, <personIDs>, <date>, <time>, <description>*

Create an application which allows the user to:

1. Manage the list of persons and activities. The application must allow the user to add, remove, update, and list both persons and activities.
2. Add/remove activities. Each activity can be performed together with one or several other persons, who are already in the user's address book. Activities must not overlap (not have the same starting date/time).
3. Search for persons or activities. Persons can be searched for using name or phone number. Activities can be searched for using date/time or description. The search must work using case-insensitive, partial string matching, and must return all matching items.
4. Create statistics:
    - Activities for a given day/week. List the activities for a given day, in the order of their start time, or their date/time.
    - Busiest days. This will provide the list of upcoming days with activities, sorted in descending order of the number of activities in each day.
    - Activities with a given person. List all upcoming activities to which a given person will participate.
    - List all persons in the address book, sorted in descending order of the number of upcoming activities to which they will participate.
5. Unlimited undo/redo functionality. Each step will undo/redo the previous operation performed by the user. Undo/redo operations must cascade and have a memory-efficient implementation (no superfluous list copying).