

Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут» ім. Ігоря Сікорського

## **Звіт**

### **Лабораторна робота №3**

*з дисципліни «Технології та інструменти розробки ПЗ»*

### **«Проектування програмного забезпечення»**

Виконав студент групи: КВ-32

Косарук Захар

Перевірив:

**Київ 2025**

**Мета роботи:** отримати навички застосування методологій проектування програмного забезпечення та складання технічного проекту.

Завдання:

1. Обрати методологію проектування:

а) структурно-функціональну,  
або

б) об'єктно-орієнтовану.

2. Відповідно обраної методології проектування скласти проектну документацію:

а) структурну та функціональну схеми програмного забезпечення (IDF0),  
або

б) діаграми прецедентів (use case), класів (classes) та послідовностей (sequences)

3) оформити звіт із лабораторної роботи. Зміст звіту:

- титульний аркуш із відомостями про виконавця;
- завдання;
- виконане завдання згідно варіанту (схеми або діаграми);
- висновки.

## Хід роботи

1.

Для проєкту «TriggerCalc» було обрано об'єктно-орієнтовану методологію проєктування (ООП), оскільки вона найкраще відповідає специфіці та вимогам, визначеним у попередніх етапах роботи.

**1. Природне моделювання предметної області.** Основні елементи програмного забезпечення — це тригери різних типів (RS, JK, D, T). Кожен такий тригер логічно представляти як окремий **об'єкт**, що має власний стан (поточні виходи) та поведінку (алгоритм обчислення). ООП дозволяє створити клас для кожного типу тригера, що робить архітектуру програми інтуїтивно зрозумілою та наближеною до реальних сутностей.

**2. Забезпечення масштабованості.** Однією з нефункціональних вимог проєкту є **масштабованість** — «можливість розширення функціоналу для інших типів логічних елементів у майбутньому». ООП ідеально підходить для цього завдання. Можна створити базовий клас Trigger, а потім додавати нові логічні елементи як його класи-нащадки. Це дозволить легко розширювати функціонал, не вносячи значних змін у вже написаний та протестований код.

**3. Підвищення надійності коду.** Завдяки **інкапсуляції**, з'являється можливість приховати складну логіку обчислень всередині кожного об'єкта-тригера. Інтерфейс користувача буде взаємодіяти з ними через простий набір публічних методів. Це знизить ризик виникнення помилок і спростить подальшу підтримку програми, що відповідає вимозі про **надійність**.

**4. Повторне використання коду.** Загальні для всіх типів тригерів характеристики та методи можна реалізувати в базовому класі, а конкретні типи тригерів будуть їх **успадковувати**. Це дозволить уникнути дублювання коду та прискорить розробку.

На відміну від структурно-функціонального підходу, який, імовірно, вимагав би створення складних умовних конструкцій (if/switch) для обробки логіки різних тригерів, об'єктно-орієнтований підхід забезпечить більш гнучку, надійну та легку в підтримці архітектуру. Саме тому ООП є оптимальним вибором для успішної реалізації проєкту «TriggerCalc».

2.

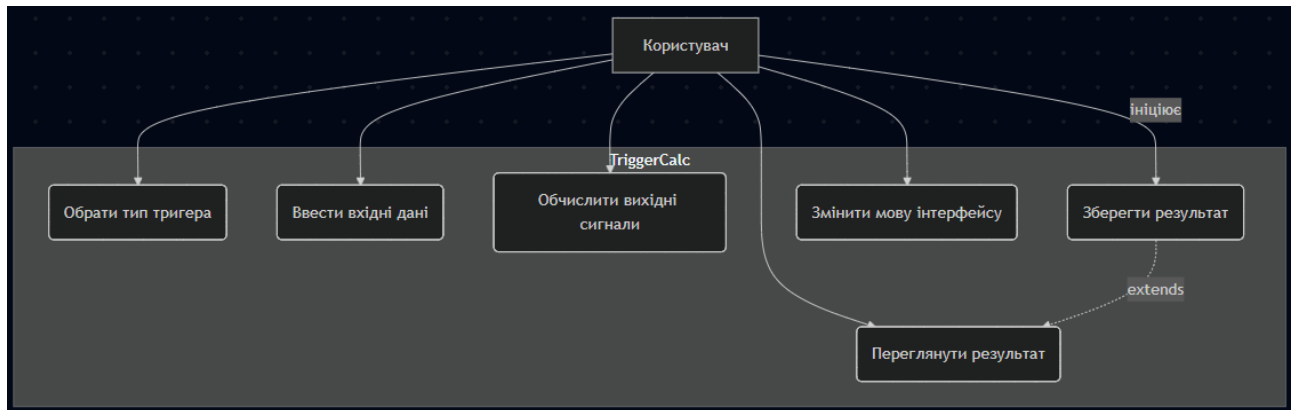


Рис. 1 — Діаграма прецедентів для системи «TriggerCalc»

- **Актор:** Єдиним актором системи є **Користувач**, в ролі якого виступає студент або викладач, що вивчає чи викладає схемотехніку.
- **Основні прецеденти:**
  - **Обрати тип тригера:** надання користувачу можливості вибрати один із доступних типів логічних елементів (RS, JK, D, T).
  - **Ввести вхідні дані:** введення початкового стану та послідовності вхідних сигналів для розрахунку.
  - **Обчислити вихідні сигнали:** запуск основного алгоритму програми для отримання результату.
  - **Переглянути результат:** відображення обчислених вихідних сигналів у зручному для користувача форматі (таблиця, діаграма).
  - **Змінити мову інтерфейсу:** можливість перемикання мови програми (українська/англійська).
- **Зв'язок <<extends>>:** Прецедент "Зберегти результат" розширює прецедент "Переглянути результат". Це означає, що збереження результату є **опціональною** функцією, яка стає доступною лише після того, як основний сценарій (перегляд результату) було виконано. Користувач не може зберегти дані, попередньо їх не обчисливши та не переглянувши.

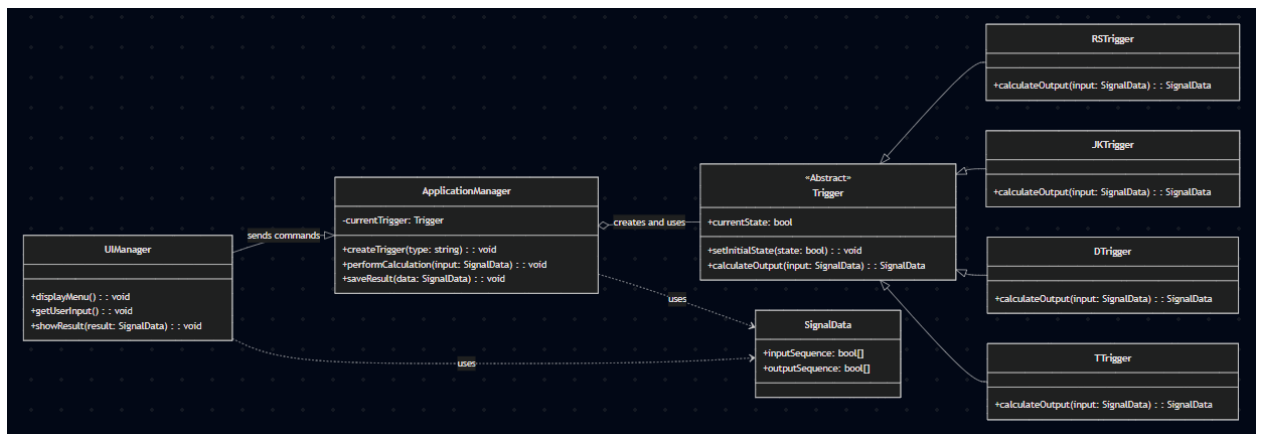


Рис. 2 — Діаграма класів для системи «TriggerCalc»

- **Trigger (Абстрактний клас):** Це базовий, узагальнений клас для всіх тригерів. Він визначає спільний інтерфейс та поведінку.
  - **Атрибути:** currentState (зберігає поточний стан тригера).
  - **Методи:** calculateOutput(input) (абстрактний метод, який буде реалізовано в кожному конкретному класі тригера), setInitialState(state) (встановлює початковий стан).
- **RSTrigger, JKTrigger, DTrigger, TTrigger (Конкретні класи):** Ці класи успадковують базовий функціонал від класу Trigger.
  - **Спадкування:** Кожен з цих класів є нащадком Trigger.
  - **Методи:** Кожен клас має власну, унікальну реалізацію методу calculateOutput(input), яка відповідає логіці роботи саме цього типу тригера.
- **ApplicationManager (Керуючий клас):** Це "мозок" програми. Він координує взаємодію між інтерфейсом та логікою обчислень.
  - **Атрибути:** currentTrigger (зберігає екземпляр обраного тригера).
  - **Методи:** createTrigger(type) (створює об'єкт потрібного тригера), performCalculation(input) (запускає обчислення), saveResult(data) (зберігає результат у файл).
- **UIManager (Клас інтерфейсу):** Цей клас відповідає за всю взаємодію з користувачем: відображення меню, отримання даних та виведення результатів.
  - **Асоціація:** Він пов'язаний з ApplicationManager, якому передає команди від користувача.

○ **Методи:** `displayMenu()` (показує головне меню), `getUserInput()` (отримує від користувача тип тригера та сигнали), `showResult(result)` (відображає результат).

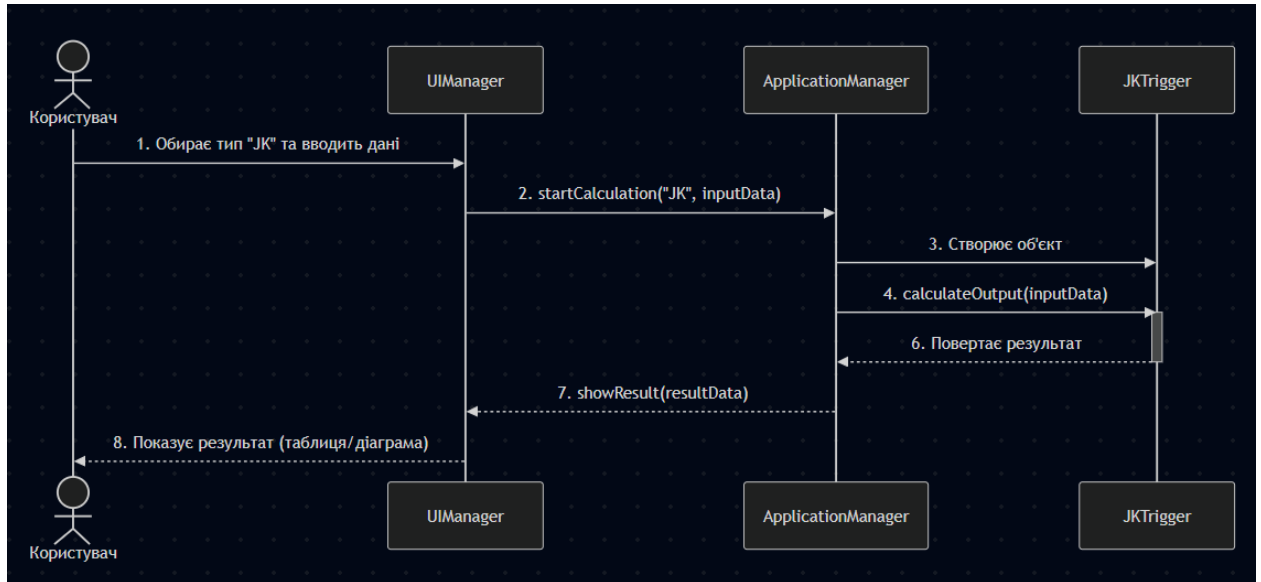


Рис. 3 — Діаграма послідовностей для системи «TriggerCalc»

**Сценарій для діаграми:** "Успішний розрахунок вихідних сигналів для JK-тригера".

Цей сценарій описує найпоширеніший шлях користувача: від вибору типу тригера до отримання результату.

• **Учасники (Об'єкти):**

- :Користувач: Актор, що ініціює весь процес.
- :UIManager: Об'єкт, що відповідає за графічний інтерфейс.
- :ApplicationManager: "Мозок" програми, що керує логікою.
- :JKTrigger: Конкретний екземпляр класу тригера, створений для обчислень.

**Послідовність взаємодії:**

1. **Користувач** обирає тип тригера ("JK") та вводить дані в :UIManager.
2. :UIManager надсилає запит на обчислення до :ApplicationManager, передаючи тип тригера та вхідні дані.
3. :ApplicationManager створює об'єкт :JKTrigger.

4. :ApplicationManager викликає метод calculateOutput() у щойно створеного об'єкта :JKTrigger, передаючи йому вхідні сигнали.
5. :JKTrigger виконує внутрішні обчислення згідно зі своєю логікою.
6. Після завершення обчислень :JKTrigger повертає результат (вихідні сигнали) до :ApplicationManager.
7. :ApplicationManager надсилає отриманий результат до :UIManager для відображення.
8. :UIManager показує результат **Користувачу**.

### Висновки

В ході виконання лабораторної роботи №3 було успішно завершено етап проєктування програмного забезпечення «TriggerCalc», що є логічним продовженням попередніх етапів аналізу вимог та передпроектної підготовки.

Було прийнято ключове архітектурне рішення — обрано **об'єктно-орієнтовану методологію проєктування**. Цей вибір обґрунтовано її перевагами для даного проєкту, а саме можливістю природного моделювання предметної області, забезпеченням високої масштабованості та надійності коду завдяки інкапсуляції.

Відповідно до обраної методології було розроблено повний комплект проєктної документації з використанням стандартної нотації UML:

1. **Діаграма прецедентів (Use Case Diagram)** дозволила чітко візуалізувати взаємодію користувача з системою та визначити всі ключові функціональні вимоги, такі як вибір тригера, введення даних та обчислення результатів.
2. **Діаграма класів (Class Diagram)** сформувала статичну структуру та архітектурний "скелет" майбутнього додатку. Було спроєктовано основні класи (Trigger, ApplicationManager, UIManager), визначено їхні атрибути, методи та зв'язки, зокрема спадкування та асоціацію, що заклало основу для гнучкої та розширюваної системи.
3. **Діаграма послідовності (Sequence Diagram)** продемонструвала динамічну поведінку системи на прикладі конкретного сценарію. Вона наочно показала, як об'єкти будуть взаємодіяти між собою в часі для виконання головного завдання — обчислення вихідних сигналів.

Таким чином, у процесі роботи було здобуто наступні **практичні навички**:

- Застосування принципів об'єктно-орієнтованого підходу до проєктування ПЗ.
- Моделювання функціональних вимог та статичної структури системи за допомогою UML-діаграм.
- Створення комплексної проєктної документації, яка слугуватиме чітким технічним планом для етапу розробки.

Виконана робота створює міцний та добре продуманий фундамент для подальшої програмної реалізації проєкту «TriggerCalc».