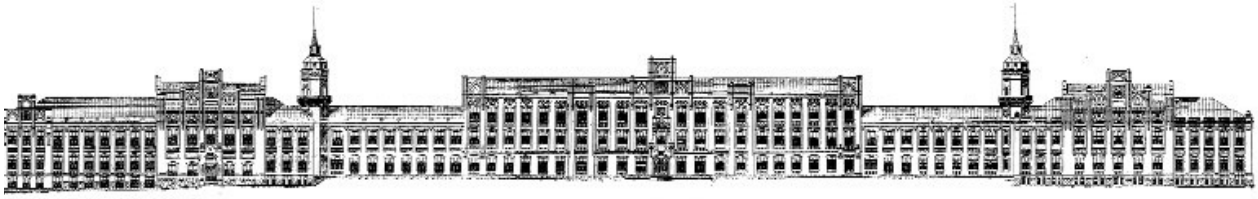

Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №6

з дисципліни «Технології розроблення

програмного забезпечення»

Тема: «Патерни проектування»

Виконала
студентка групи ІА-32:
Павлюк В. О.

Зміст

Table of Contents

Вступ

Темою роботи є «Вступ до патернів проєктування». Мета цієї лабораторної це вивчити структуру шаблонів«Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи.

Як об’єкт проєктування обрано тему «Web crawler»: програмний сканер, що розпізнає структуру сторінок сайту, переходить за посиланнями, збирає інформацію за заданим терміном, видаляє несемантичні елементи (рекламу, скрипти тощо), зберігає очищені дані у вигляді структурованого набору HTML-файлів і веде статистику відвідань та метадані.

1 Теоретичні відомості

1.1. Шаблон «Abstract Factory»

Призначення патерну: Шаблон «Абстрактна фабрика» використовується для створення сімейств об'єктів без вказівки їх конкретних класів [6]. Для цього виноситься загальний інтерфейс фабрики (AbstractFactory) і створюються його реалізації для різних сімейств продуктів. Хорошим прикладом використання абстрактної фабрики є ADO.NET: існує загальний клас DbProviderFactory, здатний створювати об'єкти типів DbConnection, DbDataReader, DbAdapter та ін.; існують реалізації цих фабрик і об'єктів – SqlProviderFactory, SqlConnection, SqlDataReader, SqlAdapter і так далі. Відповідно, якщо додатку необхідно працювати з різними базами даних (чи потрібна така можливість), то досить 70 використати базові реалізації (Db.) і підставити відповідну фабрику у момент ініціалізації фабрики (Factory = new SqlProviderFactory()). Цей шаблон передусім структурує знання про схожі об'єкти (що називаються сімействами, як класи для доступу до БД) і створює можливість взаємозаміни різних сімейств (робота з Oracle ведеться також, як і робота з SQL Server). Проте, при використанні такої схеми украй незручно розширювати фабрику – для додавання нового методу у фабрику необхідно додати його в усіх фабриках і створити відповідні класи, що створюються цим методом

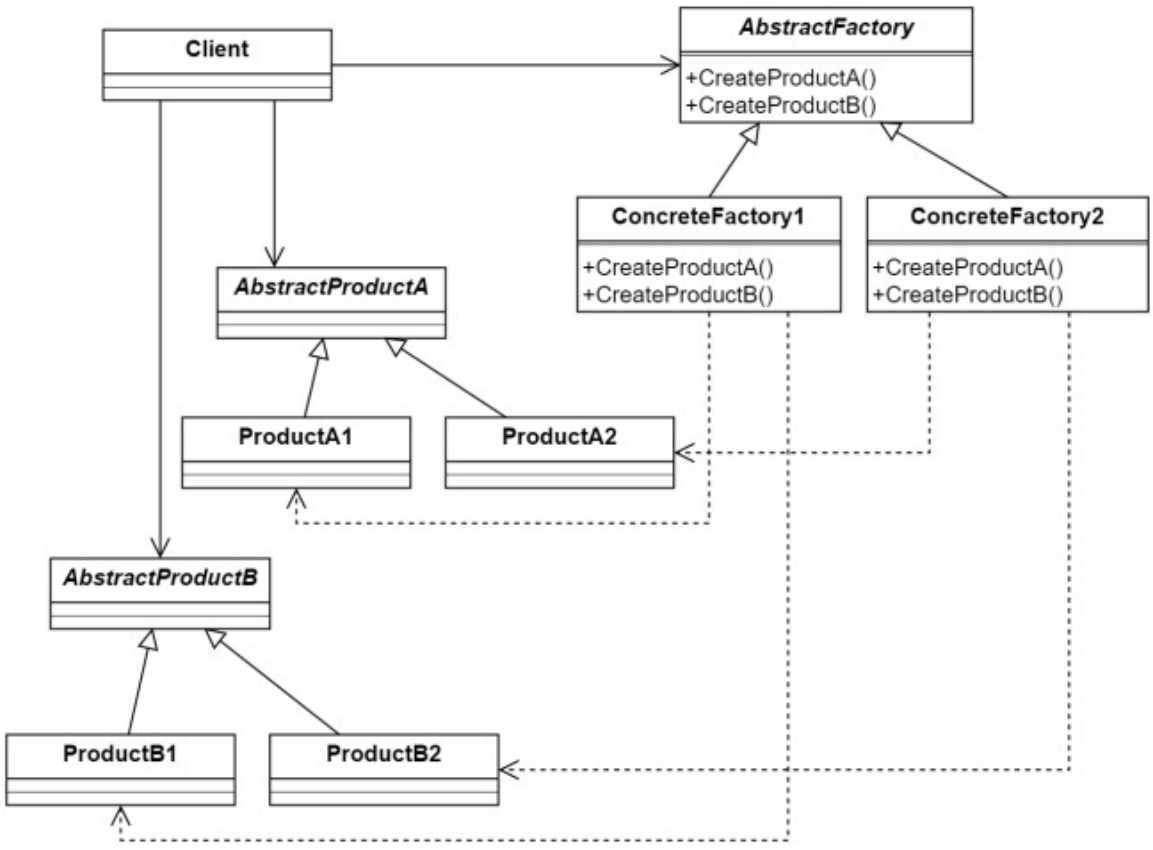

Рисунок 1.1. Структура патерну «Абстрактна фабрика»

1.2. Шаблон «Factory Method»

Призначення: Шаблон «Фабричний метод» визначає інтерфейс для створення об'єктів певного базового типу. Це зручно, коли хочеться додати можливість створення об'єктів не базового типу, а деякого дочірнього. Фабричний метод у такому разі є зачіпкою для впровадження власного конструктора об'єктів. Основна ідея полягає саме в заміні об'єктів їх підтипами, що при цьому зберігає ту ж функціональність; інша частина поведінки об'єктів не є інтерфейсною (AnOperation) і дозволяє взаємодіяти із створеними об'єктами як з об'єктами базового типу. Тому шаблон «Фабричний метод» носить ще назву «Віртуальний конструктор».

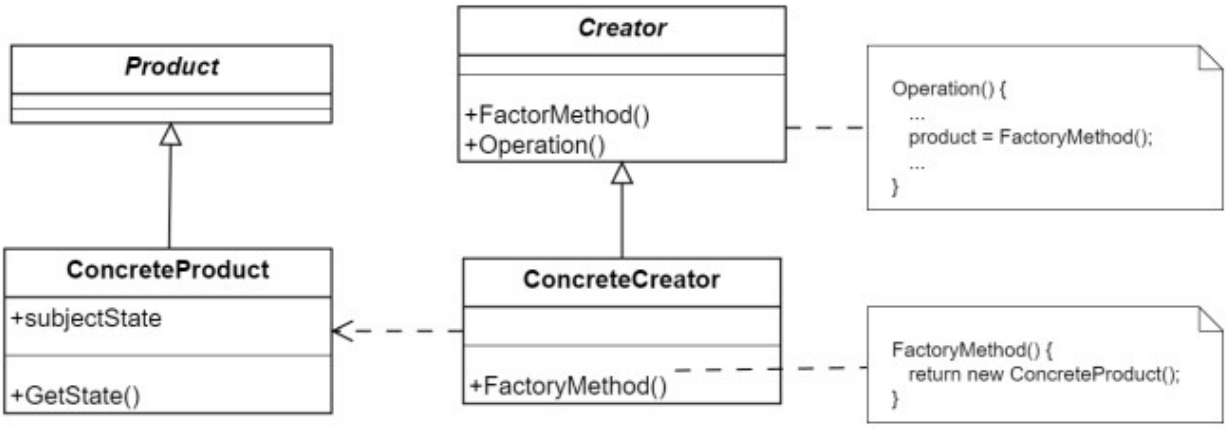

Рисунок 1.2. Структура патерну «Фабричний Метод»

1.3. Шаблон «Memento»

Призначення: Шаблон використовується для збереження і відновлення стану об'єктів без порушення інкапсуляції. Об'єкт «Memento» служить виключно для збереження змін над початковим об'єктом (Originator). Лише початковий об'єкт має можливість зберігати і отримувати стан об'єкту «Memento» для власних цілей, цей об'єкт є «порожнім» для кого-небудь ще. Об'єкт «Caretaker» використовується для передачі і зберігання мементо об'єктів в системі. Таким чином вдається досягти наступних цілей: • зберігання стану повністю відділяється від початкових об'єктів, що полегшує їх реалізацію; • передача об'єктів «Memento» лягає на плечі Caretaker об'єктів, що дозволяє гнучкіше управляти станами об'єктів і спростити дизайн класів початкових об'єктів; 74 • збереження і відновлення стану реалізовані у вигляді двох простих методів і є закритими для кого-небудь ще окрім початкових об'єктів, таким чином не порушуючи інкапсуляцію.

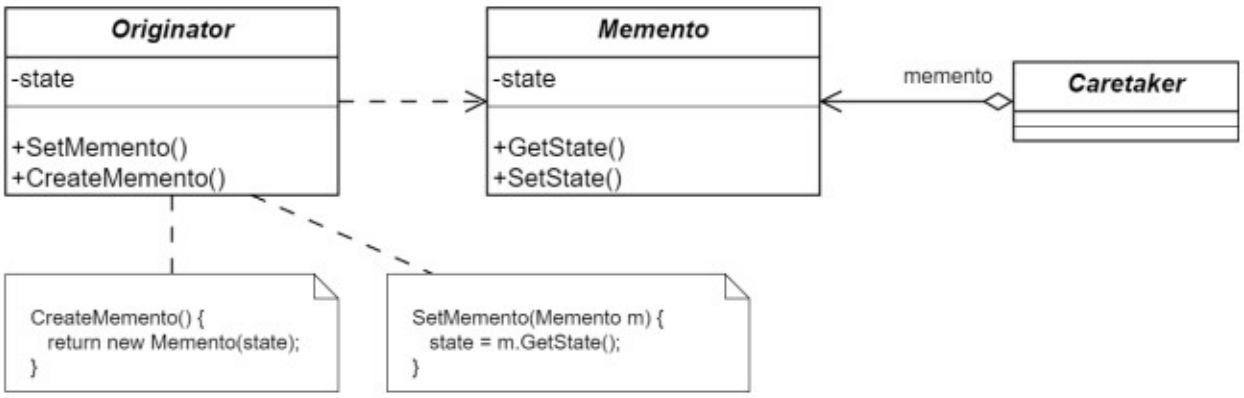

Рисунок 1.3. Структура шаблону «Знімок»

1.4. Шаблон «Observer»

Призначення: Шаблон визначає залежність «один-до-багатьох» таким чином, що коли один об'єкт змінює власний стан, усі інші об'єкти отримують про це сповіщення і мають можливість змінити власний стан також.

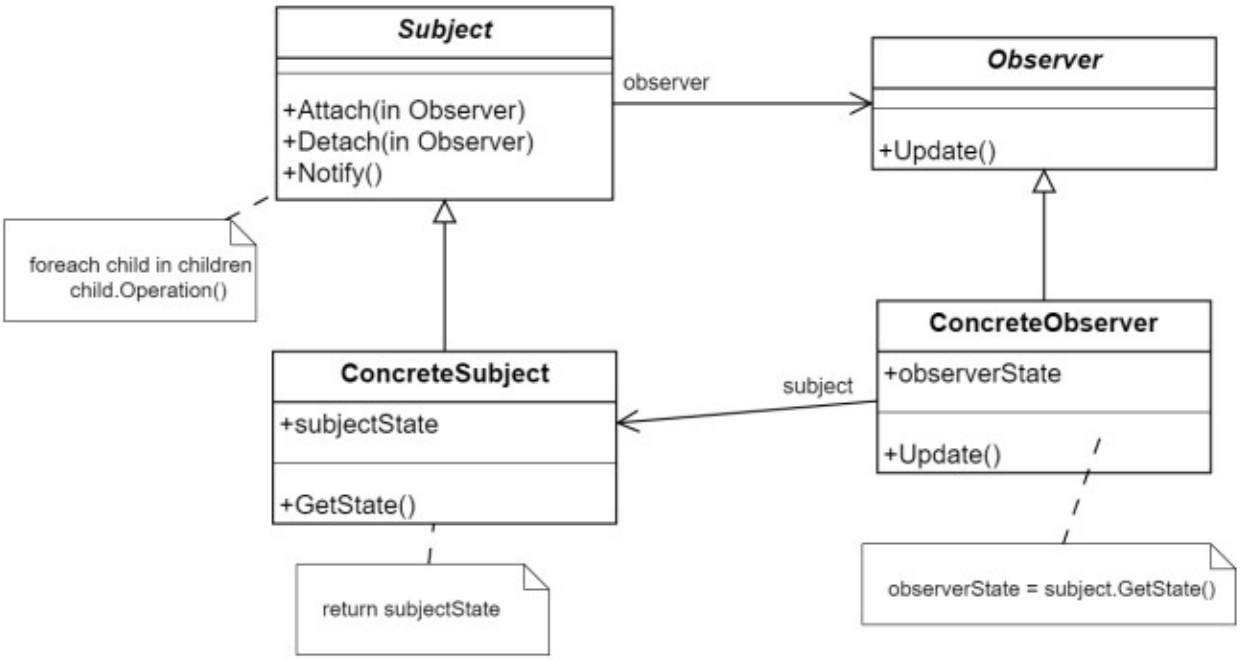

Рисунок 1.4. Структура патерна «Спостерігач»

1.5. Шаблон «Decorator»

Призначення: Шаблон призначений для динамічного додавання функціональних можливостей об'єкту під час роботи програми. Декоратор деяким чином «обертає» (за рахунок агрегації) початковий об'єкт зі

збереженням його функцій, проте дозволяє додати додаткові дії. Такий шаблон надає гнучкіший спосіб зміни поведінки об'єкту чим просте спадкоємство, оскільки початкова функціональність зберігається в повному об'ємі. Більше того, таку поведінку можна застосовувати до окремих об'єктів, а не до усієї системи в цілому. Простим прикладом є накладення смуги прокрутки до усіх візуальних елементів. Кожен об'єкт, який може прокручуватися, обертається в «прокручуваному» елементі, і при необхідності з'являється полоса прокрутки. Початкові функції елементу (наприклад, рядки статусу) залишаються незмінними.

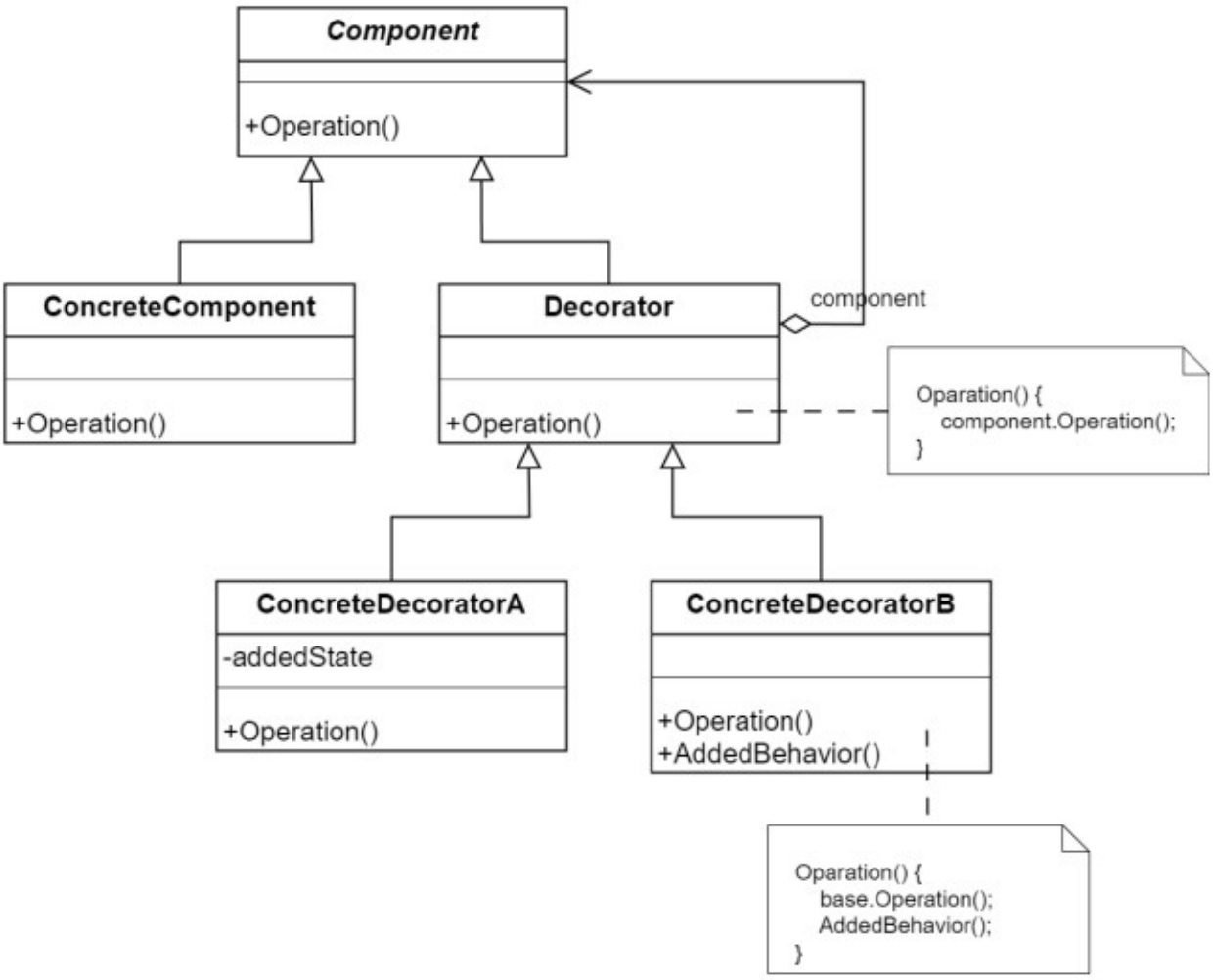

Рисунок 1.5. Структура патерну «Декоратор»

2 Хід роботи

2.1. Застосовування патерну Memento для створення контрольної точки (checkpoint) та відновлення процесу сканування.

Завдання

- Додати механізм збереження стану планувальника URL (черга + вже бачені URL).
- Реалізувати caretaker та сховище checkpoint.
- Додати можливість паузи після N сторінок та відновлення з checkpoint.

2.2 Діаграма класів (фрагмент)

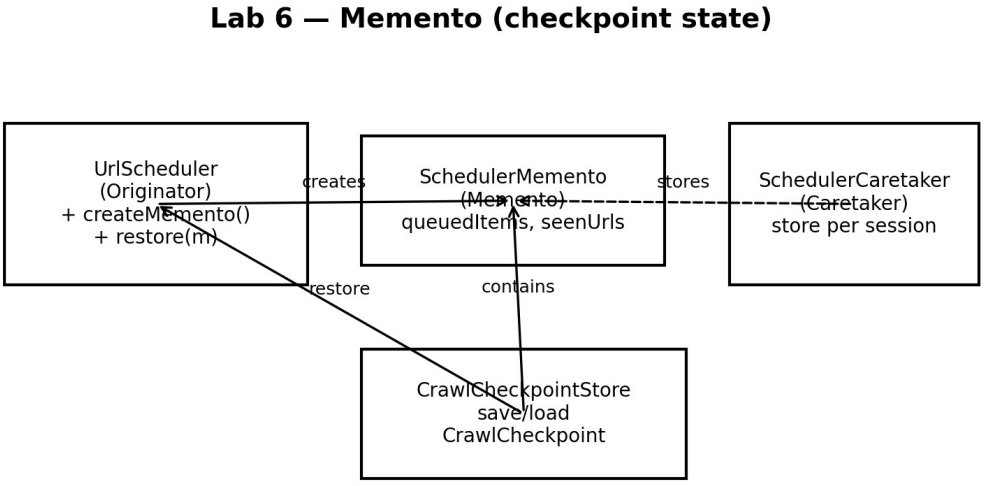

Рисунок 2.2 - Діаграма класів (фрагмент)

2.3. Фрагмент коду (Java)

UrlScheduler: createMemento() / restore()

```
public synchronized SchedulerMemento createMemento() {
    List<String> queued = new ArrayList<>();
    for (UrlTask t : queue) {
        queued.add(t.getUrl() + "|" + t.getDepth());
```

```
        }
        return new SchedulerMemento(queued, new HashSet<>(seen));
    }

    public synchronized void restore(SchedulerMemento memento) {
        queue.clear();
        seen.clear();
        seen.addAll(memento.getSeenUrls());

        for (String item : memento.getQueuedItems()) {
            String[] parts = item.split("\\|");
            if (parts.length != 2) continue;
            String url = parts[0];
            int depth;
            try {
                depth = Integer.parseInt(parts[1]);
            } catch (Exception e) {
                continue;
            }
            queue.add(new UrlTask(url, depth));
        }
    }

    private void enqueue(String url, int depth) {
        if (url == null) return;
        if (depth > maxDepth) return;
        if (!matchesDomain(url)) return;
        if (seen.add(url)) {
            queue.add(new UrlTask(url, depth));
        }
    }

    private boolean matchesDomain(String urlStr) {
        try {
            URI uri = new URI(urlStr);
            String host = uri.getHost();
            if (host == null) return false;
            return host.contains(domainFilter);
```

Пауза та збереження checkpoint у шаблоні алгоритму

```
            if (options != null && options.getPauseAfterPages() > 0 && processed >=
options.getPauseAfterPages()) {
                checkpointStore.save(new CrawlCheckpoint(session.getId(),
profile.getId(), scheduler.createMemento(), processed));
```

```
                session.markPaused();
                sessionRepo.save(session);
                afterCrawl(profile, session, scheduler);
                return session;
            }
        }

        session.markCompleted();
        sessionRepo.save(session);
        checkpointStore.remove(session.getId());
        afterCrawl(profile, session, scheduler);
        return session;
    } catch (Exception e) {
        session.markFailed();
        sessionRepo.save(session);
        afterCrawl(profile, session, scheduler);
        throw e;
    }
}
```

Висновки

У ході виконання лабораторної роботи було реалізовано збереження і відновлення стану обходу через Memento, що дозволяє коректно ставити crawl на паузу та продовжувати роботу з контрольної точки.

Контрольні запитання

1. Яке призначення шаблону «Абстрактна фабрика»?

Шаблон «Абстрактна фабрика» призначений для створення сімейств взаємопов’язаних або взаємозалежних об’єктів без зазначення їх конкретних класів. Він дозволяє ізолювати клієнтський код від деталей створення конкретних реалізацій.

2. Нарисуйте структуру шаблону «Абстрактна фабрика».

Структура шаблону включає:

- AbstractFactory – інтерфейс для створення абстрактних продуктів;
- ConcreteFactory – конкретні фабрики, що реалізують AbstractFactory;
- AbstractProductA / AbstractProductB – інтерфейси продуктів;
- ConcreteProductA / ConcreteProductB – конкретні реалізації продуктів;
- Client – працює тільки з абстрактними інтерфейсами фабрик і продуктів.

3. Які класи входять в шаблон «Абстрактна фабрика», та яка між ними взаємодія?

У шаблон входять AbstractFactory, ConcreteFactory, AbstractProduct та ConcreteProduct.
Client використовує AbstractFactory для створення AbstractProduct.
ConcreteFactory створює конкретні реалізації продуктів, але Client не знає їхніх класів і працює лише з абстракціями.

4. Яке призначення шаблону «Фабричний метод»?

Шаблон «Фабричний метод» визначає інтерфейс для створення об’єкта, але дозволяє підкласам вирішувати, який саме клас створювати. Він переносить відповідальність за створення об’єкта з базового класу на підкласи.

5. Нарисуйте структуру шаблону «Фабричний метод».

Структура включає:

- Creator – базовий клас із фабричним методом;
- ConcreteCreator – перевизначає фабричний метод;
- Product – інтерфейс продукту;
- ConcreteProduct – конкретна реалізація продукту.

6. Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія?

Creator оголошує фабричний метод, який повертає Product.
ConcreteCreator реалізує фабричний метод і створює ConcreteProduct.
Client працює з Creator і не залежить від конкретних класів продуктів.

7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

Фабричний метод створює один об’єкт і використовує наслідування для вибору реалізації. Абстрактна фабрика створює цілі сімейства об’єктів і використовує композицію для групування фабричних методів.

8. Яке призначення шаблону «Знімок» (Memento)?

Шаблон «Знімок» дозволяє зберігати та відновлювати попередній стан об’єкта без порушення інкапсуляції. Він використовується для реалізації механізмів undo/redo.

9. Нарисуйте структуру шаблону «Знімок».

Структура включає:

- Originator – об’єкт, стан якого зберігається;
- Memento – об’єкт, що зберігає стан Originator;
- Caretaker – керує збереженням і відновленням знімків, не змінюючи їх.

10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?

Originator створює Memento і відновлює стан з нього.
Caretaker зберігає Memento, але не має доступу до його внутрішнього стану.
Таким чином забезпечується інкапсуляція стану.

11. Яке призначення шаблону «Декоратор»?

Шаблон «Декоратор» дозволяє динамічно додавати нову поведінку об’єктам без зміни їхнього класу, обгортаючи їх у спеціальні об’єкти-декоратори.

12. Нарисуйте структуру шаблону «Декоратор».

Структура включає:

- Component – спільний інтерфейс;
- ConcreteComponent – базова реалізація;
- Decorator – абстрактний клас, що містить посилання на Component;
- ConcreteDecorator – додає нову поведінку.

13. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія?

ConcreteDecorator реалізує Component і містить посилання на Component. Виклики методів делегуються базовому об’єкту з додаванням нової поведінки до або після виклику.

14. Які є обмеження використання шаблону «Декоратор»?

Основні обмеження:

- збільшення кількості класів у системі;
- ускладнення налагодження через вкладені обгортки;
- важче контролювати порядок застосування декораторів;
- складніше визначити повну поведінку об’єкта під час виконання.