**Ministerul Educaţiei și Cercetării al Republicii Moldova Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**Laboratory work 3:**

Behavioral Design Patterns

# Elaborated:

st. gr. FAF-222 Șarov Andrei

# Verified:

asist. univ. Furdui Alexandru

# Chişinău - 2024

**Objectives:**

* Study and understand the Behavioral Design Patterns.
* As a continuation of the previous laboratory work, think about what communication between software entities might be involed in your system.
* Implement some additional functionalities using behavioral design patterns.

**Main Tasks:**

1. By extending your project, implement at least 1 behavioral design pattern in your project:

* The implemented design pattern should help to perform the tasks involved in your system.
* The behavioral DPs can be integrated into you functionalities alongside the structural ones.
* There should only be one client for the whole system.

2. Keep your files grouped (into packages/directories) by their responsibilities (an example project structure):

* client;
* domain;
* utilities;
* data(if applies);

3. Document your work in a separate markdown file according to the requirements presented below (the structure can be extended of course):

* Topic of the laboratory work.
* Author.
* Introduction/Theory/Motivation.
* Implementation & Explanation (you can include code snippets as well):
  + Indicate the location of the code snippet.
  + Emphasize the main idea and motivate the usage of the pattern.
* Results/Screenshots/Conclusions;

**Used Design Patterns:**

* **Observer Pattern:** Notifies registered observers when the state of a subject (the Product) changes (e.g., price updates).
* **Command Pattern:** Encapsulates a request as an object (e.g., ChangePriceCommand), allowing for parameterization and queuing of requests.
* **Strategy Pattern:** Enables selecting a behavior at runtime (e.g., different discount strategies) to calculate product prices.

**Implementation:**

**Project Setup**: Created a structured directory with folders for client, domain, and utilities. This organization aids in managing the project's components effectively.

**Class Development**: Implemented core classes, including Product, DiscountStrategy, and command-related classes. Each class was designed with specific responsibilities, aligning with the behavioral design patterns chosen.

**Integration**: Connected the implemented design patterns into the main.py client. This involved using the Observer pattern to update observers on product changes, applying the Command pattern for price changes, and selecting the appropriate Strategy for discount calculations.

**Testing**: Conducted testing to ensure that the system behaved as expected. This included verifying that price updates correctly notified observers and that the correct discount strategy was applied during calculations.

**Conclusions:**

This project successfully demonstrates the application of the Observer, Command, and Strategy design patterns, achieving the objectives of understanding behavioral patterns and enhancing communication between software entities. The implementation showcases flexibility in product pricing and discount management, reinforcing both theoretical and practical knowledge of design patterns.