Міністерство освіти і науки України
Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №6
З дисципліни «Технології розроблення програмного забезпечення»
Тема: **«ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»»**
Flexible Automatical Tool

Виконав:
Студент групи ІА-22
Сидорін Д.О.

Перевірив:
Мягкий М. Ю.

Київ-2024

Зміст

Тема:
ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»
Мета:
Ознайомитися з основними шаблонами проєктування, такими як «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator», дослідити їхні принципи роботи та навчитися використовувати їх для створення гнучкого та масштабованого програмного забезпечення.

Завдання:

Розробка інструмента для автоматизації простих дій користувача в вигляді візуального додатку, який дозволяє організовувати та автоматизувати рутинні процеси. Додаток включає можливість створення "карт пам'яті" з функціоналом для роботи з кількома картами в різних вкладках. Передбачено автоматичне з'єднання елементів карт за допомогою ліній, а також можливість додавання різноманітних медіафайлів (вкладених файлів, зображень, відео з попереднім переглядом). Користувач може також додавати значки для категорій та терміновості завдань, а також окреслювати важливі області карти за допомогою пунктирних ліній.

Автоматизація процесів дозволяє інтегрувати правила для регулярних дій (наприклад, автоматичне оновлення карт чи додавання нових елементів за певним розкладом) через планувальник завдань або за допомогою макросів, що забезпечують зручність і ефективність роботи з додатком.

Хід роботи

1. Реалізувати не менше 3-х класів відповідно до обраної теми

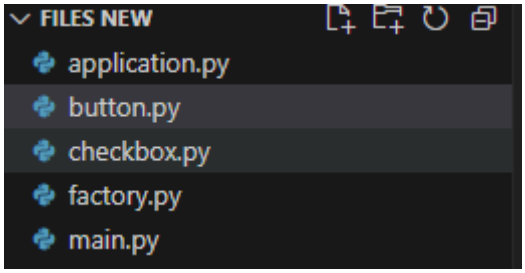

Рис. 1 — Структура проекту

У ході роботи було розроблено та модифіковано наступні класи:

Button (Абстрактний клас)
Опис:

- Абстрактний клас, який визначає загальний інтерфейс для всіх кнопок.
- Містить абстрактне визначення методу для відображення кнопки.

---

2. WinButton (Конкретний клас)

Опис:

- Клас, який реалізує кнопку для операційної системи Windows.
- Наслідується від абстрактного класу Button.

---

3. MacButton (Конкретний клас)

Опис:

- Клас, який реалізує кнопку для операційної системи macOS.
- Наслідується від абстрактного класу Button.

---

4. Checkbox (Абстрактний клас)

Опис:

- Абстрактний клас, який визначає загальний інтерфейс для всіх чекбоксів.
- Містить абстрактне визначення методу для відображення чекбокса.

---

5. WinCheckbox (Конкретний клас)

Опис:

- Клас, який реалізує чекбокс для операційної системи Windows.
- Наслідується від абстрактного класу Checkbox.

---

6. MacCheckbox (Конкретний клас)

Опис:

- Клас, який реалізує чекбокс для операційної системи macOS.
- Наслідується від абстрактного класу Checkbox.

---

7. GUIFactory (Абстрактна фабрика)

Опис:

- Абстрактний клас фабрики, який визначає методи для створення кнопок та чекбоксів.
- Кожна конкретна фабрика повинна реалізувати ці методи для створення своїх компонентів інтерфейсу.

---

8. WinFactory (Конкретна фабрика)

Опис:

- Клас, який реалізує фабрику для створення компонентів інтерфейсу для операційної системи Windows.
- Наслідується від абстрактного класу GUIFactory.

---

9. MacFactory (Конкретна фабрика)

Опис:

- Клас, який реалізує фабрику для створення компонентів інтерфейсу для операційної системи macOS.
- Наслідується від абстрактного класу GUIFactory.

---

10. Application (Клас застосунку)

Опис:

- Клас, який використовує конкретну фабрику для створення компонентів інтерфейсу (кнопок та чекбоксів).
- Залежить від абстрактної фабрики, щоб не залежати від конкретної реалізації компонентів.

---

11. main() (Основна функція)

Опис:

- Точка входу програми, де користувач вибирає операційну систему, а потім створюється відповідна фабрика для створення UI-компонентів.
- Створюється об’єкт застосунку, який використовує обрану фабрику для створення компонентів інтерфейсу та їх відображення.

2. Реалізувати один з розглянутих шаблонів за обраною темою

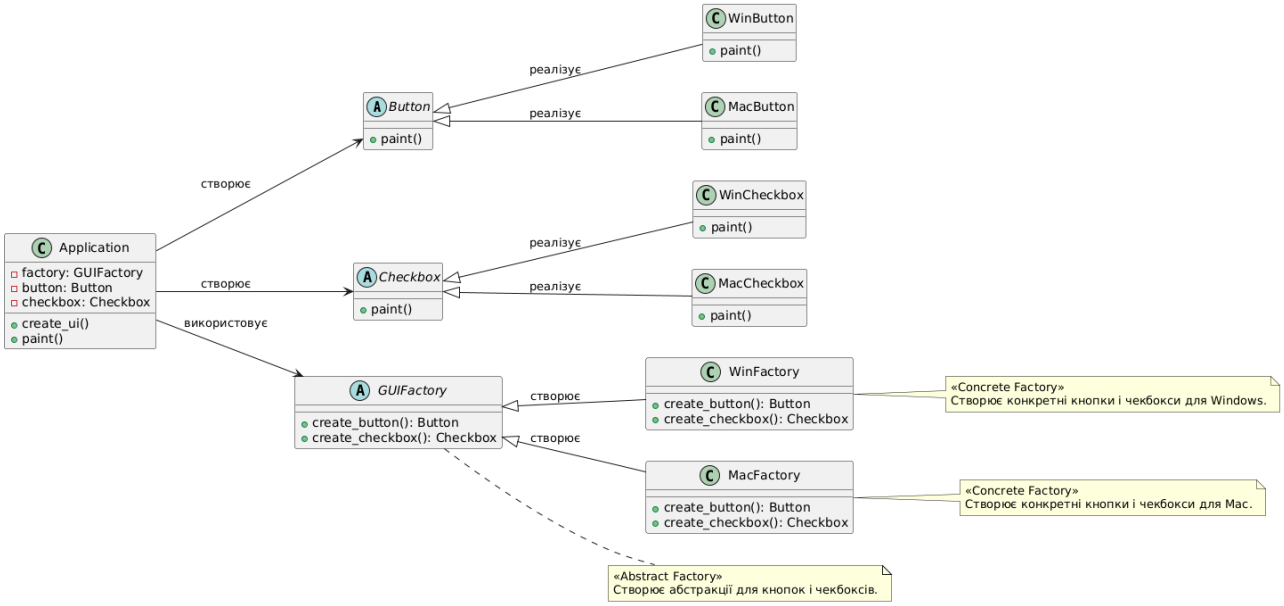

Рис. 2 — Діаграма класів

Опис діаграми класів:

Діаграма класів описує структуру патерну **Abstract Factory**, що включає взаємодії між абстрактними класами та конкретними фабриками і продуктами.

1. **Button** (абстрактний клас):
   - Це абстрактний клас для кнопок. Він має абстрактний метод paint(), який реалізує конкретні класи кнопок. Цей клас визначає загальний інтерфейс для всіх типів кнопок, забезпечуючи уніфікацію для створення елементів інтерфейсу.
2. **Checkbox** (абстрактний клас):
   - Аналогічно класу **Button**, цей клас визначає абстрактний інтерфейс для чекбоксів. Він також має абстрактний метод paint(), який реалізують конкретні класи чекбоксів.
3. **GUIFactory** (абстрактний клас):
   - Це абстрактна фабрика, яка визначає два методи: create_button() та create_checkbox(). Ці методи створюють відповідні елементи інтерфейсу: кнопку та чекбокс. Клас **GUIFactory** забезпечує абстракцію для фабрик, що створюють компоненти інтерфейсу.
4. **WinButton** (конкретна кнопка для Windows):
   - Це конкретна реалізація кнопки для операційної системи Windows. Клас реалізує метод **paint()**, який відображає кнопку в стилі Windows.
5. **MacButton** (конкретна кнопка для macOS):
   - Це конкретна реалізація кнопки для операційної системи macOS. Вона також реалізує метод **paint()**, але з особливостями стилю macOS.

6. **WinCheckbox** (конкретний чекбокс для Windows):
    - Це конкретний чекбокс для Windows, що реалізує метод **paint()** для відображення елемента в стилі Windows.
7. **MacCheckbox** (конкретний чекбокс для macOS):
    - Це конкретний чекбокс для macOS, що реалізує метод **paint()** для відображення елемента в стилі macOS.
8. **WinFactory** (конкретна фабрика для Windows):
    - Це конкретна фабрика, яка реалізує методи **create_button()** та **create_checkbox()** для створення компонентів інтерфейсу, відповідних стилю Windows. Вона створює об'єкти типу **WinButton** та **WinCheckbox**.
9. **MacFactory** (конкретна фабрика для macOS):
    - Це конкретна фабрика, яка реалізує методи **create_button()** та **create_checkbox()** для створення компонентів інтерфейсу, відповідних стилю macOS. Вона створює об'єкти типу **MacButton** та **MacCheckbox**.
10. **Application**:
    - Клас **Application** використовує конкретні фабрики для створення елементів інтерфейсу. Він залежить від фабрики **GUIFactory** для створення відповідних компонентів: кнопки та чекбокса. Клас містить методи **create_ui()** (для ініціалізації інтерфейсу) і **paint()** (для відображення інтерфейсу).

Зв'язки між класами:

- **Button <|-- WinButton** та **Button <|-- MacButton**: Класи **WinButton** і **MacButton** є конкретними реалізаціями абстрактного класу **Button** для відповідних операційних систем.
- **Checkbox <|-- WinCheckbox** та **Checkbox <|-- MacCheckbox**: Класи **WinCheckbox** і **MacCheckbox** є конкретними реалізаціями абстрактного класу **Checkbox** для Windows та macOS відповідно.
- **GUIFactory <|-- WinFactory** та **GUIFactory <|-- MacFactory**: Класи **WinFactory** і **MacFactory** реалізують абстрактний клас **GUIFactory**, надаючи реалізації методів для створення кнопок і чекбоксів, специфічних для Windows і macOS.
- **Application --> GUIFactory**: Клас **Application** взаємодіє з фабрикою **GUIFactory** для створення елементів інтерфейсу, зокрема кнопок та чекбоксів.
- **Application --> Button** та **Application --> Checkbox**: Клас **Application** використовує компоненти інтерфейсу, створені фабрикою, для відображення інтерфейсу.

Перевірка патерну

```python
import unittest
from factory import WinFactory, MacFactory
from button import WinButton, MacButton
from checkbox import WinCheckbox, MacCheckbox
from application import Application


class TestAbstractFactory(unittest.TestCase):

    def test_win_factory(self):
        factory = WinFactory()
        app = Application(factory)
        app.create_ui()

        # Перевіряємо, чи створена кнопка є WinButton
        self.assertIsInstance(app.button, WinButton)
        self.assertEqual(app.button.paint(), "Windows Button Painted")

        # Перевіряємо, чи створений чекбокс є WinCheckbox
        self.assertIsInstance(app.checkbox, WinCheckbox)
        self.assertEqual(app.checkbox.paint(), "Windows Checkbox Painted")

    def test_mac_factory(self):
        factory = MacFactory()
        app = Application(factory)
        app.create_ui()

        # Перевіряємо, чи створена кнопка є MacButton
        self.assertIsInstance(app.button, MacButton)
        self.assertEqual(app.button.paint(), "Mac Button Painted")

        # Перевіряємо, чи створений чекбокс є MacCheckbox
        self.assertIsInstance(app.checkbox, MacCheckbox)
        self.assertEqual(app.checkbox.paint(), "Mac Checkbox Painted")
```

```python
    def test_factory_integration(self):
        # Тестуємо обидві фабрики разом для порівняння
        win_factory = WinFactory()
        mac_factory = MacFactory()

        # Створюємо застосунок для Windows
        app_win = Application(win_factory)
        app_win.create_ui()
        self.assertIsInstance(app_win.button, WinButton)
        self.assertEqual(app_win.button.paint(), "Windows Button Painted")
        self.assertIsInstance(app_win.checkbox, WinCheckbox)
        self.assertEqual(app_win.checkbox.paint(), "Windows Checkbox Painted")

        # Створюємо застосунок для Mac
        app_mac = Application(mac_factory)
        app_mac.create_ui()
        self.assertIsInstance(app_mac.button, MacButton)
        self.assertEqual(app_mac.button.paint(), "Mac Button Painted")
        self.assertIsInstance(app_mac.checkbox, MacCheckbox)
        self.assertEqual(app_mac.checkbox.paint(), "Mac Checkbox Painted")


if __name__ == "__main__":
    unittest.main()
```

Рис. 3 — Перевірка роботи

Мета тестування:

- Перевірити правильність створення компонентів для кожної конкретної фабрики.
- Перевірити, чи компоненти (кнопки та чекбокси) належать до правильних класів.
- Перевірити, чи метод **paint()** правильно відображає повідомлення для кожного компонента.

Як це працює:

- Тести перевіряють, чи фабрики створюють правильні об'єкти, та чи кожен об'єкт працює за своїм призначенням (відображає відповідні повідомлення через метод **paint()**).
- Використовуючи **unittest**, усі тести виконуються автоматично, і результат тестування відображається у терміналі.

Рис. 4 — Результат роботи

Пояснення виводу:

- **Ran 3 tests** — це повідомлення про кількість тестів, які були виконані. У цьому випадку, тестується 3 методи (по одному для кожної фабрики та один інтеграційний тест).
- **in 0.000s** — це час, який знадобився для виконання тестів (може змінюватись залежно від швидкості твоєї системи).
- **OK** — це показник того, що всі тести пройшли успішно, і патерн **Abstract Factory** працює коректно, без помилок.

Висновки:

В цій лабораторній роботі був разроблений шаблон паттерну Abstract Factory, можно побачити діаграмму классів, характеристики паттерну, та перевірку ,можна зробити висновок що використання патерну Abstract Factory дозволяє забезпечити розширюваність, модульність і підтримуваність проєкту.

Код: https://github.com/Lepseich/trpz/tree/main/lab6/files