ASSIGNMENT 06-08

REQUIREMENTS

- You will be given one of the problems below to solve using the simple feature-driven development process.
- Use object oriented programming from the start. 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. Implementation details are up to you.

For week 8 (Iteration 1, 25% of grade)

- Implement features 1 and 2.
- Have at least 10 items in your application at startup, **generated procedurally**.
- Provide specification and tests for all classes and methods involved in the first functionality.
- Implement and use your **own exception classes**.

For week 9 (Iteration 2, 25% of grade)

- Implement features 3 and 4.
- Implement tests as **PyUnit unit tests**.

For week 10 (All required features must be implemented, 50% of grade)

- Implement the **undo/redo** feature ☺

Bonus possibility (0.1p, deadline week 10)

- Have 95% 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.

Bonus possibility (0.2p, deadline week 10)

- In addition to the required menu-based user interface, 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 UI, without making changes to code.

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. Any student may receive one or several grades at any of the disciplines. Deleting a student also removes their grades. Deleting a discipline deletes all grades at that discipline for all students.
3. Search for disciplines/students based on 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 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 ASSIGNMENT

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**: *<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. In case an assignment is given to a group of students, every student in the group will receive it. In case there are 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 removes their assignments. Deleting an assignment also removes all grades at that assignment.
4. Create statistics:
   - All students who received a given assignment, ordered 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.

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 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 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).
   - 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>*, *<author>*.
- **Client**: *<clientId>*, *<name>*.
- **Rental**: *<rentalID>*, *<bookId>*, *<clientId>*, *<rented date>*, *<returned date>*.

Create an application which allows the user to:

1. Manage the list of clients and 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. 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 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.
- 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 number of rentals their books have.

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. ACTIVITY PLANNER

The following information is stored in a personal activity planner:

- **Person**: <*personID*>, <*name*>, <*phone number*>
- **Activity**: <*activityID*>, <*personID*> - list, <*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 planner. Activities must not overlap (not have the same starting date/time) for a person.
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 date. List the activities for a given date, in the order of their start time.
   - Busiest days. This will provide the list of upcoming dates with activities, sorted in descending order of the number of activities in each date.
   - Activities with a given person. List all upcoming activities to which a given person 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).