OOP Project Assignment

General structure

For this semester you will have to do a more comprehensive OOP project. In order to get a full grade, you will have to use the following methodologies and OOP constructs inside your project where possible:

- **Inheritance** for your objects.
- **Constructors** will be used to instantiate any object you might use. Please do not fill the fields of your objects manually. Use **Destructors** to free the memory automatically.
- Utilizing the "**protected, private, public"** keywords (as we have done in the laboratory).
- **Overloading** and/or **Overriding** for methods.
- Use **Getters** and **Setters** when working with object attributes.
- Your project must consist of at least 3 classes.
- Utilize **abstract** classes wherever possible, establish different interactions between objects through **interfaces** (Optional, but extremely useful for extensibility of your project)

Some project statements might be abstract. This is intended since it represents an opportunity for you to translate a real problem into a set of objects, and identify how those objects interact. If you are undecided on what functionalities should you project support, we can gladly offer some specific features that each project can contain. The projects will be marked based on the following 3 criteria:

➔ Respecting the general structure
➔ Originality and project complexity
➔ Project maintainability, extensibility, overall design, and the ability to easily accommodate possible new features. (Tip: check Design Patterns)

Obs.: In less than 2 years, you will likely attend your first tech interview, where your programming skills will be evaluated. Since AI cannot replace you in the interview process, treat this homework as an opportunity to learn, grow and showcase your skills, by preparing a remarkable project for your portfolio. As an engineering student, it's crucial to establish a solid foundation for future career. The usage of AI will not be tolerated, as it sabotages your long-term growth.

Project List

These are some of the projects that will be assigned to you automatically if you do not choose to do a project equal in difficulty to the ones below:

1. A video game inventory management system

In this project you will be creating an inventory system, together with the ability to consume, add, sort and discard items. Your items can have multiple types (**consumables, unique, stackable**, etc.). Your items must also **have a price** in your chosen currency.

After that, you will implement a simple **menu** structure, where a user can choose to **add, consume, order the inventory, search an item based on id or name, discard items, sell and buy** (with the currency stored in the inventory) different items. For the sell and buy mechanism, I suggest you use a second inventory as a **shop.**

Tips: You can use inheritance for the shop and inventory, after all they are both storage spaces. Items can also benefit from using inheritance, since they have a few different attributes but most of them are common.

2. An airline reservations system

In this project you will be creating an airline reservation system. For this, you will have in mind multiple objects: the airplane you are trying to fill and customers (people that can either be randomly generated with their attributes like age, weight or manually inserted). You should have 2 different classes of passengers, Business and Economy (so your seats could cost more or less).

After implementing the above Object logic, you should create a simple menu where you can **create a new customer (either manually or automatically), choose and buy a certain seat** (Bonus: the seats should have a price and if the customer does not have enough money, you could suggest some lower priced seats automatically)**, assert if the plane is filled and reject other customers. Your menu should also contain a way to search customers by id or name, SWAP seats between 2 customers or remove the booking of one customer.**

3. A video game character system

In this project you will create a system which can manage characters of a video game. The attributes you want these characters to have been at your choice (some examples can be HP, Mana, Stamina, maybe a list of buffs). You should have multiple types of characters, that inherit from a base class. The derived classes might be something like an NPC character, a player character, an enemy, etc. All these derived classes should have attributes and methods specific to their needs.

After creating the aforementioned objects, you will implement a simple console menu that allows you to **create characters, delete characters, give them a new buff or modify a specific characters HP or Mana by commands.**

4. A fighting game between 2 characters

In this project you will create 2 game characters, derived from a base class with basic attributes and methods. These 2 characters will then each have their own methods (for possible spells and attacks) and attributes (ex: one might have a special attribute for necromancy while another might be a rogue which holds a number of arrows).

The main purpose of this project will be to have these 2 characters fight. **You should have a console menu for creating these 2 characters (inputting their initial stats) and then they will fight each other. The logic of the fight can be however you want it to be (**one example might be a turn-based approach where each character chooses a random attack from their list of attacks and uses it against the opponent).

Be creative: you can add healing methods, methods (abilities) that allow a character to attack twice in a row, methods that make the enemy take damage over time like bleeding or poison, etc. (remember, the methods of a class define the behavior of your character!).

5. **“Război”– the card game**

In this project you must create a simple card game, called “Război”. The rules for this game are simple and you can check more about them online:

- The number of players is 2 (so you will have a player class for these 2 contenders)
- The number of cards is split equally between the 2 players.
- The cards have values starting from 2 and ending with 14. A player can have multiple cards with the same value.

Game logic:

The game is simple, starting with player 1, each player puts down one card from the top of their deck (they do not see the cards, the players can **NOT** choose which card they place, it is simply the one on the top of the deck). The bigger card wins and the player who won **add both cards played to the bottom of their deck** and starts the next round. The game ends when one player has all the cards. If both players put equal cards on the table, a “war” starts – during this phase, both players must put down an equal number of cards with the value of the card on the ground.

The player who happens to have placed down the last card with the bigger value, wins takes all the cards placed down so far! (ex: if both players put on the table the card 4 -> they must both put down 4 cards, the one who has the biggest number of the 4th card, wins and takes all the cards placed down so far!).

If, the last card of this “war” was the same for both players, another “war” starts and the players must again place down a number of cards equal to the value of the draw. If at any point a player runs out of cards, they lose.

For this project, you must have a **card object** (with a simple value), a **deck object** (that can hold cards inside, using an array or a list) and 2 **player objects**. The deck of cards can be a real one, or you can generate a random amount of cards with random values (your choice). After that, you must implement a simple menu to start the game and a way to visualize (print some output) each “battle”. This visualization can be however you want, but you should make visible which player wins every round and how many cards each player still has in their deck.

6. **A parking lot management system for malls**

In this project, you will create a parking lot management system. When designing your program, you should keep in mind to account for the **capacity** of the parking lot, **vehicle types** that can be parked in that specific lot (ideally it should allow parking different parking sizes), **payment system** which will account for different types of vehicles and also it should accommodate having different rates for each hour (ex: first hour can be free, the next 2 hours priced at a specific rate, the rest of the day at a different rate).

When designing the program, keep in mind that it should be extensible, such that your system can be hooked to another mall and work without inconvenience. It should also be easily maintainable and extendable to new incoming features. Think how you can abstract every aspect.

7. A social network platform

In this project, you will create a social network platform such as Facebook / Twitter, etc. The key note of this project is to start working backwards from the users of your platform. Each user should be able to search for the other user's profiles, follow/unfollow or sent them a friend request, create groups where multiple people can join. Think about the privacy and what data can be displayed only to a specific list of people and what is generally available. Design a system for users to communicate with each other's: may be person-to-person or person-to-group. Think about what other interactions can one user perform with others (like, invites, etc.) and how would you design a notification system.

The possibilities in this project are endless, so think about creating a maintainable, scalable, and easily extensible project, that will allow you to add new features easily. A real social network platform may have a lot of new other functionalities, or some missing functionalities from the example given above. Talk with your friends about what cool feature should your social network platform contain.

8. **Gwent card game**

In this project, you will create a simplified version of the Gwent game, present in the iconic video game series Witcher, using OOP principles learned in this semester. Your task is to design a system that allows two players to complete by placing cards on a board, strategically managing their army's strength to win rounds.
The game will take place on a *board*. Each player will have 3 combat zones: *close*

combat for *infantry* units, *ranged combat* for archers and *siege combat* for *siege engines*. At the start of the game, players will draw 10 cards from the *Deck*, with the possibility of discarding at most 3 cards in order to redraw the number of discarded cards from the deck. Players take turns in placing cards on their side of the board in order to increase the total attack points of their army (siege combat + ranged combat + close combat). A player may skip their turn voluntarily or be forced to skip if they have no cards to play. When both players skip, the total attack points (sum of all zones) are compared and the player with the lower attack loses 1 life point. The game ends when a player loses 2 life points.

You must design at least the following classes: *player* (represents a player, manages their hand, deck, and life points), *board* (represents the playing field, keeping track of cards in each combat zone), *card* (a base class representing different types of cards (army cards, ability cards, etc.)). Your implementation must use the four pillars of OOP. The complexity of this project can be decided by you. For extra complexity, consider adding: *ability cards* (that can trigger effects such as: destroying one enemy card, reducing the attack of all cards in a zone to a specific value), *card effects* (special abilities for army cards, thus making use of the polymorphism of OOP), deck management system, AI opponent, player race (giving different boosts at the start of the game/turn).

Final notes

We will assign each one of you one of these projects automatically, unless you plan to execute your own personal idea. If you don’t like/know anything about the project you were assigned, you can come up with your own idea **IF** the project complexity equals to the one assigned to you.

All these projects are created with the console display as output in mind. If you want to use any other form of display, you are free and encouraged to do so!

If you plan to execute a project with your own idea, it must be **at least equal** in difficulty to the ones above, and you will have to ask prof. Năstac at the course or assistants at the laboratories. If you skip this step, we will assume you are working on the project that was assigned to you and will not be grading your personal work!

If you have any questions about how to implement certain parts of these projects or need more clarifications about the tasks, don’t hesitate to ask!