***Звіт до лабораторної №4***

з дисципліни «Проектування програмних систем»

студентки 301-А групи

Біляр Іванни

***Тема:*** «Застосування патернів програмування»

*Предметна область* – Курси іноземних мов

***Завдання***

1. Аналізуючи діаграми класів (Лабораторна робота №3), обґрунтувати

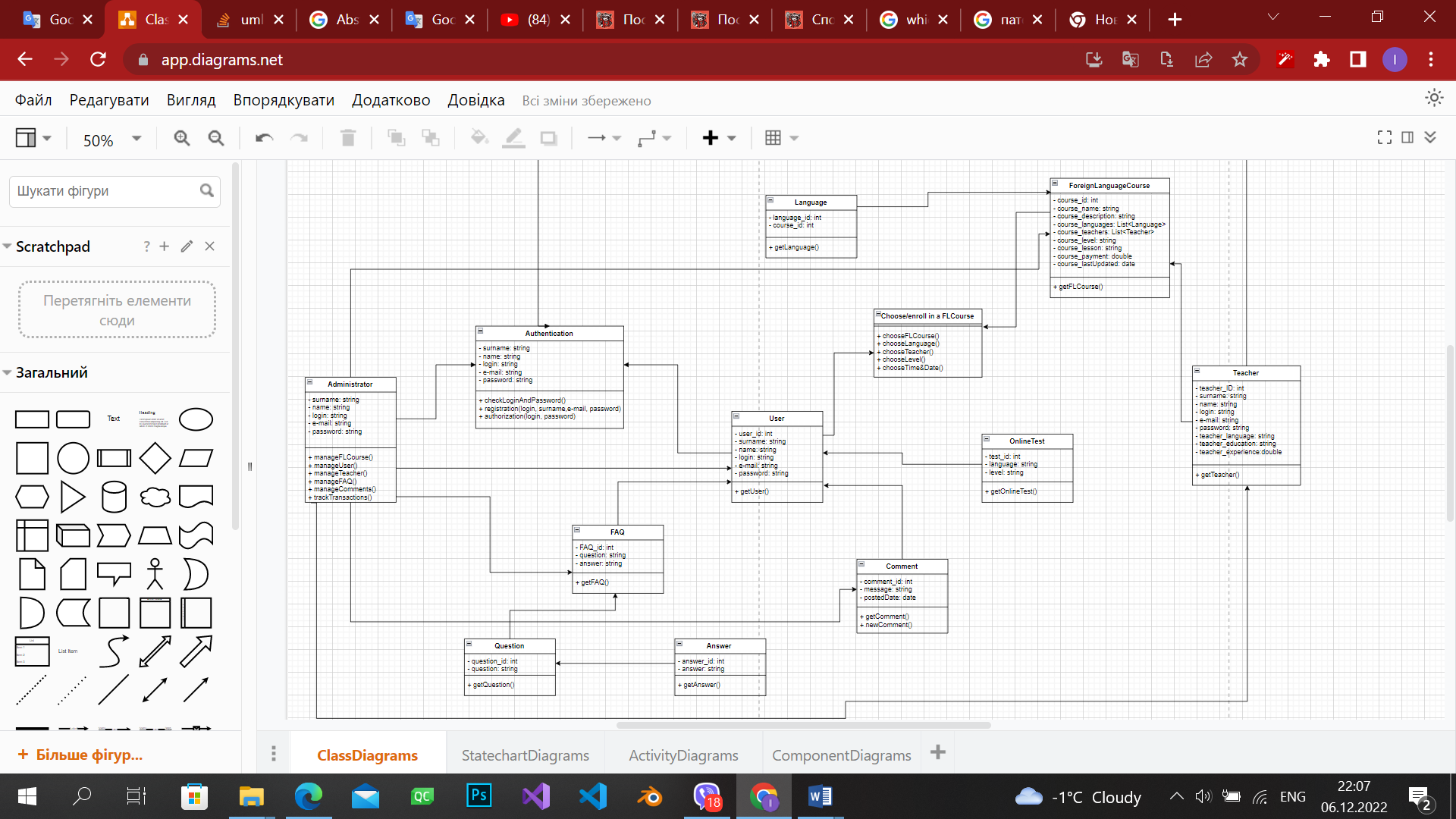
застосування підібраних для реалізації патернів.

2. Використовуючи лабораторні роботи із дисциплін професійної

підготовки, створити репозиторій, розмістити файли проектів. Надати

доступ викладачам на github / gitlab ресурсі.

***Діаграма класів (Class Diagrams)***



Аналізуючи свою діаграму класів я б використала наступні

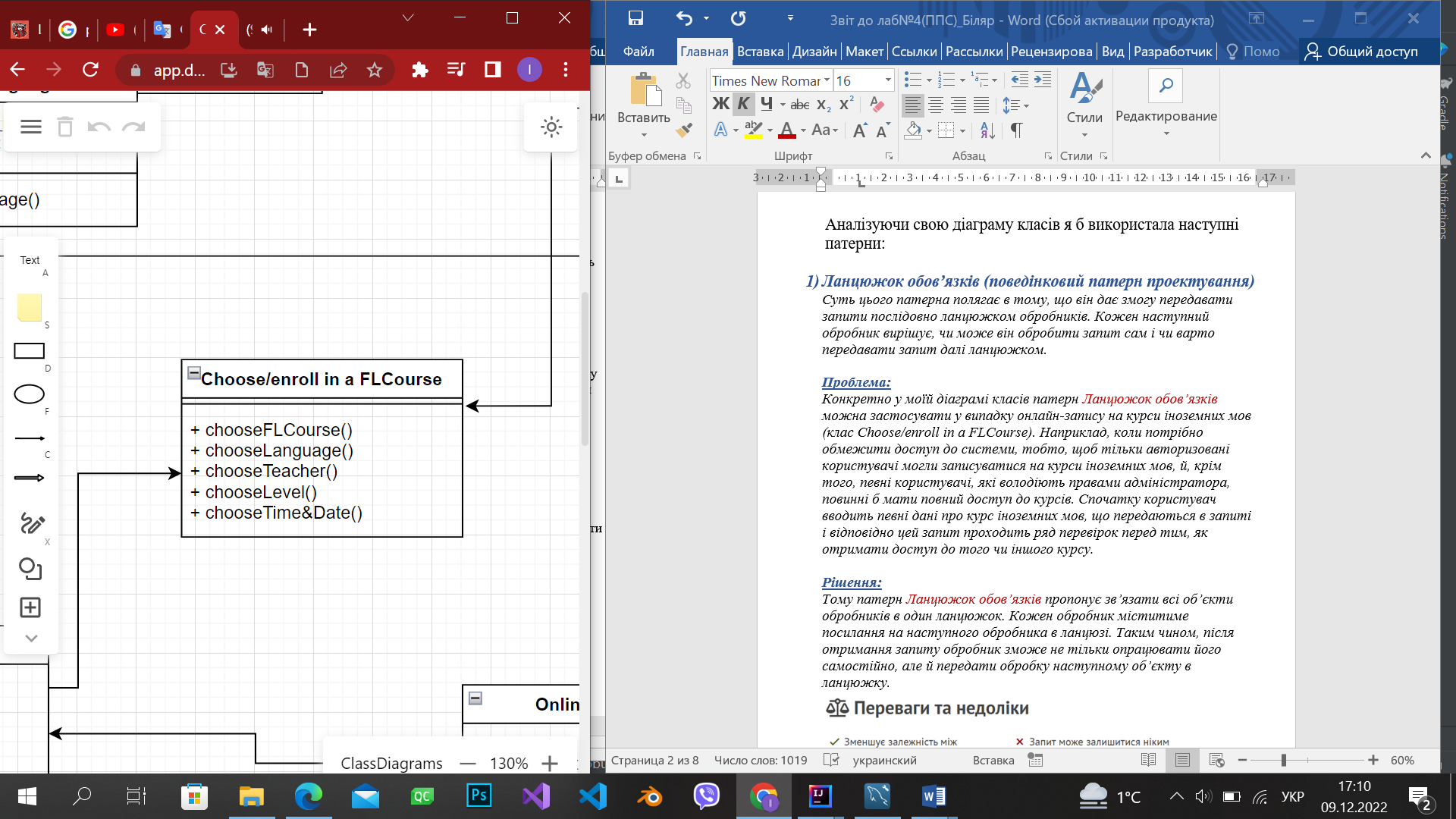
патерни:

1. ***Ланцюжок обов’язків (поведінковий патерн проектування)***

*Суть цього патерна полягає в тому, що він дає змогу передавати запити послідовно ланцюжком обробників. Кожен наступний обробник вирішує, чи може він обробити запит сам і чи варто передавати запит далі ланцюжком.*

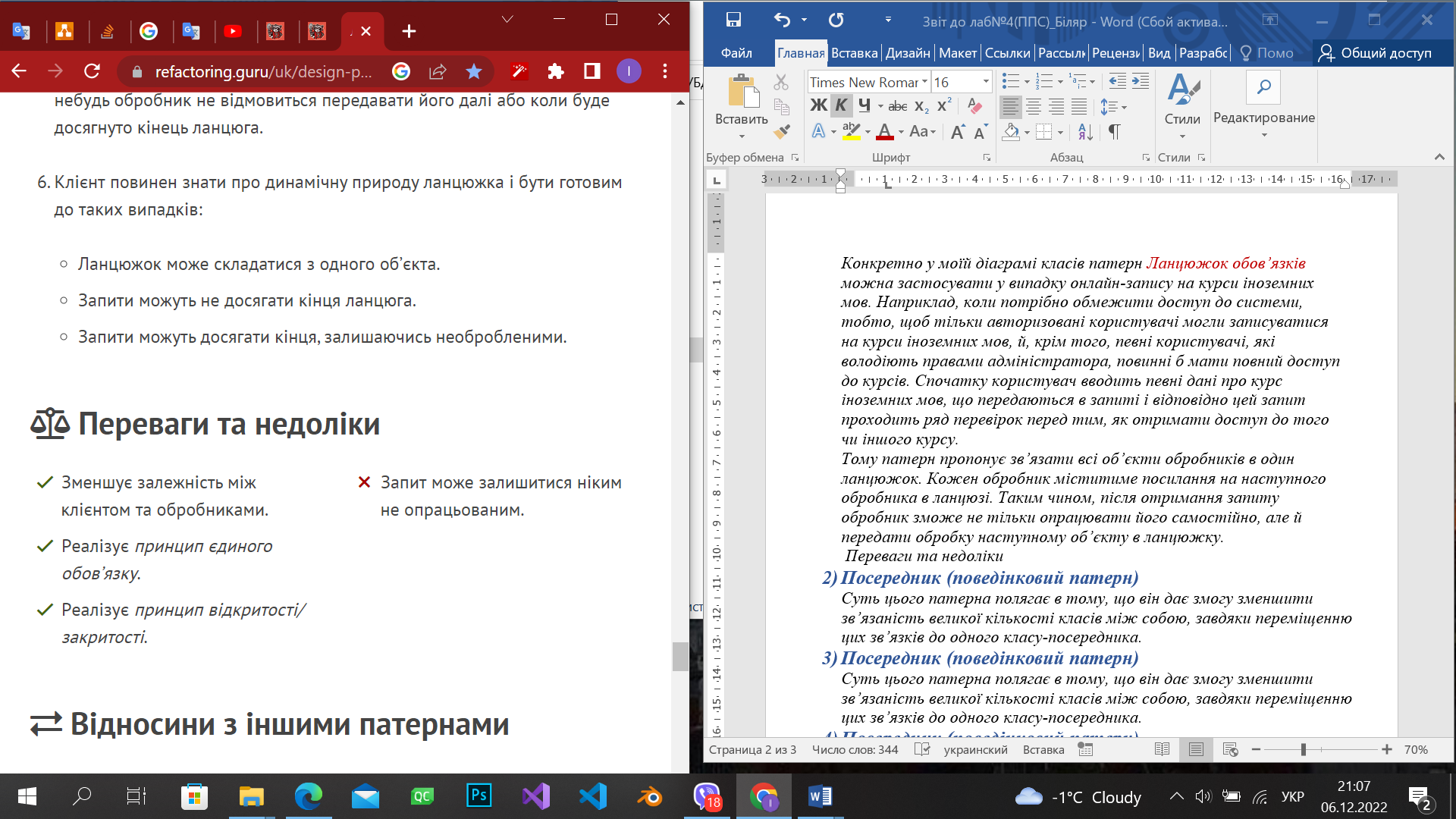
***Проблема:***

*Конкретно у моїй діаграмі класів патерн Ланцюжок обов’язків можна застосувати у випадку онлайн-запису на курси іноземних мов (клас Choose/enroll in a FLCourse). Наприклад, коли потрібно обмежити доступ до системи, тобто, щоб тільки авторизовані користувачі могли записуватися на курси іноземних мов, й, крім того, певні користувачі, які володіють правами адміністратора, повинні б мати повний доступ до курсів. Спочатку користувач вводить певні дані про курс іноземних мов, що передаються в запиті і відповідно цей запит проходить ряд перевірок перед тим, як отримати доступ до того чи іншого курсу.*



***Рішення:***

*Тому патерн Ланцюжок обов’язків пропонує зв’язати всі об’єкти обробників в один ланцюжок. Кожен обробник міститиме посилання на наступного обробника в ланцюзі. Таким чином, після отримання запиту обробник зможе не тільки опрацювати його самостійно, але й передати обробку наступному об’єкту в ланцюжку.*



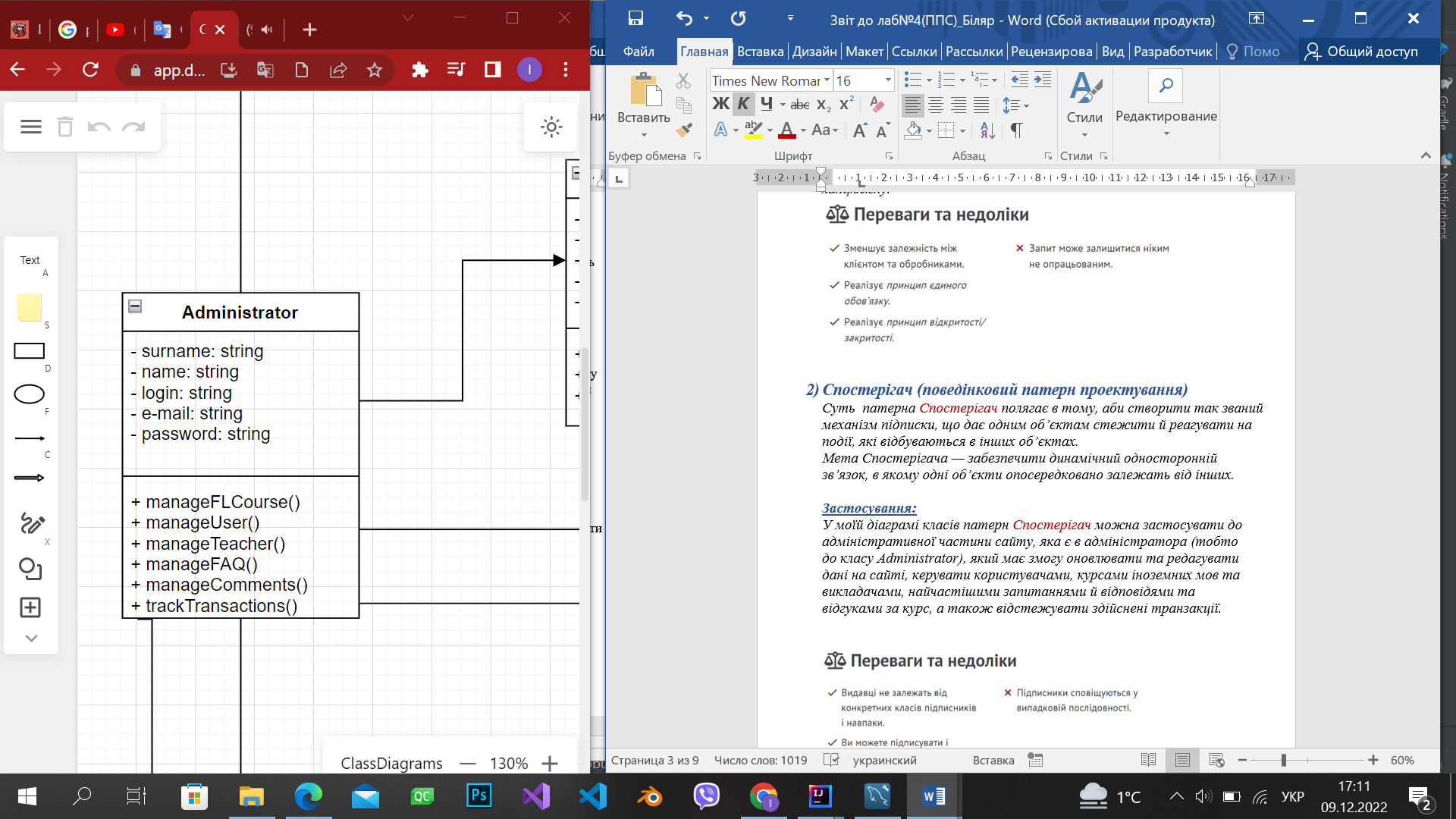
1. ***Спостерігач (поведінковий патерн проектування)***

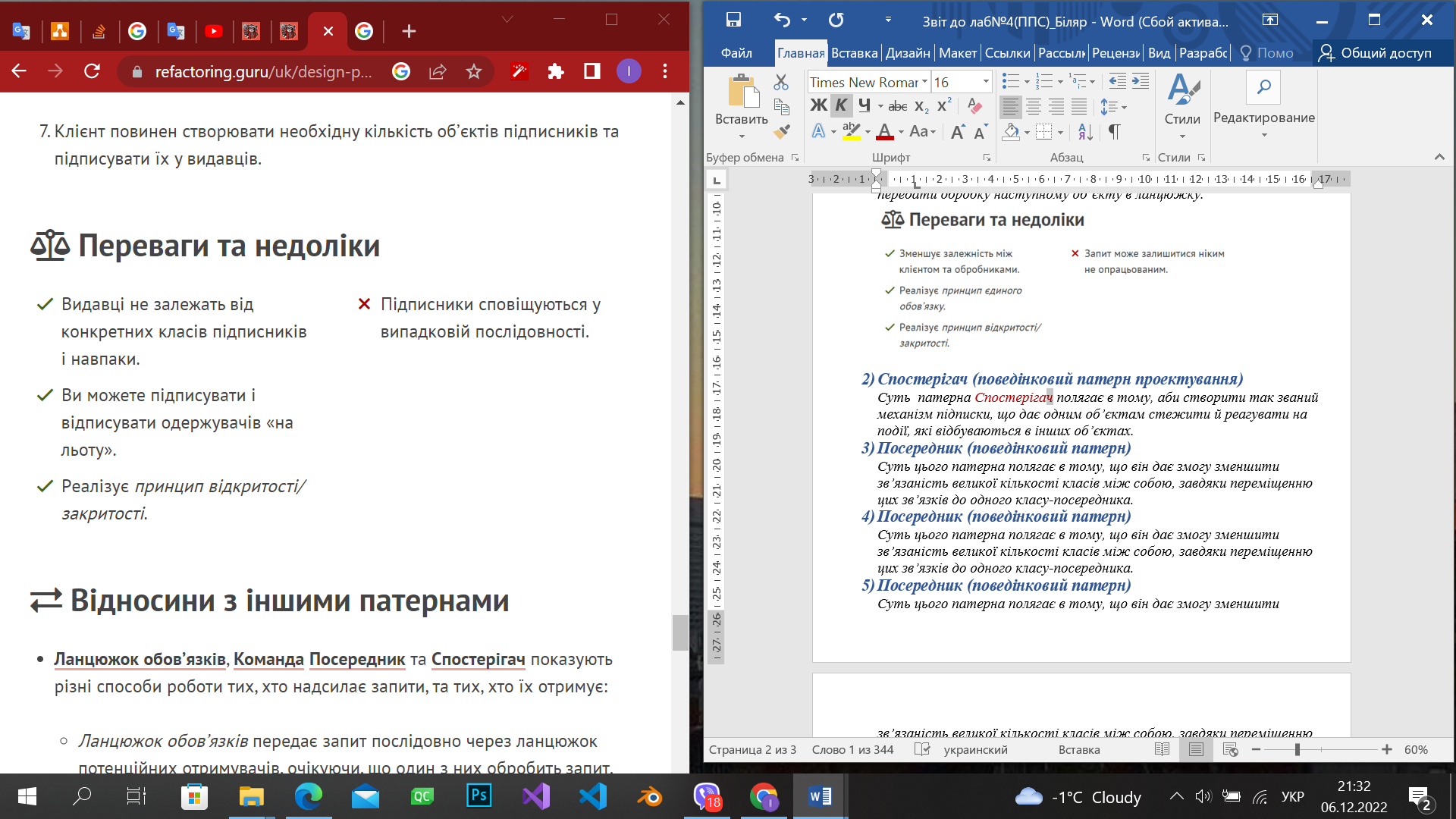
*Суть патерна Спостерігач полягає в тому, аби створити так званий механізм підписки, що дає одним об’єктам стежити й реагувати на події, які відбуваються в інших об’єктах.*

*Мета Спостерігача — забезпечити динамічний односторонній зв’язок, в якому одні об’єкти опосередковано залежать від інших.*

***Застосування:***

*У моїй діаграмі класів патерн Спостерігач можна застосувати до адміністративної частини сайту, яка є в адміністратора (тобто до класу Administrator), який має змогу оновлювати та редагувати дані на сайті, керувати користувачами, курсами іноземних мов та викладачами, найчастішими запитаннями й відповідями та відгуками за курс, а також відстежувати здійснені транзакції.*



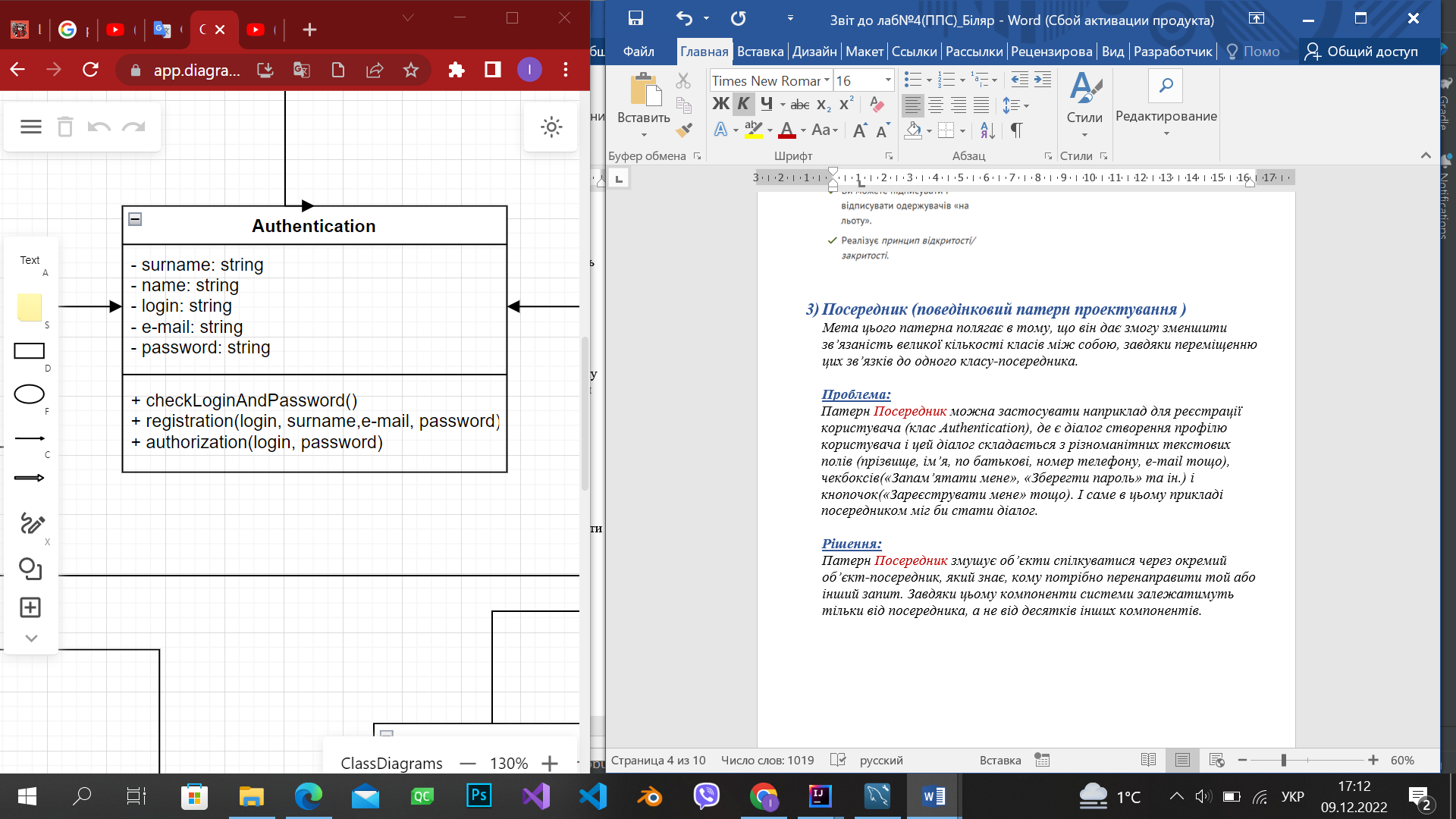


1. ***Посередник (поведінковий патерн проектування )***

*Мета цього патерна полягає в тому, що він дає змогу зменшити зв’язаність великої кількості класів між собою, завдяки переміщенню цих зв’язків до одного класу-посередника.*

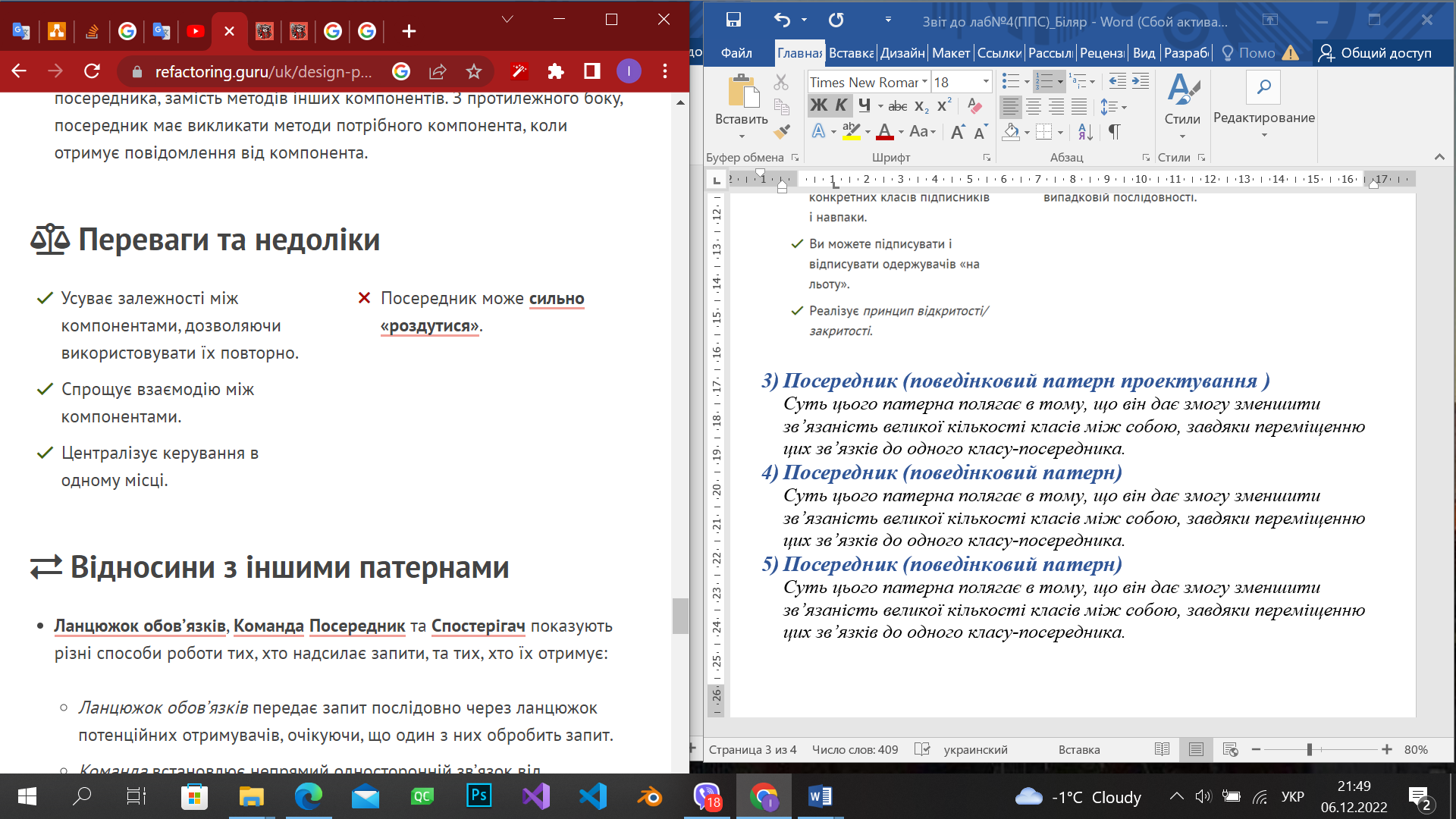
***Проблема:***

*Патерн Посередник можна застосувати наприклад для реєстрації користувача (клас Authentication), де є діалог створення профілю користувача і цей діалог складається з різноманітних текстових полів (прізвище, ім’я, по батькові, номер телефону, e-mail тощо), чекбоксів(«Запам’ятати мене», «Зберегти пароль» та ін.) і кнопочок(«Зареєструвати мене» тощо). І саме в цьому прикладі посередником міг би стати діалог.*



***Рішення:***

*Патерн Посередник змушує об’єкти спілкуватися через окремий об’єкт-посередник, який знає, кому потрібно перенаправити той або інший запит. Завдяки цьому компоненти системи залежатимуть тільки від посередника, а не від десятків інших компонентів.*

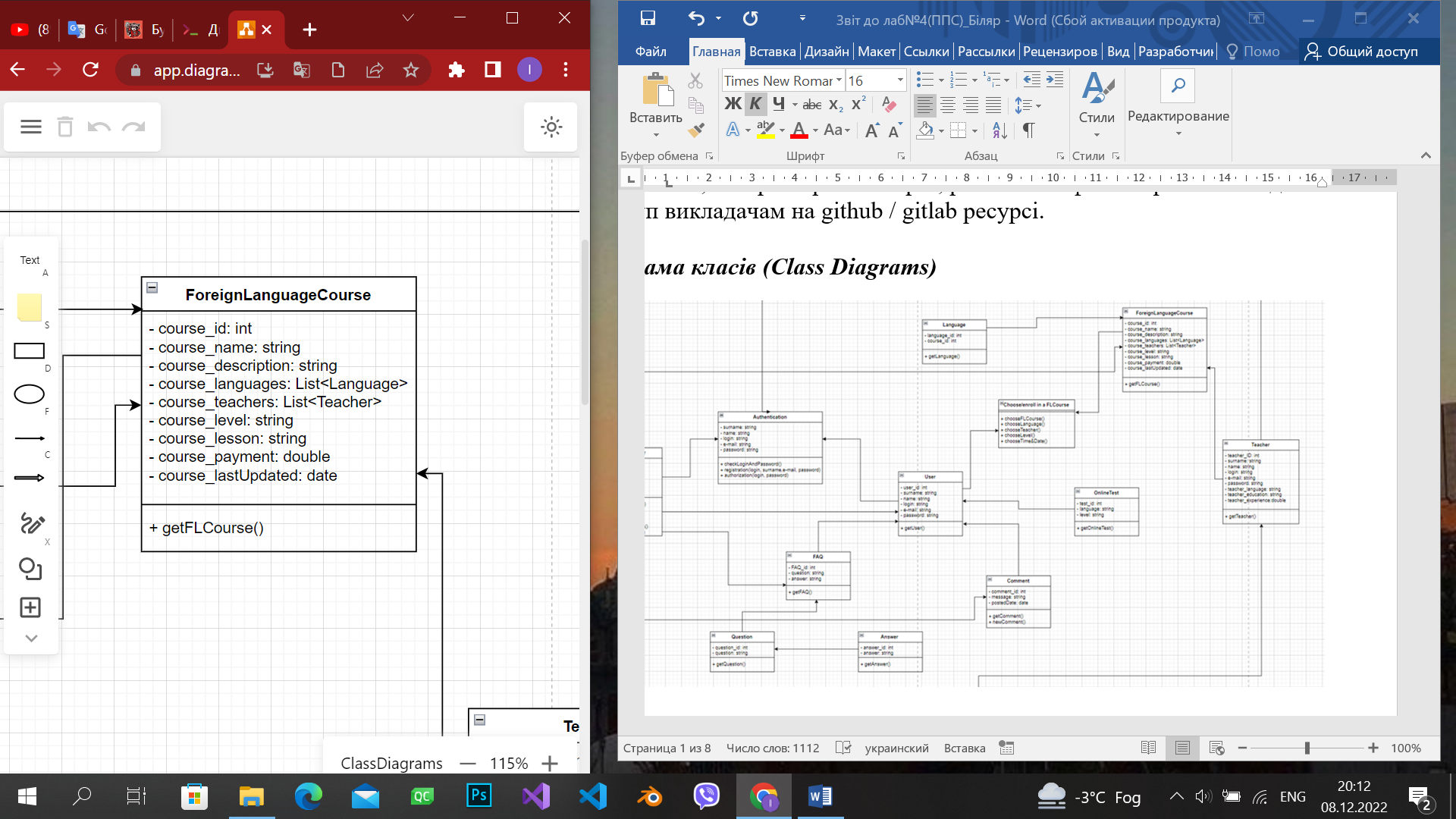


1. ***Будівельник (породжувальний патерн проектування)***

*Суть цього патерна полягає в тому, що він дає змогу створювати складні об’єкти крок за кроком. Будівельник дає можливість використовувати один і той самий код будівництва для отримання різних відображень об’єктів.*

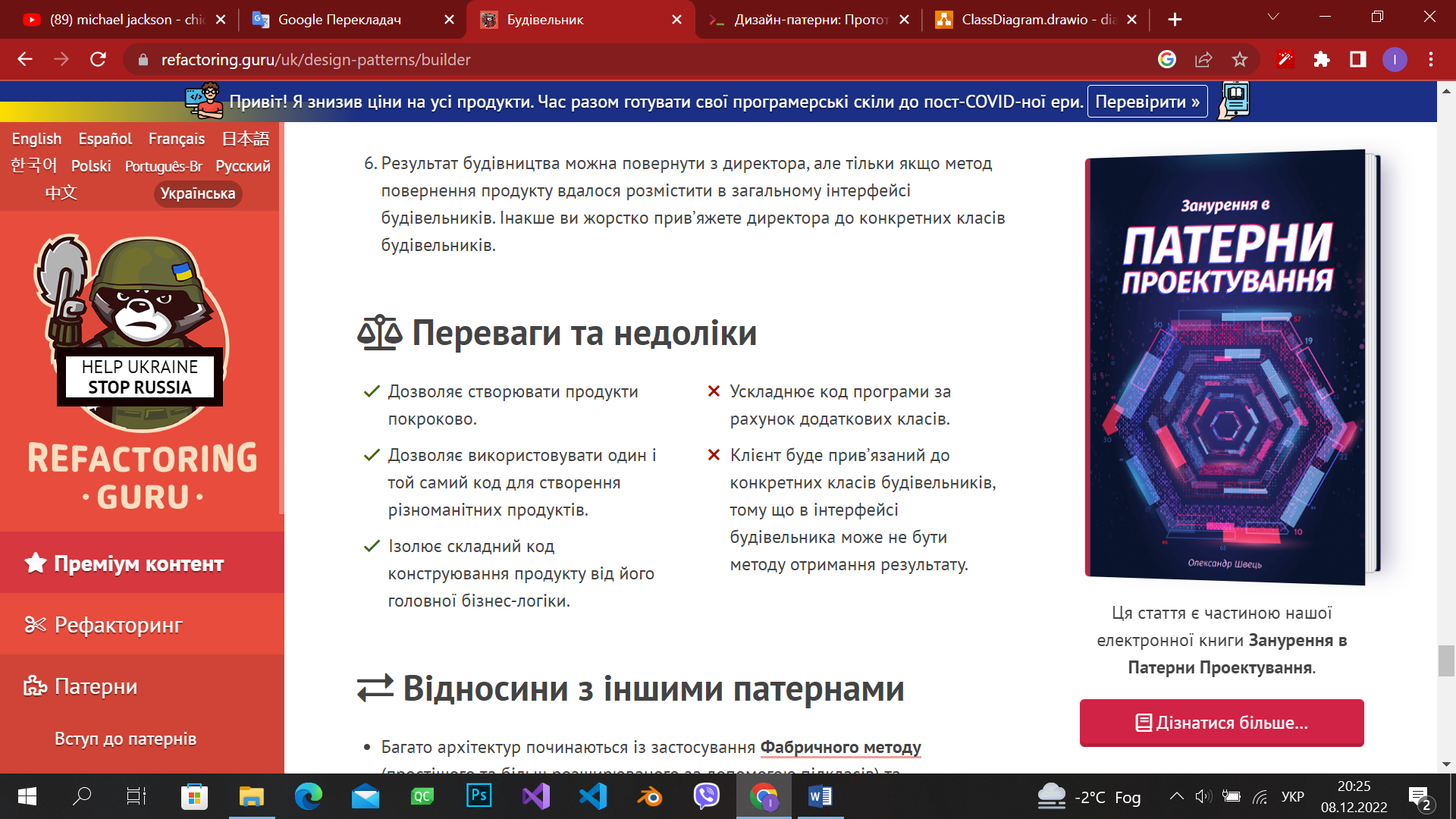
***Проблема:***

*У моїй діаграмі класів патерн Будівельник можна застосувати до класу ForeignLanguageCourse, який є класом створення об’єкту Course. Це складний об’єкт, який містить багато параметрів і вимагає кропіткої покрокової ініціалізації безлічі полів і вкладених об’єктів.*



***Рішення:***

*Тому патерн Будівельник пропонує винести конструювання об’єкта за межі його власного класу, доручивши цю справу окремим об’єктам, які називаються будівельниками. А саме патерн пропонує розбити процес конструювання об’єкта на окремі кроки. Щоб створити об’єкт, вам потрібно по черзі викликати методи будівельника. До того ж не потрібно викликати всі кроки, а лише ті, що необхідні для виробництва об’єкта певної конфігурації.*

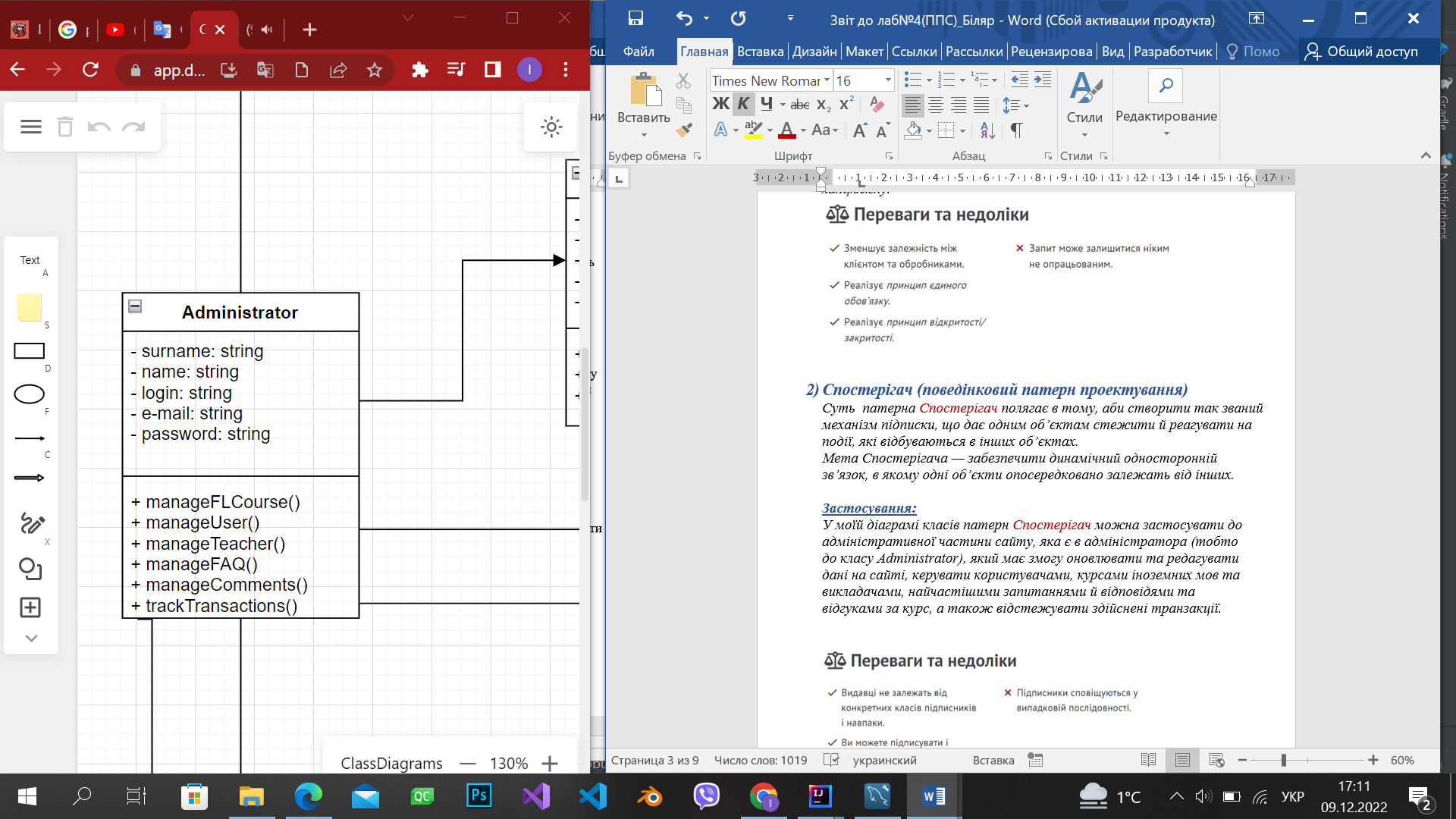


1. ***Одинак (породжувальний патерн проектування)***

*Суть цього патерна полягає в тому, що клас має лише один екземпляр, доступний усім клієнтам (наприклад, спільний доступ до бази даних з різних частин програми), та надає глобальну точку доступу до нього.*

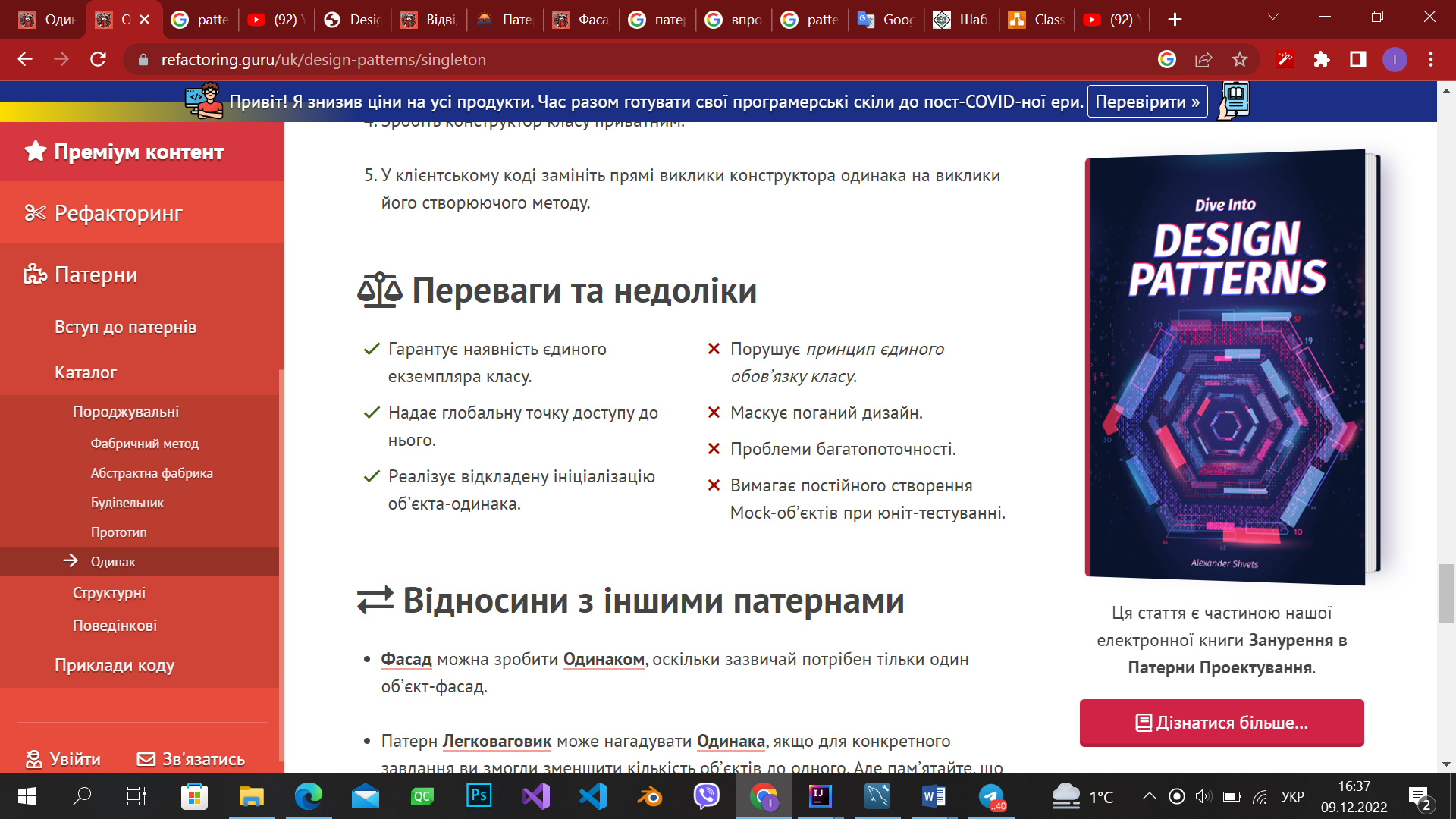
***Застосування:***

*У моїй діаграмі класів патерн Одинак можна застосувати до адміністративної частини сайту, яка є в адміністратора, профіль якого зареєстрований у базі даних (тобто до класу Administrator), щоб таким чином повний контроль над сайтом курсів іноземних мов був тільки в одного користувача.*



***Рішення:***

*Тому патерн Одинак пропонує приховати типовий конструктор та створити публічний статичний метод, який і контролюватиме життєвий цикл об’єкта-одинака. І відповідно, якщо буде доступ до класу одинака, отже, буде і доступ до цього статичного методу.*



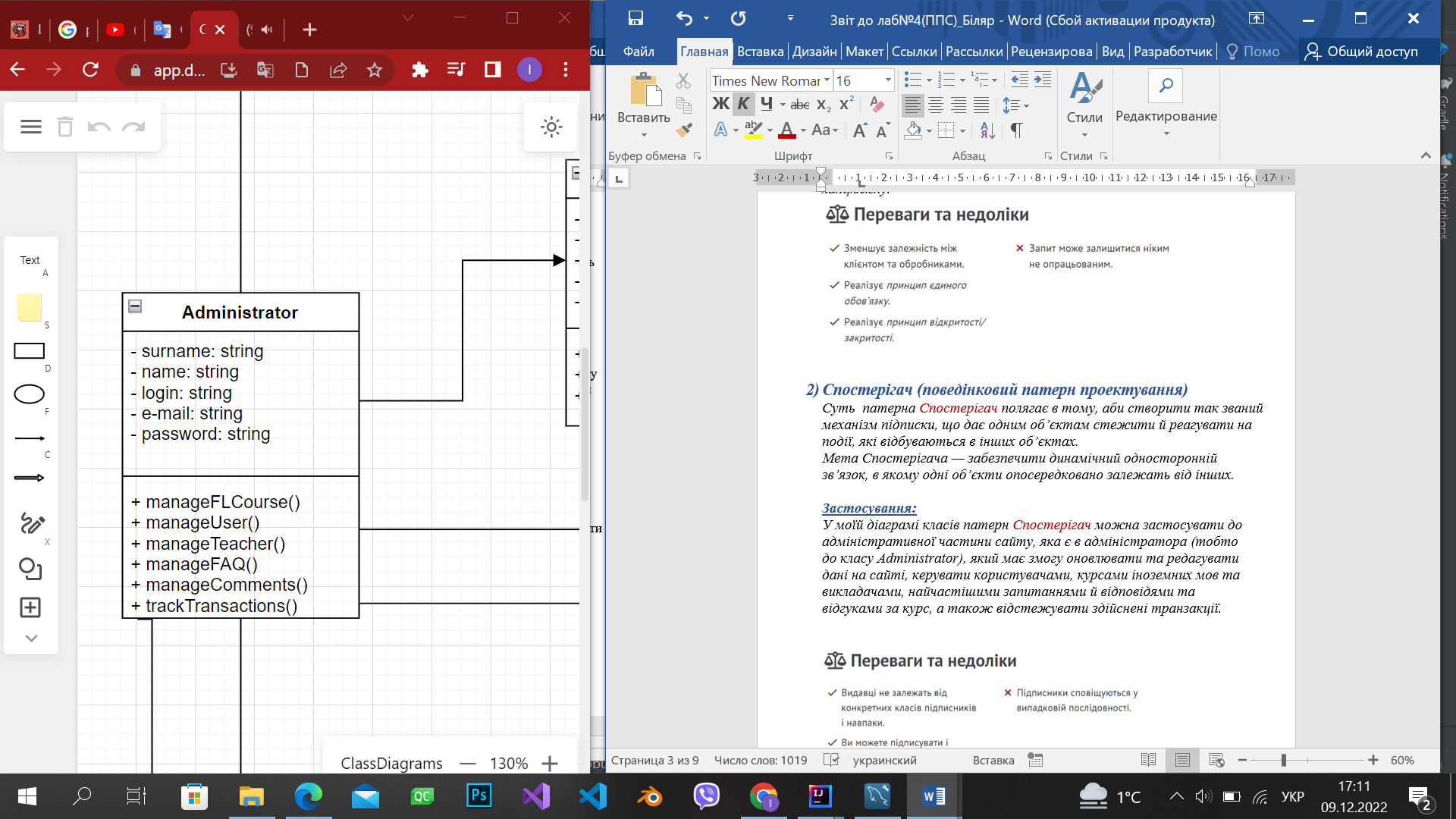
1. ***Шаблонний метод (поведінковий патерн проектування)***

*Суть цього патерна полягає в тому, що він визначає кістяк алгоритму, перекладаючи відповідальність за деякі його кроки на підкласи. Патерн дозволяє підкласам перевизначати кроки алгоритму, не змінюючи його загальної структури.*

***Проблема:***

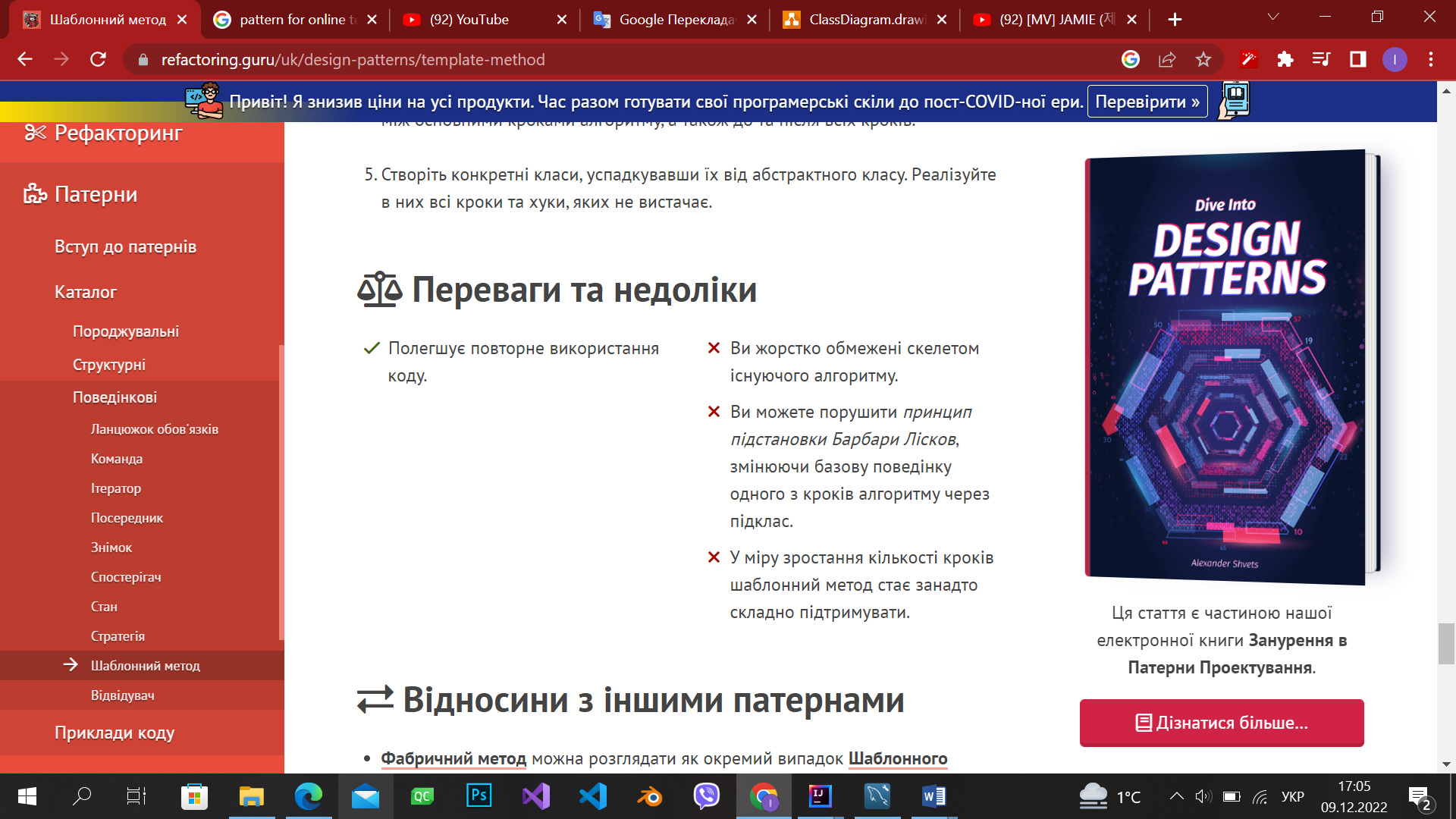
*Конкретно у моїй діаграмі класів патерн Шаблонний метод можна застосувати у для методу manageFLCourse() в класі Administrator. Тобто, можна виокремити окремі методи, щоб видобути необхідну та корисну інформацію про курс іноземних мов. Наприклад:*

* *метод для перевірки чи існує курс за певною мовою, яку вводить користувач або введеною ціною;*
* *метод для надання повної інформації про курс іноземних мов (короткий опис курсу, тривалість навчання, ціна тощо);*
* *метод для того, щоб отримати інформацію від користувача для зміни та її опрацювання.*



***Рішення:***

*Тому патерн Шаблонний метод пропонує розбити алгоритм на послідовність кроків, описати ці кроки в окремих методах і викликати їх в одному шаблонному методі один за одним. Це дозволить підкласам перевизначити деякі кроки алгоритму, залишаючи без змін його структуру та інші кроки, які для цього підкласу не є важливими. У моєму прикладі з керуванням інформації про курс іноземних мов, можна створити загальний базовий клас для всіх алгоритмів і цей клас буде складатися з шаблонного методу, який послідовно викликатиме кроки отримання запитів про курс.*



1. ***Стратегія (поведінковий патерн проектування)***

*Суть цього патерна полягає в тому, що він визначає сімейство схожих алгоритмів і розміщує кожен з них у власному класі. Після цього алгоритми можна заміняти один на інший прямо під час виконання програми.*

***Проблема:***

*Конкретно у моїй діаграмі класів патерн Стратегія можна застосувати у випадку онлайн-запису на курси іноземних мов (клас Choose/enroll in a FLCourse), а саме до методу ChooseFLCourse(). Таким чином, це дозволить змінювати поведінку для запису/вибору різних курсів з іноземних мов. Тобто, для того, щоб отримати розклад занять з обраного курсу іноземних мов, користувач має спочатку заплатити за курс та обрати для себе зручний час.*

***Рішення:***

*Тому патерн Стратегія пропонує визначити сімейство схожих алгоритмів, які часто змінюються або розширюються, й винести їх до власних класів, які називають стратегіями. Замість того, щоб початковий клас сам виконував той чи інший алгоритм, він відіграватиме роль контексту, посилаючись на одну зі стратегій та делегуючи їй виконання роботи. Також важливо, щоб всі стратегії мали єдиний інтерфейс. Використовуючи цей інтерфейс, контекст буде незалежним від конкретних класів стратегій. Тому з іншого боку, можна буде змінювати та додавати нові види алгоритмів, не чіпаючи код контексту.*

