Івано-Франківський національний технічний

університет нафти і газу

Кафедра

інженерії програмного забезпечення

Лабораторна робота №2

Візуалізація шаблонів

Виконав

Ст. гр. ІП-22-1

Хімій Денис

Перевірила

Піх М.М.

Івано-Франківськ

2024

Мета: побудувати діаграму класів проєкту із викоРистанням шаблонів проєктування. У діаграмі вказати відношення між класами, типи даних для полів та методів. Виділити викоРистані шаблони проєктування

Хід виконання роботи

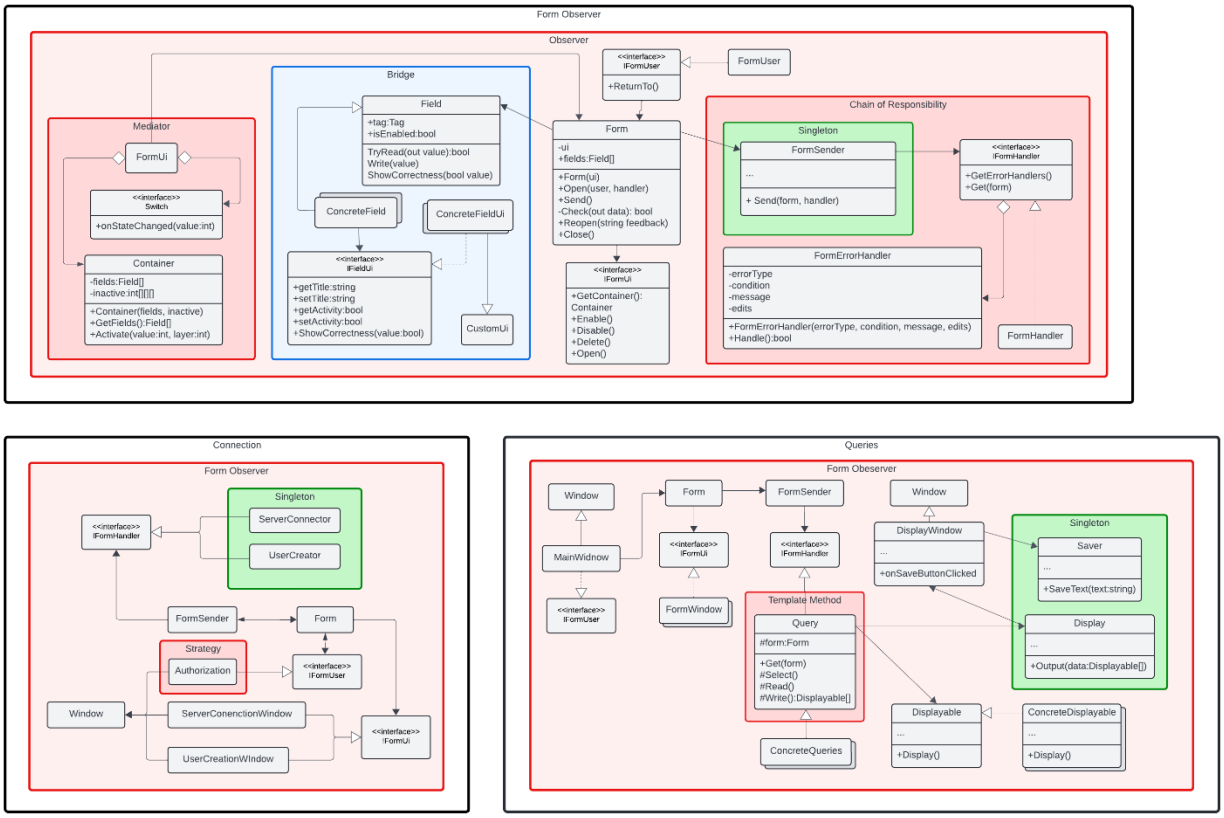
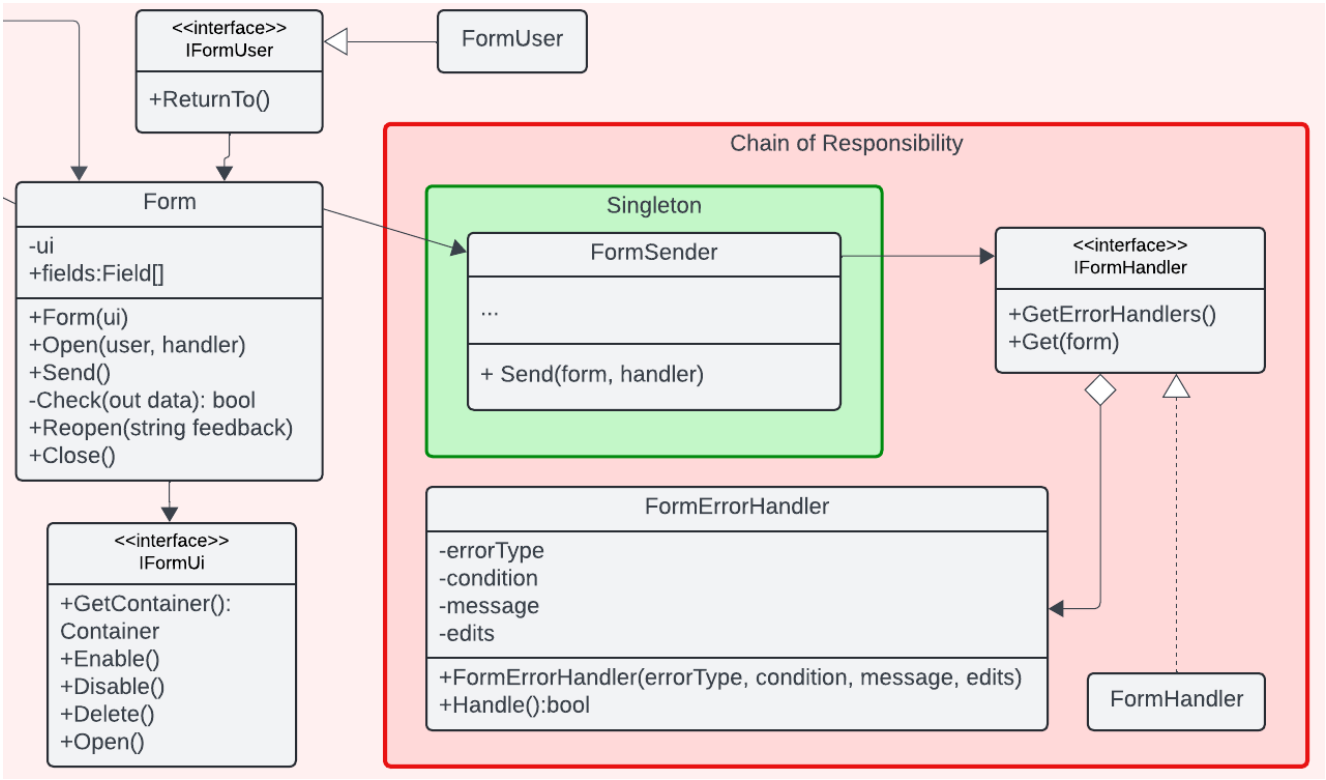


Рис. 1 – Загальна діаграма класів проєкту

Спочатку слід прояснити деякі **особливості діаграми класів**, наведеної в ілюстрації вище, оскільки її візуальне сприйняття сильно спростить розуміння наведеного матеріалу. Загалом **діаграма поділена на 3 блоки**: **Шаблон** (Form Observer), та реалізацію на його основі **Авторизації** (Connection) і **Здійснення запитів** (Queries). Кожен блок окрім самих класів та інтерфейсів містить **підблоки**. Вони **означають шаблони проєктування**, **які імплементують класи поміщені в них**. Зеленим показано породжувальні патерни, блакитним – структурні, червоним – поведінкові.

Основний Шаблон

Рис. 2 – UML-діаграма класів архітектури Проєкту.



Тепер власне Архітектурний Патерн (див. розділ 2). Він не дарма називається **Form Observer,** оскільки **є окремим випадком шаблону Observer** (Спостерігач). Як відомо суть цього шаблону полягає в тому, що певні об’єкти (publishers) можуть надсилати іншим об’єктам (subscribers) інформацію про зміну свого стану.

Те саме реалізовано тут. **Publisher – це діалогове вікно** (форма), **Subscriber – це об’єкт, який використовує дані з форми** для виконання обраної користувачем дії. **Підписку** subscriber’a publisher на publisher **здійснює елемент інтерфейсу, який користувач використовує для виклику форми** (наприклад: кнопка авторизації чи здійснення конкретного запиту)

Тепер детальніше. Інтерфейс користувача (надалі **UI** для уникнення плутанини з інтерфейсами класів (надалі CI)) **для виклику форми мусить реалізувати CI IFormUser (user)**. Він містить метод, який дозволяє повернутись до UI після завершення роботи.

Власне **форма поділена на 2 частини**: **клас Form та інтерфейс FormUi**. **FormUi містить** наступне. **Поля з даними**, які можна використати як для зчитування значень так і для їх заповнення ззовні (наприклад для показу помилок). **Методи для її відкриття, приховування, показу та закриття**. Цей CI реалізують конкретні діалогові вікна, які власне містять всі необхідні поля. **FormUi** лише **надає впорядкований доступ** до них, при цьому **роблячи клас Form незалежним від** реалізації **UI** в застосунку.

Form містить власне **екземпляр FormUi та набір методів для роботи з ним**. Це методи для відкриття форми, надсилання даних, їх попередньої перевірки та повторного відкриття форми в разі помилки.

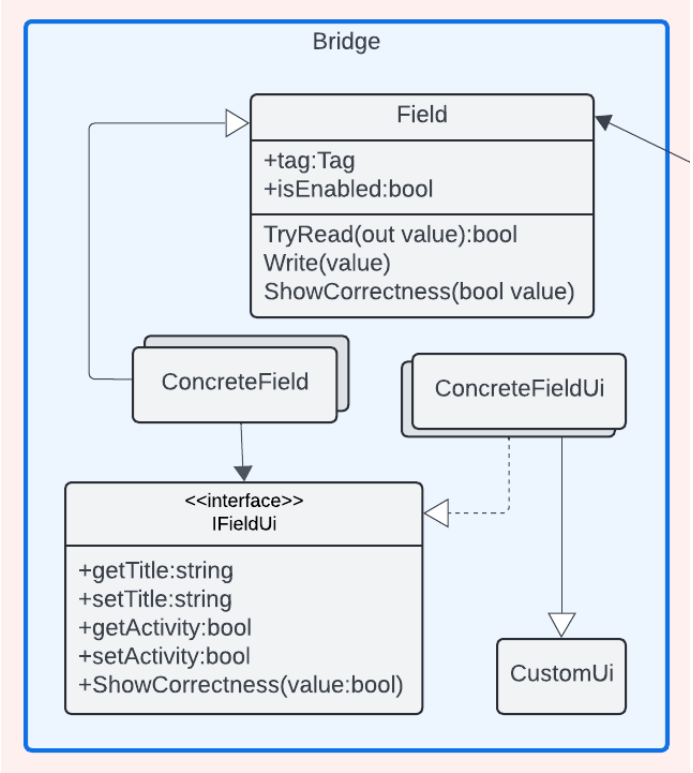
Будь-який **об'єкт, що обробляє форму повинен реалізувати інтерфейс IFormHandler (handler)**. Він володіє методом, що приймає форму в якості параметру. **За надсилання форми до її обробника відповідає клас FormSender**. Цей клас **обробляє помилки, що можуть виникнути при роботі handler`a**. Відповідно він може повторно відкрити форму (із повідомленням щодо контексту помилки) або в разі успіху повернутись user`a.

Таким чином **робота бізнес-логіки**, що імплементує Form Observer **поділяється на** такі етапи:

1. **IFormUser** відкриває **Form** для певного **IFormHandler**
2. Form перевіряє дані введені в **IFormUi**
3. **FormSender** передає **Form** до **IFormHandler**
4. **IFormHandler** виконує дію, маючи дані з форми
5. **FormSender** обробляє помилки, пов'язані з роботою **IFormHandler**

Структура форми

Рис. 3 – UML-діаграма класів, що відповідають за поля діалогових вікон



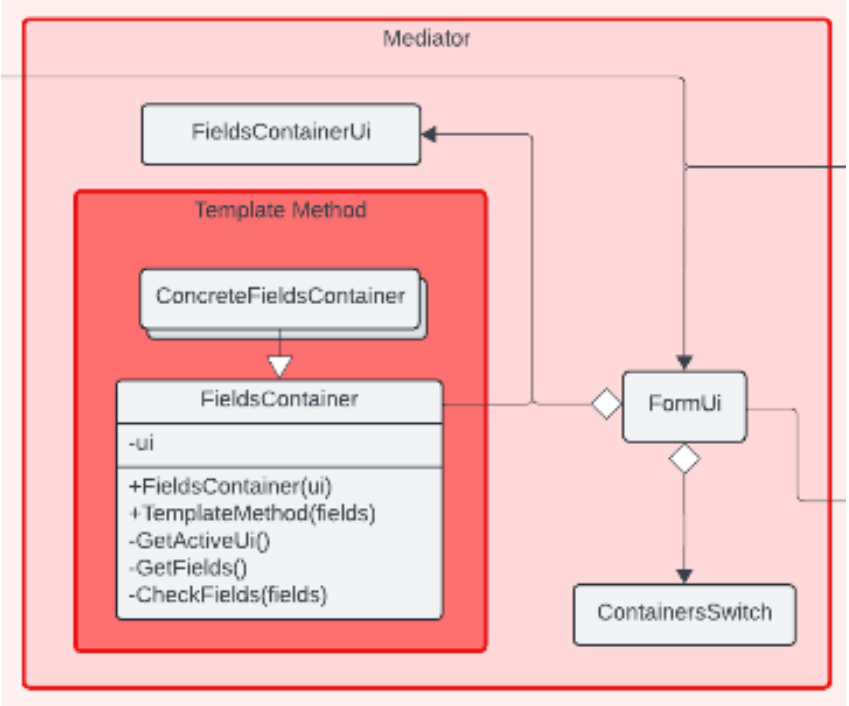
Основою форми (тобто графічного діалогового вікна) **є її поля**. Поля, як об'єкти, **розділені на абстракцію** Field **та реалізацію** IFieldUi, що **відповідає Патерну Міст.**

Такий підхід має 2 переваги. По-перше, він **робить Form незалежним від графічного інтерфейсу**. По-друге **вирішує проблему успадковування** різних за типами полів.

Будь-яке **поле має тип**. І тип це **не просто змінна** – це **певний графічний інтерфейс**, необхідний для введення даних: текстове поле, список, календар для вибору дати тощо. Проблема в тому, що в **більшості бібліотек та фреймворків, що відповідають за графічний інтерфейс неможливо успадковувати користувацькі UI-елементи одне від одного**. Сама по собі це не проблема, однак, якщо зробити ці UI-класи єдиним складовими полів (тобто інтегрувати в них базову бізнес-логіку), то доведеться дублювати значні обсяги коду.

Для уникнення цього **клас бізнес-логіки Field та UI-клас FieldUi розділено**. Таким чином клас **Field отримує дані з FieldUi, а також перевіряє правильність та, за потреби, редагує його заповнення.** Наприклад UI-клас TextField може використовуватися різними класами логіки: StringField, IntField, FloatField… Ці класи будуть по-різному перевіряти правильність заповнення TextField.

Рис. 4 – UML-діаграма класів, що відповідають за реалізацію деактивації окремих полів при їх непотрібності



Не в кожній формі потрібно заповнювати всі поля. Наприклад запит на виготовлені автомобілі може стосуватися як підприємства в цілому так і окремого цеху, чи навіть ділянки. Тож **доцільно поділити діалогове вікно на вкладки, які помістити в загальний контейнер**. В першій будуть обов'язкові поля, а в інших поля, потребу в яких визначає користувач.

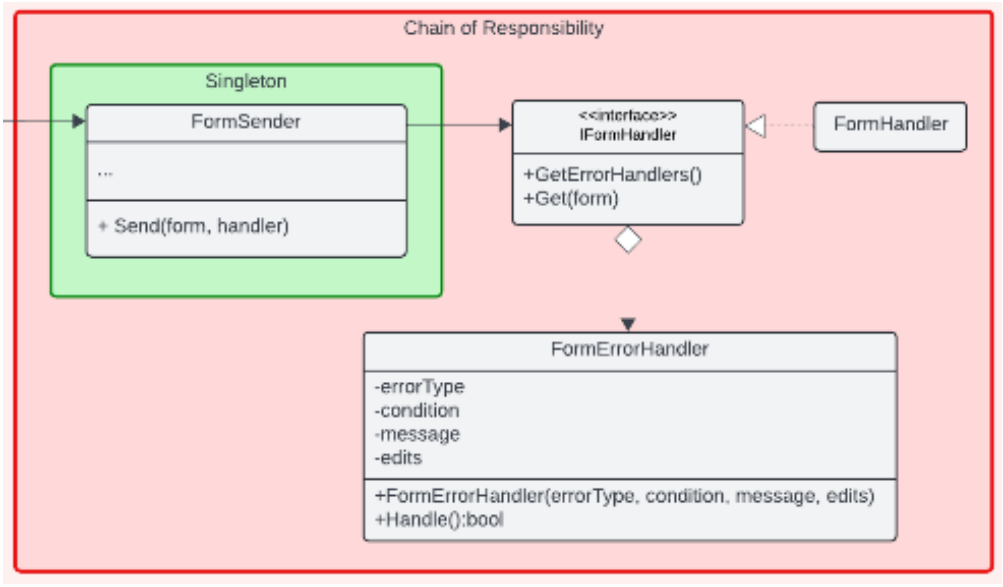
Конкретизуємо на нашому прикладі. **Потрібно створити поле – Switch**, що матиме список на кшталт «підприємство, цех, ділянка». Тобто **це поле визначатиме необхідні поля**. **Залежно від вибраного варіанту вкладки**, які стосуються відповідно цехів, ділянок, **будуть доступні або недоступні** для заповнення. Крім того **форма буде передавати лише дані з полів активних вкладок**.

Реалізувати цей задум **можна на основі патерну Посередник** (яким буде виступати сам об’єкт, що реалізує IFormUi, або просто форма), **щоб уникнути непотрібних залежностей між елементами інтерфейсу**. Форма відстежує подію вибору для поля Switch і надає контейнеру (через CI) інформацію про вибір. А він самостійно реагує на цей вибір, роблячи форму незалежною від конкретного контейнера.

Крім того **container виступає як посередник при передачі даних полів** до IFormUi. Тобто форма отримує лише поля з активних вкладок контейнера. Взагалі важливо розуміти, що **контейнер може не мати вкладок і бути просто обгорткою над всіма полями, однак для форми внутрішня реалізація контейнера не має значення**. Для отримання даних з контейнера викликається так званий Шаблонний метод. Іншими словами **метод отримання полів з контейнера є шаблонним**. Тобто він поділений на окремі частини, які можна довільним чином перевизначити в дочірніх класах. Цими частинами є отримання активних вкладок, отримання з них полів та перевірка цих полів на правильність.

Обробка помилок

Рис. 5 – UML-діаграма класів, що відповідають за обробку помилок, пов’язаних із некоректно введеними користувачем даними



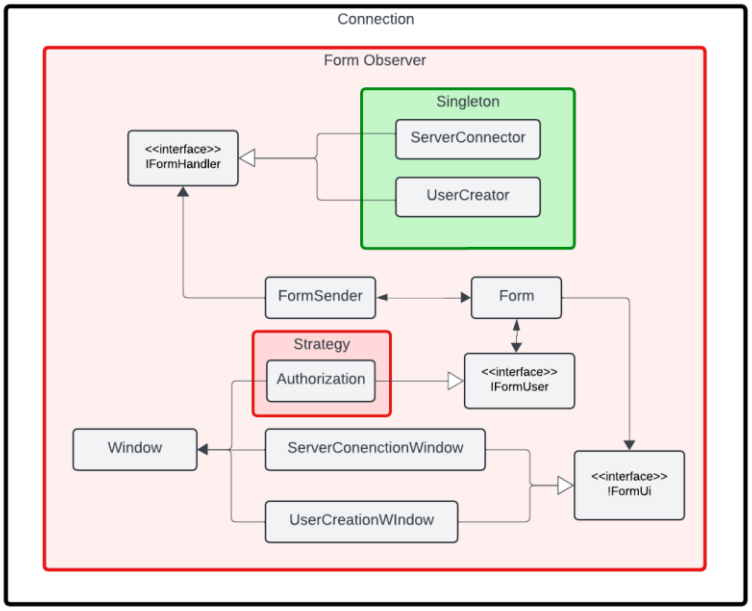
Обробка помилок, викликаних виконанням IFormHandler **здійснюється шляхом імплементації патерну Ланцюг Відповідальності**. Програма містить клас з єдини екземпляром – FormSender, (тобто він реалізує Singleton). Цей клас після надсилання даних handler`у може отримати від нього помилку.

Ця **помилка проходить через ланцюг FormErrorHandler** (надалі errorHandler) – обробників конкретних помилок, передаючи дані про помилку в якості параметру для їх єдиного методу, що **перев іряє відповідність помилки умовам** кожного конкретного errorHandler'а. Примітка: ланцюг errorHandler'ів знаходить в самому IFormHandler'і та отримується FormSender'ом через інтерфейс.

Говорячи точніше **FormErrorHandler отримує дані про помилку від IFormSender'а**. **Якщо помилка не відповідає** його **умовам** (наприклад типу чи коду помилки) **він передає її наступному** errorHandler'у. **Якщо ж помилкавідповідає** поточному обробнику, то **він передає FormSender інформацію** про те, яке повідомлення вивести та які поля вважати неправильно заповненими.

Авторизація

Рис. 6 – UML-діаграма класів, що відповідають за обробку помилок, пов'язаних із некоректно введеними користувачем даними

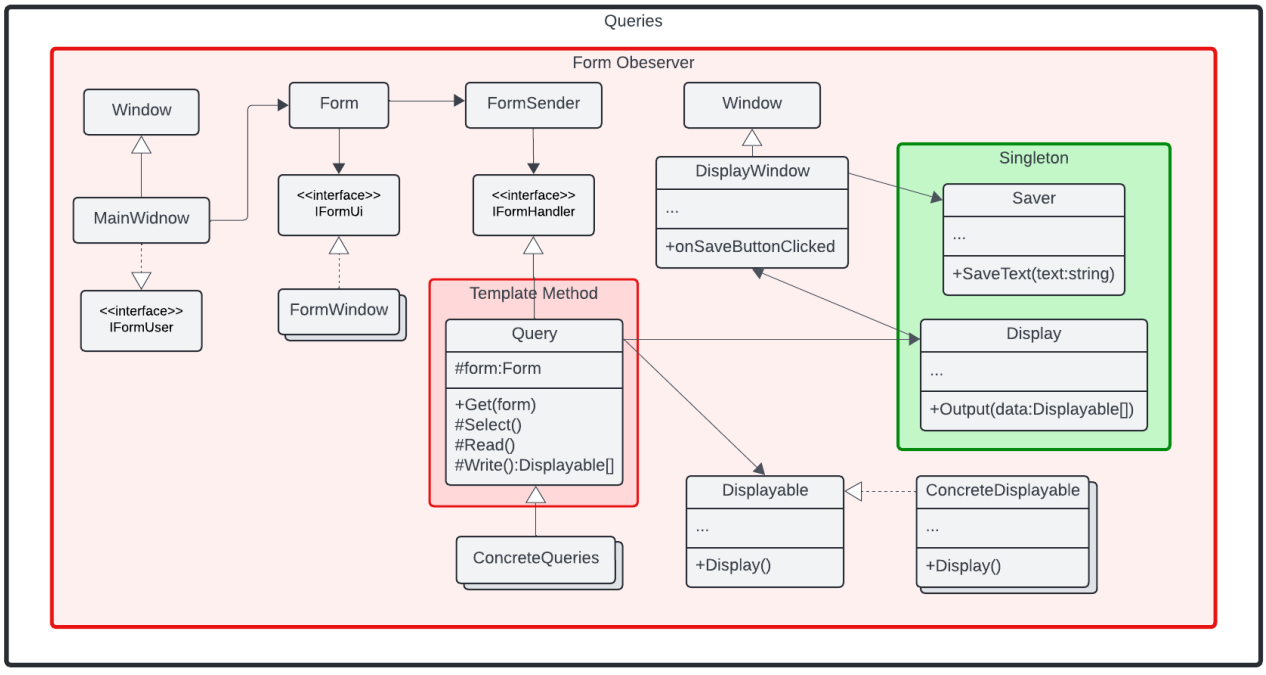


Блок авторизації користувача імплементує Архітектурний Патерн Form Observer. Тож означимо реалізацію основних ролей:

1. IFormUser: Цю роль виконує вікно, що містить 2 кнопки:
    1) для під'єднання до сервера із вибором користувача та бази даних,
    2) для додавання нових користувачів до бази даних
2. IFormUi: Інтерфейс містить 2 діалогові вікна для кожного зі сценаріїв:
    1) Вікно авторизації: host, user, password, database
    2) Вікно додавання користувача: username, password
3. IFormHandler: В якості обробників виступають 2 singleton-класи
    a. ServerConnector відповідає за під'єднання до сервера
    b. UserCreator відповідає за створення користувача бази даних

Здійснення запитів

Рис. 7 – UML-діаграма класів, що відповідають за обробку помилок, пов'язаних із некоректно введеними користувачем даними



Блок здійснення запитів так само реалізує Form Observer. IFormUser – це вікно, що містить кнопки для кожного типу запиту. Відповідно кожен запит має унікальне

діалогове вікно, яке використовується специфічним IFormHandler’ом. Його реалізація доволі складна.

Інтерфейс IFormHandler реалізує клас Reqeust. Цей клас містить єдиний **шаблонний метод**. Він поділяється на 2 складові**: метод, що здійснює SQL запит та метод, що перетворює його в набір екземплярів класів, які потім можна вивести на екран**. Перший метод (пов’язаний з SQL) перевизначається в дочірніх класах суперкласу Request.

Детальніше про класи для виводу. Всі вони реалізують інтерфейс **IDisplayable**, що містить метод для їх перетворення на текст. Колекції цих об’єктів **виводяться класом Display** на інтерфейс користувача. Крім того **Display забезпечує функціонал для збереження** цих IDisplayable-об’єктів у текстовий файл.

Висновок

На цій лабораторній роботі побудовано діаграму класів проєкту із використанням шаблонів проєктування: породжувальних (Синглтон), структурних (Міст), поведінкових (Спостерігач, Посередник, Шаблонний метод, Ланцюг відповідальності). У діаграмі вказано відношення між класами, типи даних для полів та методів. Виділено використання кожного шаблону проєктування.