1. Практична частина
   1. Аналіз та проєктування

2.1.1 Постановка завдання

Структурно підприємство складається з цехів, які в свою чергу поділяються на дільниці. Вироби, що випускаються підприємства: вантажні, легкові автомобілі, автобуси, сільськогосподарські, дорожньо-будівельні машини, мотоцикли та інші вироби. Кожна категорія виробів має специфічні, притаманні лише їй атрибути. Наприклад, для автобусів це місткість, для сільськогосподарських і дорожньо-будівельних машин – продуктивність тощо. По кожній категорії виробів може збиратися кілька видів виробів. Кожній категорії інженерно-технічного персоналу (інженери, технологи, техніки) і робітників (збирачі, токарі, слюсарі, зварювальники тощо) також характерні атрибути, властиві тільки для цієї групи. Робочі об'єднується в бригади, якими керують бригадири. Бригадири обираються з числа робітників; майстри, начальники ділянок і цехів призначаються з числа інженерно-технічного персоналу.

Робітники: інженерно-технічний персонал (інженери, технологи, техніки), робітники (збирачі, токарі, слюсарі, зварювальники…) також характерні атрибути, властиві тільки для цієї групи. Робочі об'єднується в бригади, якими керують бригадири. Бригадири обираються з числа робітників; майстри, начальники ділянок і цехів призначаються з числа інженерно-технічного персоналу. Кожен виріб збирається у своєму цеху (в цеху може збиратися кілька видів виробів) та в процесі виготовлення проходить певний цикл робіт, переміщаючись з однієї ділянки на іншу. Всі роботи по збірці конкретного виробу на певній ділянці виконує одна бригада робітників, при цьому на ділянці може працювати декілька бригад. Очолює роботу на ділянці начальник ділянки, у підпорядкуванні якого знаходиться кілька майстрів. Різні вироби можуть проходити одні й ті самі цикли робіт на одних і тих же ділянках цеху. Зібраний виріб проходить серію випробувань у випробувальних лабораторіях. Випробувальні лабораторії можуть обслуговувати декілька цехів, у свою чергу цехи можуть користуватися кількома лабораторіями. Випробування проводяться фахівцями на обладнанні випробувальної лабораторії, при цьому при випробуванні конкретного виробу в лабораторії можуть бути задіяні різні види обладнання.

Ведеться облік руху кадрів та облік продукції, що випускається.

Види запитів в інформаційній системі:

1. Отримати перелік видів виробів окремої категорії і в цілому, що збираються зазначеним цехом, підприємством.

2. Отримати число і перелік виробів окремої категорії і в цілому, зібраних зазначеним цехом, ділянкою, підприємством в цілому за певний відрізок часу.

3. Отримати дані про кадровий склад цеху, підприємства в цілому і по зазначеним категоріям інженерно-технічного персоналу і робітників.

4. Отримати число і перелік ділянок зазначеного цеху, підприємства в цілому та їх начальників.

5. Отримати перелік робіт, які проходить вказаний виріб.

6. Отримати склад бригад зазначеної ділянки, цеху.

7. Отримати перелік майстрів вказаної ділянки, цеху.

8. Отримати перелік виробів окремої категорії і в цілому, що збираються зараз зазначеним ділянкою, цехом, підприємством.

9. Отримати складу бригад, що беруть участь в складанні зазначеного виробу.

10. Отримати перелік випробувальних лабораторій, що беруть участь у випробуваннях деякого конкретного виробу.

11. Отримати перелік виробів окремої категорії і в цілому, що проходили випробування у зазначеній лабораторії за певний період.

12. Отримати перелік випробувачів, що беруть участь у випробуваннях зазначеного виробу, виробів окремої категорії і в цілому у вказаній лабораторії за певний період.

13. Отримати склад обладнання, що використовувалося при випробуванні зазначеного виробу, виробів окремої категорії і в цілому у вказаній лабораторії за певний період.

14. Отримати число і перелік виробів окремої категорії і в цілому, що збираються зазначеним цехом, ділянкою, підприємством в даний час

2.1.2 Дорожня карта

Рис 2.1.2.1 – Дорожня карта Проєкту. URL – <http://surl.li/uagfz>

Зображення, що містить текст, знімок екрана, Стікер, схема

Автоматично згенерований опис

Зважаючи на загальну складність завдання поточної курсової Роботи та помилки здійснені при виконанні роботи з курсу «Об’єктно орієнтоване програмування». **Було вирішено приділити увагу детальному аналізу завдання** перед його безпосереднім виконанням.

**Мету роботи можна коротко підсумувати наступним чином**. Створити АІС, що надає користувачам можливість робити запити до бази даних, яка містить дані з вказаної предметної області (автомобілебудівного підприємства). Фактично розроблювана програма лише надає зручний (той, що не потребує знання SQL) інтерфейс роботи з певною базою даних

Першим етапом цього аналізу стало **створення дорожньої карти Проєкту**. Робота було поділена на окремі завдання, які було класифіковано за 4-ма категоріями:

1. **Теорія**: містить завдання, що стосуються аналізу предметної області (тобто само постановки завдання та теми патернів проєктування)
2. **Проєктування**: містить завдання, що стосуються проєктування архітектури програми, реалізації в ній шаблонів проєктування
3. **База даних**: містить завдання, що стосуються підготовки бази даних, необхідної для роботи застосунку
4. **Програмування**: містить завдання, безпосередньо пов’язані із програмною реалізацією завдання Роботи

Завдання на карті з’єднані стрілками, які позначають порядок їх виконання. Ж**одна задача не може бути розпочата доки не будуть завершені всі ті, що йдуть перед нею**. В усьому іншому порядок виконання завдань довільний.

**Передумовою всіх практичних завдань є аналіз теорії**. Поверхневого аналізу предметної області (тобто змісту Завдання та даних про роботу автомобілебудівних заводів) достатньо для проєктування структури класів Проєкту, що є основою для створення бази даних. Однак для її заповнення правдоподібними тестовими даними потрібно більш глибоке заглиблення в матеріал.

Аналіз теми курсу, тобто шаблонів проєктування необхідний для проєктування структури класів Проєкту – найважливішої частини роботи. Адже саме **інтеграція патернів у архітектуру проєкту визначає ключову відмінність від роботи з курсу «ООП».**

**Після завершення проєктування структури Проєкту та створення бази даних можна перейти до програмної реалізації Роботи**. Однак попередні підготування до цього етапу, зокрема вибір конкретних технологій та створення на їх основі інтерфейсу користувача можливе вже при поверховому аналізі теорії.

Про всі наведені етапи розробки Проєкту **детальніше розповідається в наступних розділах**.

2.1.3 Структура даних

Рис 2.1.3.1 UML-діаграма структури класів даних Проєкту. URL – <http://surl.li/uagha>

Зображення, що містить текст, схема, План, Креслення

Автоматично згенерований опис

Дані проєкту (тобто все, що стосується інформації про підприємство, а не бізнес-логіки програми як такої) можна поділити на **3 частини**:

1. Структура підприємства
2. Персонал підприємства
3. Вироби підприємства

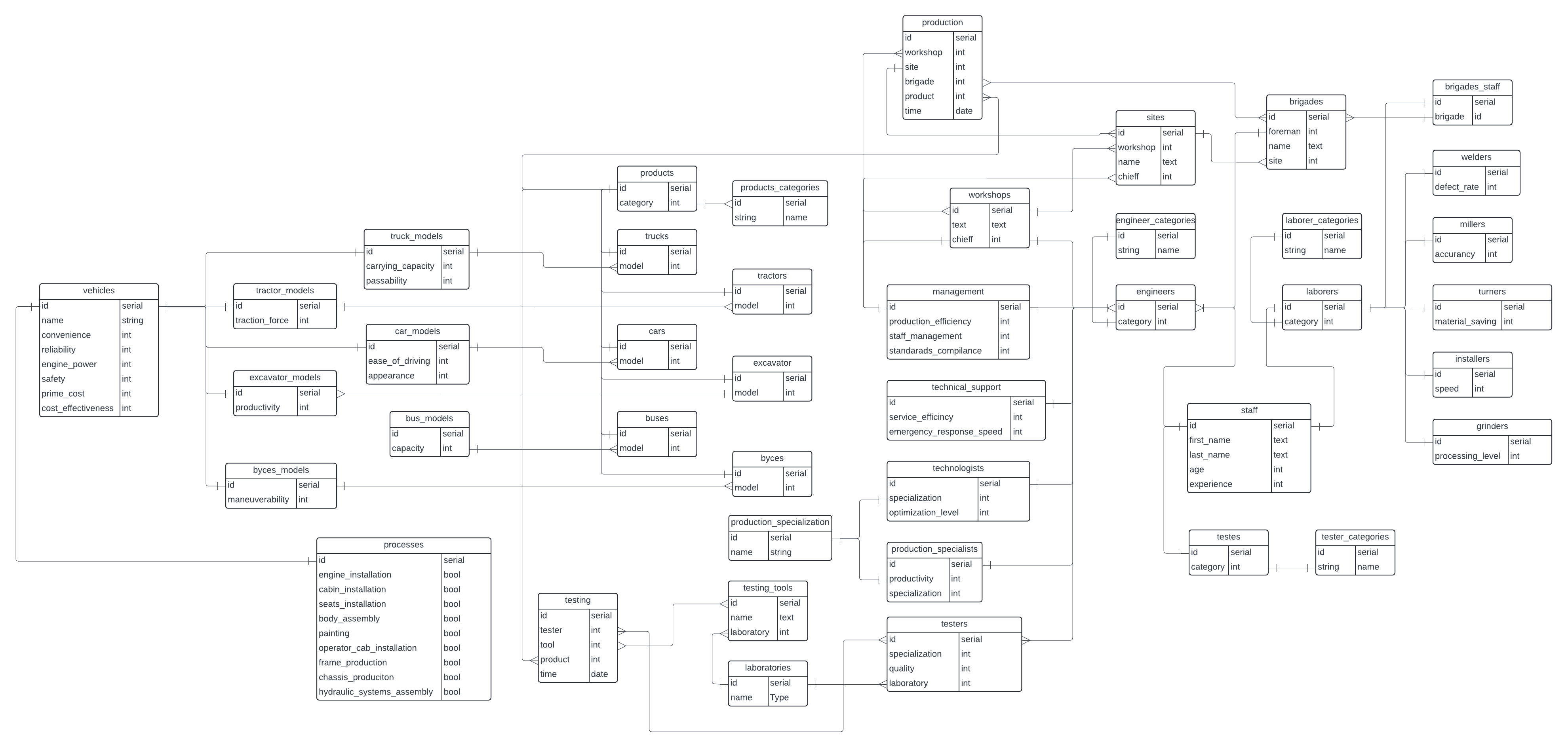
**Структурно** підприємство поділяється на цехи, частинами яких ї ділянки (агрегація), а також дослідницькі лабораторії, пов’язані з цехами (композиція).

**Персонал** підприємства поділяється на робітників та інженерно технічний персонал, тобто інженери та тестувальники. Всі 3 підгрупи мають окремі категорії спеціалістів. На цій основі можна встановити численні зв’язки **успадкування**. Оскільки весь персонал розподілений між цехами, їх ділянками та лабораторія, то між ними наявна також агрегація.

**Вироби**, що виробляє підприємство ділиться на категорії, кожна з яких містить конкретні найменування (черговий ланцюг **успадковування**). Кожен виріб збирається на певних ділянках певного цеху та тестується в певних лабораторіях, а всі ці процеси здійснюють відповідні члени персоналу. Тож можна встановити між ними **композицію**.

2.1.4 База даних

Рис 2.1.4.1 – Діаграма структури бази даних Проєкту. URL – <http://surl.li/uaghe>



**Попередній розділ став основою для проєктування бази даних** – фундаменту, зручним інтерфейсом доступу до якої, виступає розроблювана АІС. Основою БД ПЗ є 2 таблиці: **персонал** (staff) та **виробництво** (production).

Рис 2.1.4.2 – Структура таблиці staff

Зображення, що містить текст, схема, План, Креслення

Автоматично згенерований опис

**Таблиця staff містить дані про всіх працівників підприємтсва**. Від неї успадковуються (тобто зв’язуються відношенням один до одного) всі таблиці, що містить всі конкретні категорії працівників, інженерів, тестувальників. **Дочірні таблиці містять специфічні атрибути** кожної категорії й підкатегорії персоналу:

1. **Інформація про ефективність роботи** різнихвидів **працівників**. Наприклад: швидкість роботи монтажників, ефективність управління менеджерів
2. **Принадлежність особи до** певної **структруної частини підприємства**. Кожен **робітник є членом однієї з бригад** (можливо пригадиром). Кожен **інженер є, або майстром**, що працює на певній дільниці певного цеху, **або начальником** якогось цеху чи його ділянки. Кожен **тестувальник займається своєю діяльністю в** певній **лабораторії**. Ці **принаджлежності реалізуються через зв’язок з відповідними таблицями** (workshops, sites, brigades, laboratories).

Рис 2.1.4.3 – Структура таблиці production

Зображення, що містить текст, схема, План, Креслення

Автоматично згенерований опис

**В таблиці production знаходяться всі вироби підприємства**, дата їх виголовлення та структруні частини підприємства на яких вони вироблялись, Якщо дата є звичайнимполем, то решта з переліченого реалізоване набагато складніше:

1. **Виріб. Тобто тип**, а не конкретний транспортний засіб. Всі ці типи **знаходяться в таблиці products** (з якою таблиця production з’єднана у відношенні багато до одного). **Products містить категорії виробів та посилання** (зв’язок один до одного) **на їх характеристики**, які містяться **в таблиці vehicles**. Ця таблиця **містить загальні** для всіх транспортних засобів **атрибути**, а від **неї успадковуються таблиці зі специфічними** для кожної категорії виробу атрибути.
2. **Структурні частини підприємства**, тобто цехи, ділянки й бригади задіяні у виготовленні вибору **представлені зв’язками багато до одного з відповідними таблицями**. Тобто **кожен виріб у таблиці production містить кількість записів рівну кількості бригад, що брали участь в його виробництві**, оскільки бригади є найменшою структурною одиницею заводу.

Таким чином спроєктовану базу даних можна назвати в значній мірі **нормалізованою**. Вона непогано оптимізована для надсилання до неї запитів, однак заповнення такої бази даними займе значний час.

2.1.5 Архітектура

2.1.5.1 Вступ

Рис 2.1.4.1 – UML Use Case діаграма Проєкту

Зображення, що містить схема, коло, текст, дизайн

Автоматично згенерований опис

Для проєктування архітектури АІС потрібно визначити її **функціонал**. Виділити основні **сценарії застосування**. Єдиним завданням Застосунку є здійснення **запитів до бази даних**. Фактично це можна назвати основним сценарієм. Однак слід виділити й сценарії, що випливають із цього. По-перше потрібно реалізувати **вхід користувача в систему**. Тобто підключення до бази даних. По-друге доцільно реалізувати можливість **збереження результату** будь-якого **запиту** в текстовий файл (фактично цей сценарій розширенням здійснення запиту).

Обидва **сценарії**: авторизація та здійснення **можна звести до** наступного узагальненого **формулювання**:

1. Користувач обирає Дію через певний інтерфейс (надалі Вікно)
2. Система відкриває діалогове вікно, що містить поля, в які користувач вводить дані щодо цієї Дії
3. Користувач заповнює поля
4. Система перевіряє правильність заповнення (якщо воно неправильне, користувач повторює крок 3)
5. Система використовує введені дані, щоб виконати Дію (в разі помилки здійснюється повернення до кроку 2)
6. Система повертає користувача назад до можливості виконання кроку 1

Таким чином **за основу проєкту** було **взято роботу з діалоговими вікнами** (замість нього використано слово «form – форма», оскільки «dialog» є зарезервованим словом у багатьох бібліотеках та фреймворках відповідальних за роботу графічного інтерфейсу). **Їхній виклик, заповнення та подальша обробку.** Цей процес потрібно зробити архітектурним шаблоном (надалі Шаблон, Патерн) Проєкту.

Рис 2.1.5.1.2 – UML-діаграма класів архітектури Проєкту. URL – <http://surl.li/uaisr>Зображення, що містить текст, знімок екрана, схема, План

Автоматично згенерований опис

Перш ніж говорити про реалізацію Патерну слід пояснити 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.

2.1.5.2 Архітектурний (основний) Шаблон

Рис 2.1.5.2 – UML-діаграма класів архітектури Проєкту.

Зображення, що містить текст, знімок екрана, схема, План

Автоматично згенерований опис

Тепер власне Архітектурний Патерн (див. розділ 2.1.5.1). 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’а 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`а**. Відповідно він може повторно відкрити форму (із повідомленням щодо контексту помилки) або в разі успіху повернутись user`а.

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**

2.1.5.2.1 Структура форми

Рис 2.1.5.2.1.1 – UML-діаграма класів, що відповідають за поля діалогових вікон

Зображення, що містить текст, знімок екрана, схема, План

Автоматично згенерований опис

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

**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 2.1.5.2.1.2 – UML-діаграма класів, що відповідають за реалізацію деактивації окремих полів при їх непотрібності

Зображення, що містить текст, знімок екрана, схема, План

Автоматично згенерований опис

**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.

2.1.5.2.2 Обробка помилок

Рис 2.1.5.2.2.1 – UML-діаграма класів, що відповідають за обробку помилок, пов’язаних із некоректно введеними користувачем даними

Зображення, що містить текст, знімок екрана, схема, План

Автоматично згенерований опис

**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’a. 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.

2.1.5.3 Авторизація

Рис 2.1.5.3.1 – UML-діаграма класів, що відповідають за обробку помилок, пов’язаних із некоректно введеними користувачем даними

Зображення, що містить текст, знімок екрана, схема, План

Автоматично згенерований опис

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

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

2.1.5.4 Здійснення запитів

Рис 2.1.5.4.1 – UML-діаграма класів, що відповідають за обробку помилок, пов’язаних із некоректно введеними користувачем даними

Зображення, що містить текст, знімок екрана, схема, План

Автоматично згенерований опис

**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.