ЗМІСТ

ВСТУП

В умовах тісної конкуренції на ринку IT, термін виконання проекту є важливим показником у виборі виконавця. Для того щоб встигнути у терміни, програмісти пишуть код, який легко втілити в короткотерміновій перспективі, замість найкращого рішення. Це призводить до такого поняття, як технічний борг або борг проектування чи борг коду. Якщо технічний борг не погасити вчасно, він може акумулюватися, внаслідок чого з часом стає все важче втілювати зміни. Зміни впроваджуються довше, що погано для двох сторін, тому що проект виконується довше і потребує додаткового фінансування.

Одним із рішень цієї проблеми є рефакторинг коду. Рефакторинг – перетворення програмного коду, зміна внутрішньої структури програмного забезпечення для полегшення розуміння коду і легшого внесення подальших змін без зміни зовнішньої поведінки самої системи.

Існує безліч показників, які повідомляють нас, що потрібно провести рефакторинг. Це дублювання коду, великі підпрограми, велика кількість рівнів вкладеності, клас має багато обов’язків, які слабо пов’язані між собою, інтерфейс класу не забезпечує достатній рівень абстракції, потрібно одночасно змінювати кілька паралельних ієрархій в класі, споріднені дані, які використовуються разом, не організовані в клас, клас не виконує роботу самостійно, а передоручає її іншим класам, назва класу або методу має ім’я, яке недостатньо точно відповідає змісту, клас має занадто багато відкритих членів, нестатичний клас складається тільки з даних або тільки з методів, в ланцюжку викликів методів передається багато зайвих даних, занадто поширене використання глобальних змін.

Для вирішення проблеми існують патерни проектування, які вирішують проблему на початку. Патерн проектування - повторювана архітектурна конструкція, що є вирішенням проблеми проектування, у рамках деякого часто виникаючого контексту.

Абстрактна фабрика — це породжувальний патерн проектування, що дає змогу створювати сімейства пов’язаних об’єктів, не прив’язуючись до конкретних класів створюваних об’єктів.

Використання цього патерну відкидає необхідність у клієнтському коді турбуватись про те як потрібно створювати об’єкти. Розширення ієрархії класів не буде зачіпати клієнтський код, або зачіпатиме мінімально. Тобто, ієрархія класів і клієнтський код розвиватимуться паралельно не зачіпаю одне одного. З таким кодом, програмістам буде комфортно працювати, легко розширювати ієрархію класів, код буде легко читатись. А також, найважливіше, це скорочення терміну розробки, що вигідно всім сторонам.

Отже, дослідження патернів проектування є актуальним в нас час, тому метою курсової роботи є реалізація патерну проектування “Абстрактна фабрика”.

1 ПОСТАНОВКА ПРОБЛЕМИ

1.1 Опис проблеми

Уявімо, що ми пишете симулятор меблевого магазину. Наш код містить:

Сімейство залежних продуктів. Скажімо, Крісло + Диван + Столик.

Кілька варіацій цього сімейства. Наприклад, продукти Крісло, Диван та Столик представлені в трьох різних стилях: Ар-деко, Вікторіанському і Модерн.

Рисунок 1.1 – Сімейства продуктів та їхні варіації

Нам потрібно створювати об’єкти продуктів у такий спосіб, щоб вони завжди підходили до інших продуктів того самого сімейства.

Крім того, ми не хочемо вносити зміни в існуючий код під час додавання в програму нових продуктів або сімейств. Постачальники часто оновлюють свої каталоги, але ми б не хотіли змінювати вже написаний код кожен раз при надходженні нових моделей меблів.

1.2 Аналіз предметної області

Патерн проектування [1] — це типовий спосіб вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм.

На відміну від готових функцій чи бібліотек, патерн не можна просто взяти й скопіювати в програму. Патерн являє собою не якийсь конкретний код, а загальний принцип вирішення певної проблеми, який майже завжди треба підлаштовувати для потреб тієї чи іншої програми.

Патерни часто плутають з алгоритмами, адже обидва поняття описують типові рішення відомих проблем. Але якщо алгоритм — це чіткий набір дій, то патерн — це високорівневий опис рішення, реалізація якого може відрізнятися у двох різних програмах.

Якщо провести аналогії, то алгоритм — це кулінарний рецепт з чіткими кроками, а патерн — інженерне креслення, на якому намальовано рішення без конкретних кроків його отримання.

Описи патернів зазвичай дуже формальні й найчастіше складаються з таких пунктів:

- проблема, яку вирішує патерн;
- мотивація щодо вирішення проблеми способом, який пропонує патерн;
- структура класів, складових рішення;
- приклад однією з мов програмування;
- особливості реалізації в різних контекстах;
- зв’язки з іншими патернами.

Такий формалізм опису дозволив зібрати великий каталог патернів, додатково перевіривши кожен патерн на дієвість.

1.3 Аналіз вимог до програмної системи

Для демонстрації переваг та гнучкості які надає патерн проектування “Абстрактна фабрика” [2] створювана програма повинна відповідати наступним вимогам:

- бізнес-логіка програми повинна працювати з різними видами пов’язаних один з одним продуктів, незалежно від конкретних класів продуктів;
- абстрактна фабрика приховує від клієнтського коду подробиці того, як і які конкретно об’єкти будуть створені;
- клієнтський код повинен працювати з усіма типами створюваних продуктів через загальний інтерфейс;
- користувач має побачити різницю між сімействами об’єктів;
- об’єкти відображатимуться у таблиці;
- програма повинна реалізувати простий графічний інтерфейс;
- програма повинна бути кросплатформеною.

Предметна область – меблі, а саме крісло, кавовий стіл, диван. Програма повинна вміти створювати меблі різних стилів (сучасний стиль, вікторіанський стиль).

Основне призначення програми – демонстрація патерну проектування “Абстрактна фабрика”, тому програма не обов’язково повинна мати практичне застосування у реальному світі.

2 ОБ’ЄКТНО-ОРІЄНТОВАНИЙ АНАЛІЗ ДОСЛІДЖУВАНОЇ ПРОБЛЕМИ

2.1. Вибір моделі розроблення програмної системи

Модель життєвого циклу – це структура, що складається із процесів, робіт та задач, які включають в себе розробку, експлуатацію і супровід програмного продукту; охоплює життя системи від визначення вимог до неї до припинення її використання. Найбільш широко використовувані моделі життєвого циклу (моделі розробки): каскадна, прототипного проектування, еволюційна. Каскадна [3] – модель розробки, в якій кожний етап виконується лише один раз. На кожному етапі робота виконується настільки ретельно, щоб не повертатись до попереднього не виникало. Результат виконання кожного етапу, перед передачею в наступний, піддається верифікації. Вигляд моделі наведено на рис. 2.1.

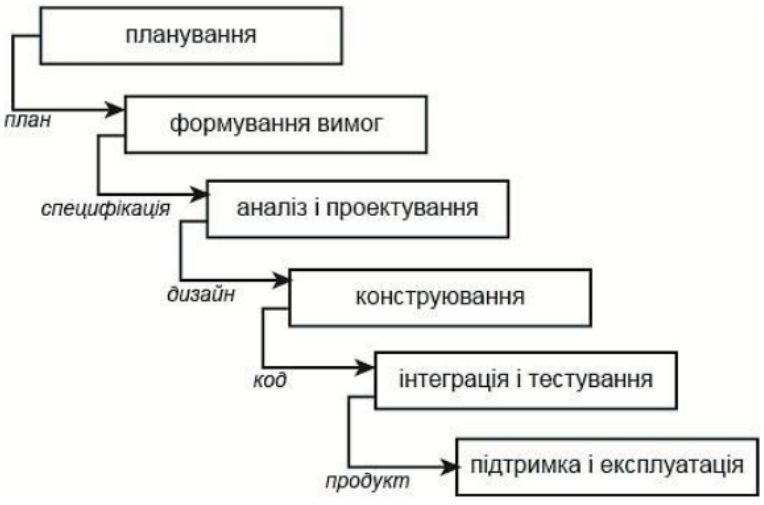

Рисунок 2.1 – Каскадна модель розробки

2.2. Проектування варіантів використання

Програма повинна виконувати такі дії:

- вибір фабрики для породження об’єктів;
- створення дивану;
- створення крісла;
- створення кавового столика.

Актор буде один – користувач.

Діаграма використання [4] наведена на рис. 2.2.

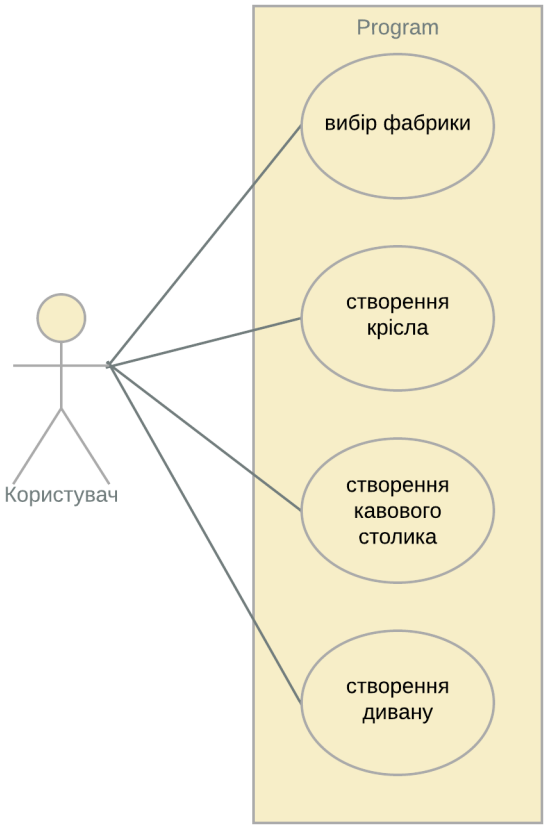

Рисунок 2.2 – Діаграма варіантів використання

Вибір фабрики – користувач повинен мати змогу обрати фабрику. Графічно, це повинно бути реалізовано за допомогою випадаючого списку.

Створення крісла – користувач повинен мати змогу створювати об’єкт типу Крісло. Графічно, для цієї дії повинна бути передбачена кнопка “Створити крісло”.

Створення дивану – користувач повинен мати змогу створювати об’єкт типу Диван. Графічно, для цієї дії повинна бути передбачена кнопка “Створити диван”.

Створення кавового столику – користувач повинен мати змогу створювати об’єкт типу КавовийСтолик. Графічно, для цієї дії повинна бути передбачена кнопка “Створити кавовий столик”.

Звернімо увагу, у діаграмі використання немає таких прецедентів, як: створення об’єктів типу СучаснеКрісло, СучаснийДиван, СучаснийКавовийСтолик, ВікторіанськийДиван, ВікторіанськийКавовийСтолик, ВікторіанськеКрісло, тощо. Ми бачимо одну із переваг використання патерну проектування “Абстрактна фабрика” - відсутність прив’язки до конкретних класів, а взаємодія через абстракцію.

Зменшення кількості варіантів використання, дозволить зменшити графічний інтерфейс користувача. Через це програма стане простішою і її легше буде використовувати.

3 ПРОЦЕС РОЗРОБЛЕННЯ ПРОГРАМИ

3.1 Опис діаграм класів

Для спрощення подальшої розробки програми було побудовано діаграму класів [5] абстрактної фабрики. Програма містить базові класи для меблів: Chair, CoffeeTable, Sofa та базовий клас абстрактної фабрики FurnitureFactory. Від цих класів наслідуються наступні: ModernChair, ModernCoffeeTable, ModernSofa, VictorianChair, VictorianCoffeeTable, VictorianSofa, ModernFurnitureFactory, VictorianFurnitureFactory. Також діаграма містить коментарі, в яких показано можливий варіант використання коду. Діаграма наведена на рис. 3.1.

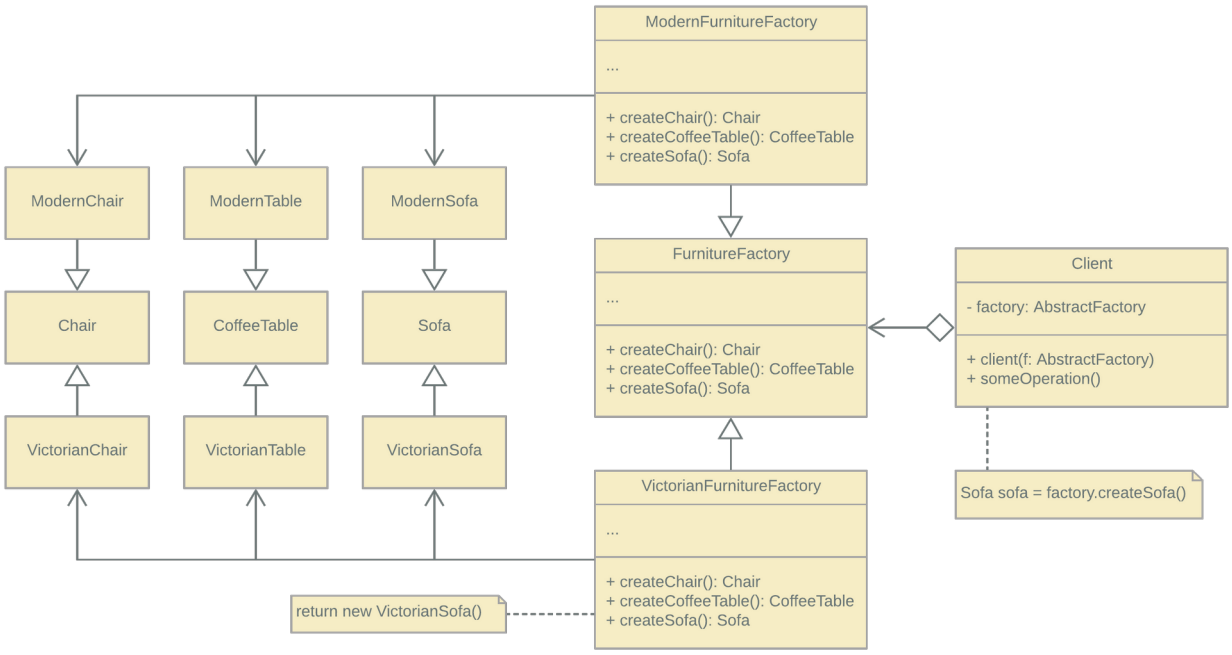

Рисунок 3.1 – Діаграма класів абстрактної фабрики

Діаграма описує основну логіку патерну, проте не описує детально класи. Детальний опис класів наведено на рис. 3.2.

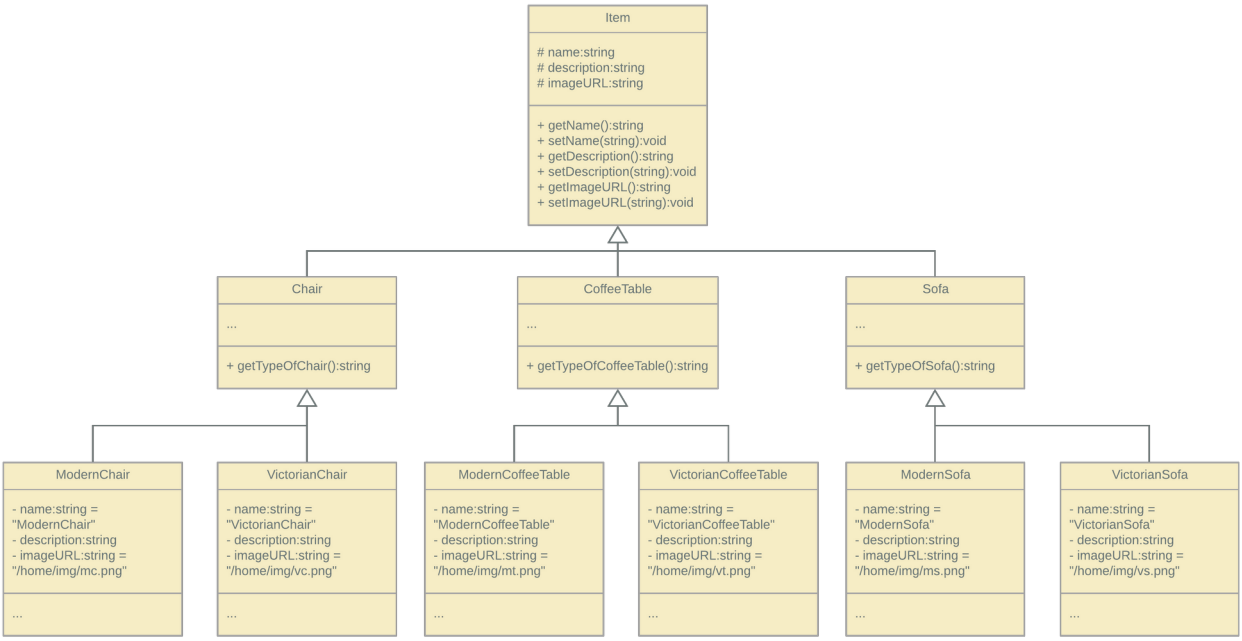

Рисунок 3.2 – Діаграма класів

У програмі можна виділити наступні класи:

- Item (абстрактний клас, використовується для зберігання всіх меблів);
- Chair (абстрактний клас крісло);
- Sofa (абстрактний клас диван);
- CoffeeTable (абстрактний клас кавовий столик);
- ModernChair (клас сучасне крісло);
- ModernSofa (клас сучасний диван);
- ModernCoffeeTable (клас сучасний кавовий столик);
- VictorianChair (клас вікторіанське крісло);
- VictorianSofa (клас вікторіанський диван);
- VictorianCoffeeTable (клас вікторіанський кавовий столик);

- FurnitureFactory (абстрактна фабрика);
- ModernFurnitureFactory (фабрика сучасних меблів);
- VictorianFurnitureFactor (фабрика вікторіанських меблів);
- MainWindow (містить логіку та представлення елементів);
- Main (точка входу програми).

3.2 Опис класів системи

Клас Main – відповідає за точку входу в програму. Ініціалізує створення головного вікна.

Клас MainWindow – клас, який містить основний клієнтський код. Налаштовує компоненту tableView, ініціалізує зміну factory, містить методи опрацювання подій на кліки по кнопках “створити крісло”, “створити кавовий столик”, “створити диван” та випадаючого списку вибору фабрики, відображає компоненти у tableView.

Поля:

- ui – графічний інтерфейс;
- items – список меблів;
- factory – об’єкт типу абстрактна фабрика.

Методи:

- setTable() - відображення меблів в таблиці;

Слоти:

- on_pushButton_clicked();
- on_pushButton_2_clicked();
- on_pushButton_3_clicked();
- on_comboBox_currentIndexChanged().

Клас Item

Поля:

- name – назва предмету;
- description – опис предмету;
- imageURL – посилання на зображення предмету.

Методи:

- getName() - отримання імені;
- setName() - встановлення нового імені;
- getDescription() - отримання опису;
- setDescription() - встановлення опису;
- getImageURL() - отримання посилання на зображення предмету;
- setImageURL() - встановлення посилання на зображення предмету.

Клас Sofa: public Item

Методи:

- getTypeOfSofa() - повертає тип дивану.

Клас CoffeeTable: public Item

Методи:

- getTypeOfCoffeeTable() - повертає тип дивану.

Клас Chai: public Item

Методи:

- getTypeOfChairTable() - повертає тип крісла.

Клас FurnitureFactory: public Item

Методи:

- createCoffeeTable() - поверта об’єкт типу CoffeeTable;
- createChair() - повертає об’єкт типу Chair;
- createSofa() - повертає об’єкт типу Sofa.

Клас ModernFurnitureFactory: public FurnitureFactory

Клас ModernChair: public Chair

Клас ModernCoffeeTable: public CoffeeTable

Клас ModernSofa: public Sofa

Клас VictorianFurnitureFactory: public FurnitureFactory

Клас VictorianChair: public Chair

Клас VictorianCoffeeTable: public CoffeeTable

Клас VictorianSofa: public Sofa

Похідні класи, унаслідують відповідні поля та методи базових класів. Це призвело до зменшення кількості стрічок та покращення читання та модифікації коду.

3.3 Тестування програмного продукту

Для перевірки виконання програми було використано операційну систему Ubuntu 18.04. Тестування передбачає перевірку правильності виконання методів і відображення результату у вікні. Запустивши програму, ми побачимо основне вікно в якому розміщено такі елементи як кнопки створення об’єктів, випадаючий список вибору стилю та таблиця для відображення об’єктів. Початкове вікно наведено на рис. 3.3.

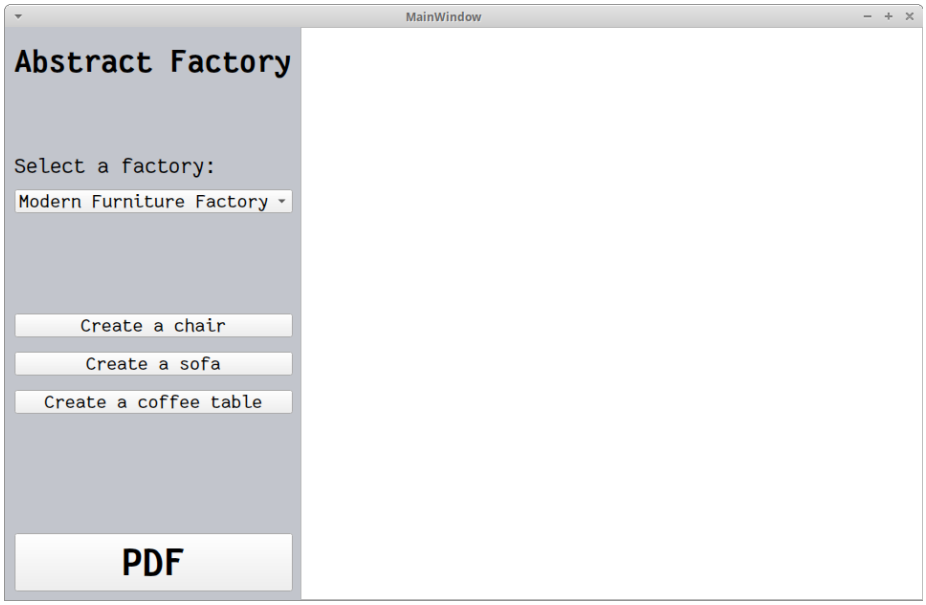

Рисунок 3.3 – Початковий вигляд програми

Після додавання меблів сучасного стилю, наше вікно буде містить зображення відповідних об’єктів. Вигляд вікна наведено на рис. 3.4.

Рисунок 3.4 – Сучасні меблі

Після додавання меблів вікторіанського стилю, наше вікно буде містить зображення відповідних об’єктів. Вигляд вікна наведено на рис. 3.5.

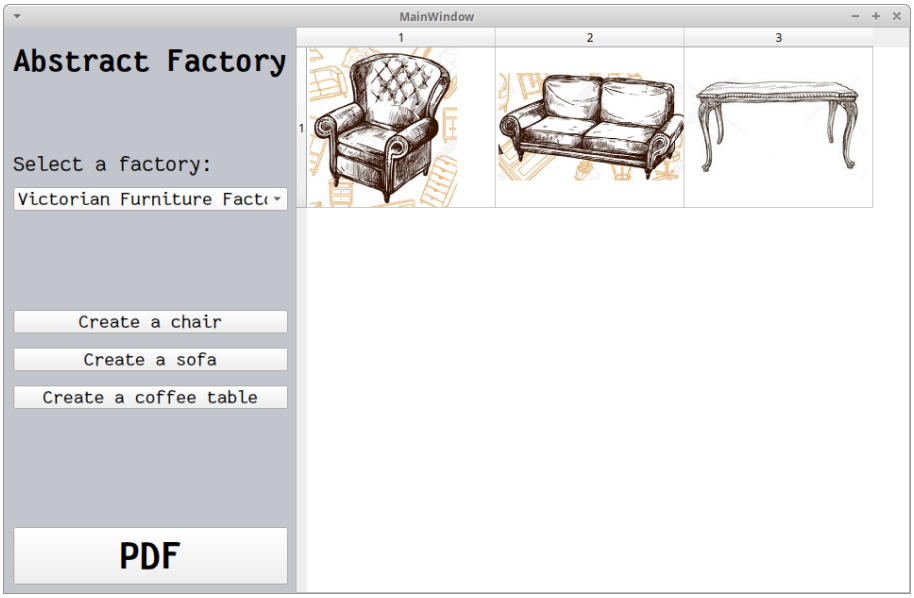

Рисунок 3.5 – Вікторіанські меблі

Після додавання меблів двох стилів, наше вікно буде містить зображення відповідних об’єктів. Вигляд вікна наведено на рис. 3.6.

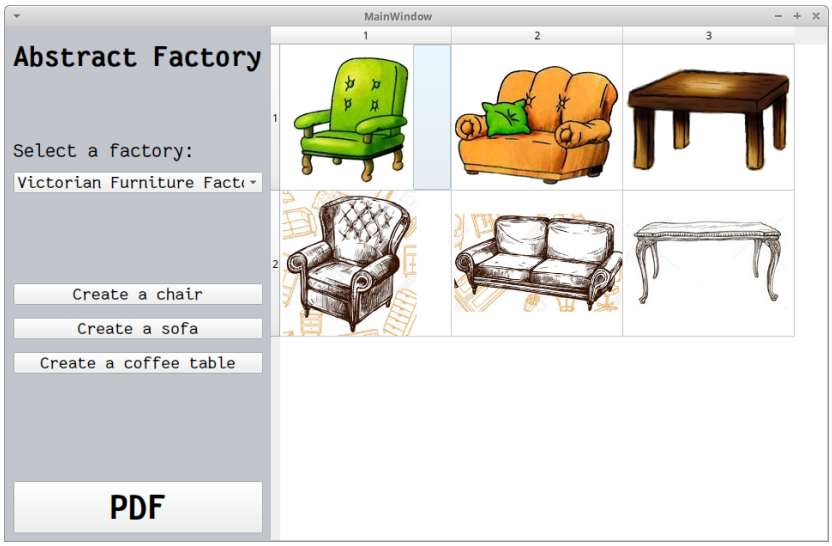

Рисунок 3.6 – Меблі різних типів

ВИСНОВКИ

Під час виконання курсової роботи було досліджено патерн проектування “Абстрактна фабрика”.

Ознайомлено, в який ситуаціях краще його застосовувати.

Було розроблено програму для демонстрації роботи цього патерну.

Програма розроблялась за допомогою Qt Creator [6-8] та мови програмування C++ [9], в операційній системі Ubuntu 18.04.

Діаграми класів та варіантів використання проектувались за допомогою інтернет ресурсу lucidchart.com.

Написана програма виконує усі потрібні функції. Під час тестування програми, неполадок не виявлено. Технічна демонстрація застосунку працює добре.

Поставлені завдання були виконані, а саме:

- реалізовано простий графічний інтерфейс;
- програма працює із сімейством класів;
- клієнтський код мінімально залежний від ієрархії класі, та працює максимально абстрактно.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Швець О. Занурення в патерни проектування / Олександр Швець. – Київ: Буква і Цифра, 2017. – 393 с. – (3).
2. AbstractFactory [Електроний ресурс] / Режим доступу: https://refactoring.guru/design-patterns/abstract-factory
3. Життєвий цикл програмного забезпечення [Електроний ресурс] / Режим доступу: https://uk.wikipedia.org/wiki/Життєвий_цикл_програми
4. UseCase [Електроний ресурс] / Режим доступу: https://www.bridging-the-gap.com/what-is-a-use-case/
5. ClassDiagram [Електроний ресурс] / Режим доступу: https://www.smartdraw.com/class-diagram/
6. Qt Documentation [Електроний ресурс] / Режим доступу: https://doc.qt.io – Назва зі сторінки Інтернету
7. QtableView Class [Електроний ресурс] / Режим доступу: https://doc.qt.io/qt-5/qtableview.html
8. М. Шлее Qt 5.3 Професійне програмувння C++ / Санкт-Петербург «БХВ-Петербург» 2015. – 233 с.
9. Б. Строуструп Мови програмування C++ / Addison-Wesley Pub Co 2000. - 1009 с.
10. Петрик М. Методичні вказівки до виконання курсових робіт з дисципліни «ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ» (CS-102) / М. Петрик, О. Петрик. – Тернопіль: ТНТУ, 2015. – 36 с.

ДОДАТКИ

Додаток А – Код програми

mainwindow.cpp

```
#include "ModernFurnitureFactory.cpp"
#include "VictorianFurnitureFactory.cpp"
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include <QPixmap>
#include <QPixmap>
#include <QStandardItemModel>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    ui->tableView->verticalHeader()->setDefaultSectionSize(170);
    ui->tableView->horizontalHeader()->setDefaultSectionSize(200);

    this->factory = new ModernFurnitureFactory();
}

void MainWindow::setTable()
{
    int i = 0;
    QStandardItemModel *model = new QStandardItemModel();
    for (auto el = this->items.begin(); el != this->items.end(); ++el)
    {
        QString imageURL = QString::fromStdString(el->getImageURL());
        QImage image(imageURL);
        QStandardItem *item = new QStandardItem();
        item->setData(QVariant(QPixmap::fromImage(image)), Qt::DecorationRole);
        model->setItem(i / 3, i % 3, item);
        i++;
    }
    ui->tableView->setModel(model);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

```
void MainWindow::on_pushButton_clicked()
{
    Chair *chair = this->factory->createChair();
    this->items.push_back(*chair);
    setTable();
}

void MainWindow::on_pushButton_2_clicked()
{
    Sofa *sofa = this->factory->createSofa();
    this->items.push_back(*sofa);
    setTable();
}

void MainWindow::on_pushButton_3_clicked()
{
    CoffeeTable *coffeeTable = this->factory->createCoffeeTable();
    this->items.push_back(*coffeeTable);
    setTable();
}

void MainWindow::on_comboBox_currentIndexChanged(int index)
{
    if (index == 0) {
        this->factory = new ModernFurnitureFactory();
    } else if (index == 1) {
        this->factory = new VictorianFurnitureFactory();
    }
}
```

main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

VictorianFurnitureFactory.cpp

```
#include "FurnitureFactory.h"
#include "VictorianChair.cpp"
#include "VictorianCoffeeTable.cpp"
#include "VictorianSofa.cpp"
```

```
class VictorianFurnitureFactory: public FurnitureFactory
{
public:
  Chair* createChair()
  {
    return new VictorianChair();
  }

  CoffeeTable* createCoffeeTable()
  {
    return new VictorianCoffeeTable();
  }

  Sofa* createSofa()
  {
    return new VictorianSofa();
  }
};
```

VictorianSofa.cpp

```
#include <string>
#include "Sofa.h"

using namespace std;

class ModernSofa: public Sofa
{
public:
  ModernSofa()
  {
    name = "Modern Sofa";
    description = "Modern Sofa, 2019";
    imageURL = "/home/gaba/Desktop/coursework2019/images/ms.png";
  }

  string getTypeOfSofa()
  {
    return "ModernSofa";
  }
};
```

VictorianChair.cpp

```
#include <string>
#include "Chair.h"

using namespace std;
```

```
class VictorianChair: public Chair
{
public:
  VictorianChair()
  {
    name = "Victorian Chair";
    description = "Victorian Chair, 1890 ";
    imageURL = "/home/gaba/Desktop/coursework2019/images/vc.png";
  }

  string getTypeOfChair()
  {
    return "VictorianChair";
  }
};
```

VictorianCoffeeTable.cpp

```
#include <string>
#include "CoffeeTable.h"

using namespace std;

class VictorianCoffeeTable: public CoffeeTable
{
public:
  VictorianCoffeeTable()
  {
    name = "Victorian Coffee Table";
    description = "Victorian Coffee Table, 1890";
    imageURL = "/home/gaba/Desktop/coursework2019/images/vct.png";
  }

  string getTypeOfCoffeeTable()
  {
    return "VictorianCoffeeTable";
  }
};
```

ModernFurnitureFactory.cpp

```
#include "FurnitureFactory.h"
#include "ModernChair.cpp"
#include "ModernCoffeeTable.cpp"
#include "ModernSofa.cpp"

class ModernFurnitureFactory: public FurnitureFactory
{
public:
```

```
  Chair* createChair()
  {
    return new ModernChair();
  }

  CoffeeTable* createCoffeeTable()
  {
    return new ModernCoffeeTable();
  }

  Sofa* createSofa()
  {
    return new ModernSofa();
  }
};
```

ModernChair.cpp

```
#include <string>
#include "Chair.h"

using namespace std;

class ModernChair: public Chair
{
public:
  ModernChair()
  {
    name = "Modern Chair";
    description = "Modern Chair, 2019";
    imageURL = "/home/gaba/Desktop/coursework2019/images/mc.png";
  }

  string getTypeOfChair()
  {
    return "ModernChair";
  }
};
```

ModernCoffeeTable.cpp

```
#include <string>
#include "CoffeeTable.h"

using namespace std;

class ModernCoffeeTable: public CoffeeTable
{
public:
```

```
  ModernCoffeeTable()
  {
    name = "Modern Coffee Table";
    description = "Modern Coffee Table, 2019";
    imageURL = "/home/gaba/Desktop/coursework2019/images/mct.png";
  }

  string getTypeOfCoffeeTable()
  {
    return "ModernCoffeeTable";
  }
};
```

ModernSofa.cpp

```
#include <string>
#include "Sofa.h"

using namespace std;

class ModernSofa: public Sofa
{
public:
  ModernSofa()
  {
    name = "Modern Sofa";
    description = "Modern Sofa, 2019";
    imageURL = "/home/gaba/Desktop/coursework2019/images/ms.png";
  }

  string getTypeOfSofa()
  {
    return "ModernSofa";
  }
};
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "FurnitureFactory.h"
#include "Item.h"

#include <vector>

#include <QMainWindow>
```

```
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    void setTable();

private slots:
    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

    void on_pushButton_3_clicked();

    void on_comboBox_currentIndexChanged(int index);

private:
    Ui::MainWindow *ui;
    vector<Item> items;
    FurnitureFactory *factory;
};
#endif // MAINWINDOW_H
```

Item.h

```
#pragma once
#include <iostream>

using namespace std;

class Item
{
protected:
  string name;
  string description;
  string imageURL;
public:
  virtual string getName()
  {
    return name;
  }

  virtual void setName(string _name)
  {
    name = _name;
```

```
  }

  virtual string getDescription()
  {
    return description;
  }

  virtual void setDesctiption(string _description)
  {
    description = _description;
  }

  virtual string getImageURL()
  {
    return imageURL;
  }

  virtual void setImageURL(string _imageURL)
  {
    imageURL = _imageURL;
  }
};
```

FurnitureFactory.h

```
#pragma once

#include "Chair.h"
#include "CoffeeTable.h"
#include "Sofa.h"

class FurnitureFactory
{
public:
  virtual Chair* createChair() = 0;
  virtual CoffeeTable* createCoffeeTable() = 0;
  virtual Sofa* createSofa() = 0;
};
```

Chair.h

```
#pragma once

#include <iostream>
#include "Item.h"

using namespace std;

class Chair: public Item
{
public:
```

```
  virtual string getTypeOfChair() = 0;
};
```

Sofa.h

```
#pragma once
#include <iostream>
#include "Item.h"

using namespace std;

class Sofa: public Item
{
public:
  virtual string getTypeOfSofa() = 0;
};
```

CoffeeTable.h

```
#pragma once
#include <iostream>
#include "Item.h"

using namespace std;

class CoffeeTable: public Item
{
public:
  virtual string getTypeOfCoffeeTable() = 0;
};
```

Додаток Б – Диск з електроним варіантом звіту