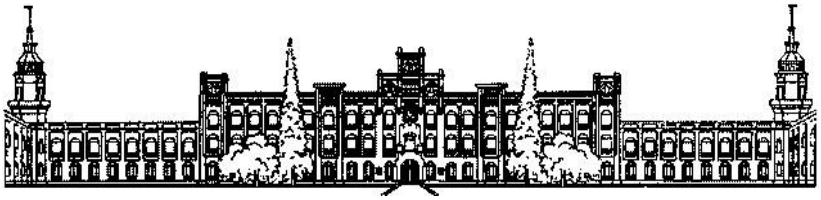


Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Лабораторна робота №5

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконав:

студент групи ІА-32

Нагорний Максим

Перевірив:

Мягкий М. Ю.

Київ 2025

Зміст

Вступ

Тема: Вступ до паттернів проектування.

Мета: Вивчити структуру шаблонів «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» та навчитися застосовувати їх в реалізації програмної системи.

Завдання

- Ознайомитись з короткими теоретичними відомостями.

- Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

- Реалізувати один з розглянутих шаблонів за обраною темою.

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

- Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

14. **Архіватор** (strategy, adapter, factory method, facade, visitor, p2p)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) – додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Теоретичні відомості

Шаблон «Adapter»

Шаблон «Adapter» (Адаптер) належить до структурних шаблонів і застосовується тоді, коли потрібно узгодити класи або інтерфейси, які спочатку не можуть працювати разом. Основна ідея полягає в тому, щоб створити проміжний клас-адаптер, який перетворює один інтерфейс в інший, сумісний із очікуваннями клієнта. Адаптер дозволяє використовувати сторонній або застарілий код без його змін, приховуючи різницю в реалізації. У результаті клієнт взаємодіє із системою через єдиний зрозумілий інтерфейс, що підвищує модульність і спрощує інтеграцію різних бібліотек.

Шаблон «Builder»

Шаблон «Builder» (Будівельник) належить до порожуючих шаблонів і використовується для поетапного створення складних об’єктів. У ситуаціях, коли конструктор має багато параметрів або існує потреба створювати різні варіації одного класу, використання Builder спрощує процес створення та робить код більш читабельним. Замість передачі великої кількості аргументів у конструктор, об’єкт створюється послідовним викликом методів будівельника, а наприкінці повертається готовий екземпляр. Це полегшує модифікацію, тестування і масштабування конструкцій складних об'єктів.

Шаблон «Command»

Шаблон «Command» (Команда) належить до поведінкових шаблонів і дозволяє інкапсулювати дію у вигляді окремого об’єкта. Команда містить у собі інформацію про операцію, яку потрібно виконати, та об’єкт, над яким вона буде виконуватися. Це дає можливість відкладати виконання, ставити команди в чергу, робити їх скасування та повторне виконання. Клієнтський код не знає, як саме виконується команда — він лише викликає її метод. Шаблон часто використовується в системах з undo/redo, макросами, чергами завдань, а також у графічних інтерфейсах.

Шаблон «Chain of Responsibility»

Шаблон «Chain of Responsibility» (Ланцюг обов’язків) — поведінковий шаблон, що дозволяє передавати запит по ланцюгу обробників, поки один із них не виконає потрібну дію. Кожен обробник містить посилання на наступний. Такий підхід виключає жорстке зв’язування відправника запиту з його конкретним обробником. Клієнт не знає, хто саме опрацює запит, а порядок обробників можна

змінювати динамічно. Шаблон зручний у реалізації систем перевірки доступу, обробки помилок, логування та фільтрації даних.

Шаблон «Prototype»

Шаблон «Prototype» (Прототип) належить до породжуючих шаблонів і використовується для створення нових об’єктів шляхом копіювання вже існуючих. Це особливо корисно, коли створення нового екземпляра є дорогим за часом або ресурсами. Прототип дозволяє створювати копії зі збереженням структури та стану початкового об’єкта. Замість використання конструктора, клієнт викликає метод клонування. Це спрощує створення об’єктів зі складною ініціалізацією та дозволяє уникати дублювання коду.

Хід виконання

Для розробки системи архіватора було обрано шаблон проєктування «Адаптер». Вибір цього шаблону обумовлений необхідністю використання сторонніх програм та бібліотек архівування, які мають власні інтерфейси та не можуть безпосередньо працювати з нашим застосунком.

Суть шаблону «Адаптер» полягає у тому, що створюється проміжний клас, який узгоджує інтерфейс сторонньої бібліотеки з інтерфейсом, який очікує клієнтський код. Таким чином, наш архіватор може працювати з різними архіваторами, не змінюючи основну логіку програми. Адаптер перехоплює виклик методу і перенаправляє його на відповідний метод сторонньої бібліотеки, перетворюючи формат виклику у потрібний для системи.

Для реалізації цього шаблону ми виконали такі дії:

1. Створено загальний інтерфейс StrategyArchive, цей інтерфейс визначає метод для архівування.
2. Створено клас ZipStrategyArchive, що імплементується від StrategyArchive та перевизначає метод compress
3. Створено клас TarGzStrategyArchive, що імплементується від StrategyArchive та перевизначає метод compress
4. Створено клас RarStrategyArchive, що імплементується від StrategyArchive та перевизначає метод compress
5. Створено клас ZipAdapter для реалізації логіки архівування типом zip.
6. Створено клас RarAdapter для реалізації логіки архівування типом rar.
7. Створено клас TarAdapter для реалізації логіки архівування типом tar.

Переваги обраного шаблону:
- Можливість роботи з кожним типом архівування окремо не впливаючи, на інші типи архівування;
- Зменшення залежностей між класами, підтримка IoC, тепер модулі верхнього рівня не залежать від модулів нижнього рівня;
- Робота зійснюється через 1 інтерфейс;
- Підвищення гнучкості та розширюваності системи;

На рис. 1 наведена модифікована діаграма класів із представленням використання шаблону в реалізації системи

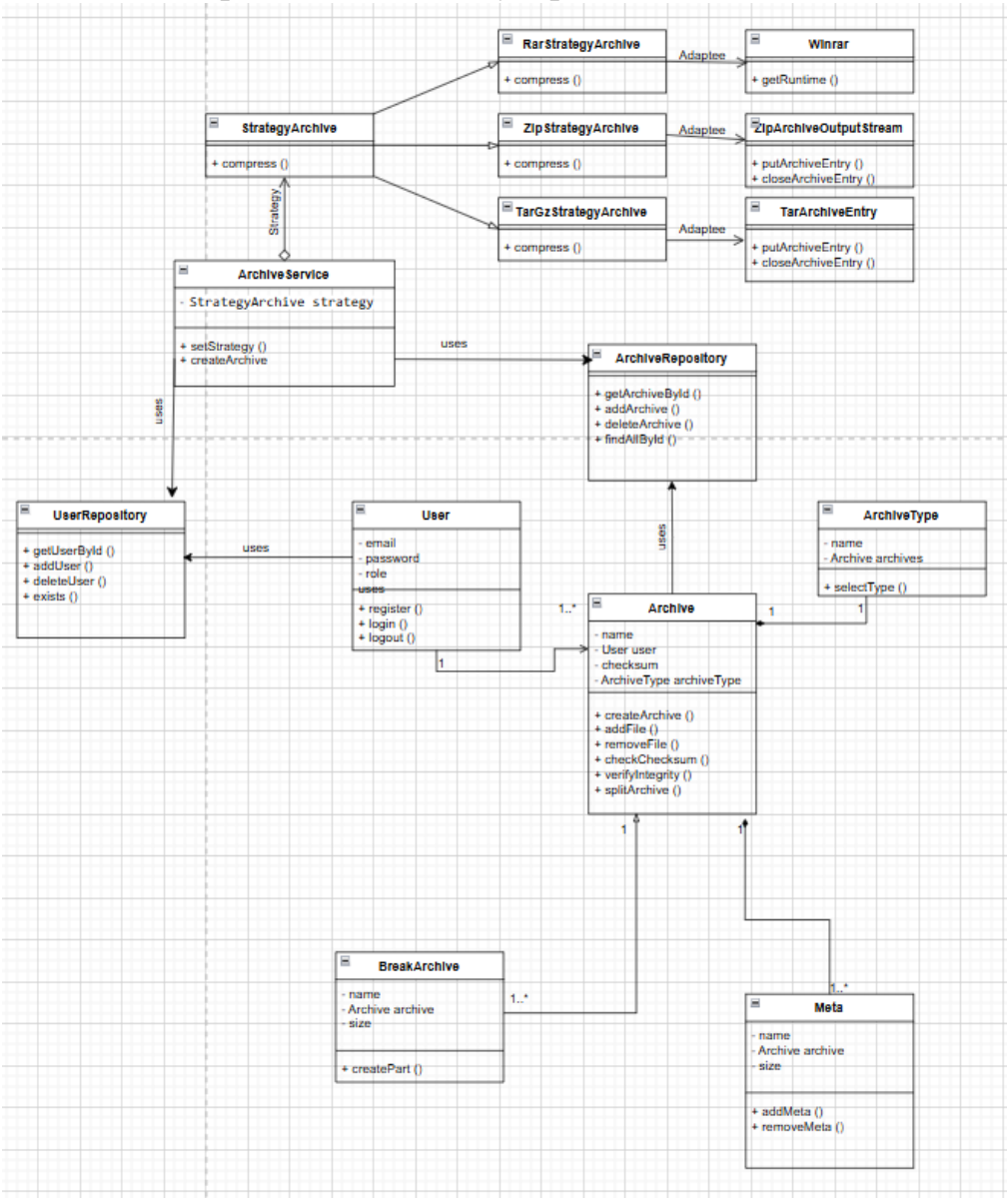


Рис. 1. Діаграма класів

Код програми

```
package com.example.demo.strategy;

import java.io.File;
```

```
import java.io.IOException;
import java.util.List;
public interface StrategyArchive {
    void compress(List<File> files, String outputPath) throws IOException;
}
package com.example.demo.strategy;
import com.example.demo.adapter.ZipAdapter;
import java.io.File;
import java.io.IOException;
import java.util.List;
public class ZipStrategyArchive implements StrategyArchive {
    private final ZipAdapter zipAdapter;
    public ZipStrategyArchive() {
        this.zipAdapter = new ZipAdapter();
    }
    @Override
    public void compress(List<File> files, String outputPath) throws IOException {
        zipAdapter.compress(files, outputPath);
    }
}
package com.example.demo.adapter;
import com.example.demo.strategy.StrategyArchive;
import org.apache.commons.compress.archivers.zip.ZipArchiveEntry;
import org.apache.commons.compress.archivers.zip.ZipArchiveOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.List;
public class ZipAdapter implements StrategyArchive {
    @Override
    public void compress(List<File> files, String outputPath) throws IOException {
        try (ZipArchiveOutputStream zipOut = new ZipArchiveOutputStream(new
FileOutputStream(outputPath))) {
```

```
            for (File file : files) {
                ZipArchiveEntry entry = new ZipArchiveEntry(file.getName());
                zipOut.putArchiveEntry(entry);
                try (FileInputStream fis = new FileInputStream(file)) {
                    fis.transferTo(zipOut);
                }
                zipOut.closeArchiveEntry();
            }
        }
    }
}
package com.example.demo.strategy;
import com.example.demo.adapter.RarAdapter;
import java.io.File;
import java.io.IOException;
import java.util.List;
public class RarStrategyArchive implements StrategyArchive {
    private final RarAdapter rarAdapter;
    public RarStrategyArchive(String winrarPath) {
        this.rarAdapter = new RarAdapter(winrarPath);
    }
    @Override
    public void compress(List<File> files, String outputPath) throws IOException {
        rarAdapter.compress(files, outputPath);
    }
}
package com.example.demo.adapter;
import com.example.demo.strategy.StrategyArchive;
import java.io.File;
import java.io.IOException;
import java.util.List;
public class RarAdapter implements StrategyArchive {
    private final String winrarPath;
    public RarAdapter(String winrarPath) {
```

```
        this.winrarPath = winrarPath;
    }
    @Override
    public void compress(List<File> files, String outputPath) throws IOException {
        StringBuilder cmd = new StringBuilder("\"" + winrarPath + "\" a -ep \"" + outputPath + "\"");
        for (File file : files) {
            cmd.append(" \"").append(file.getAbsolutePath()).append("\"");
        }
        Process process = Runtime.getRuntime().exec(cmd.toString());
        try {
            int exitCode = process.waitFor();
            if (exitCode != 0) {
                throw new IOException("WinRAR архівація завершилась з помилкою: " +
exitCode);
            }
        } catch (InterruptedException e) {
            throw new IOException("Процес WinRAR перервано", e);
        }
    }
}
package com.example.demo.strategy;
import com.example.demo.adapter.TarAdapter;
import java.io.File;
import java.io.IOException;
import java.util.List;
public class TarGzStrategyArchive implements StrategyArchive {
    private final TarAdapter tarAdapter;
    public TarGzStrategyArchive() {
        this.tarAdapter = new TarAdapter();
    }
    @Override
    public void compress(List<File> files, String outputPath) throws IOException {
        tarAdapter.compress(files, outputPath);
    }
```

```
}
package com.example.demo.adapter;
import com.example.demo.strategy.StrategyArchive;
import org.apache.commons.compress.archivers.tar.TarArchiveEntry;
import org.apache.commons.compress.archivers.tar.TarArchiveOutputStream;
import org.apache.commons.compress.compressors.gzip.GzipCompressorOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.List;
public class TarAdapter implements StrategyArchive {
    @Override
    public void compress(List<File> files, String outputPath) throws IOException {
        try (FileOutputStream fos = new FileOutputStream(outputPath);
             GzipCompressorOutputStream gzipOut = new GzipCompressorOutputStream(fos);
             TarArchiveOutputStream tarOut = new TarArchiveOutputStream(gzipOut)) {
            for (File file : files) {
                TarArchiveEntry entry = new TarArchiveEntry(file, file.getName());
                tarOut.putArchiveEntry(entry);
                try (FileInputStream fis = new FileInputStream(file)) {
                    fis.transferTo(tarOut);
                }
             tarOut.closeArchiveEntry();
            }
            tarOut.finish();
        }
    }
}
package com.example.demo.service;
import org.springframework.stereotype.Service;
import com.example.demo.strategy.StrategyArchive;
import java.io.File;
import java.io.IOException;
```

```
import java.util.List;
@Service
public class ArchiveService {
    private StrategyArchive strategy;
    public void setStrategy(StrategyArchive strategy) {
        this.strategy = strategy;
    }
    public void createArchive(List<File> files, String outputPath) throws IOException {
        strategy.compress(files, outputPath);
    }
}
```

Висновок

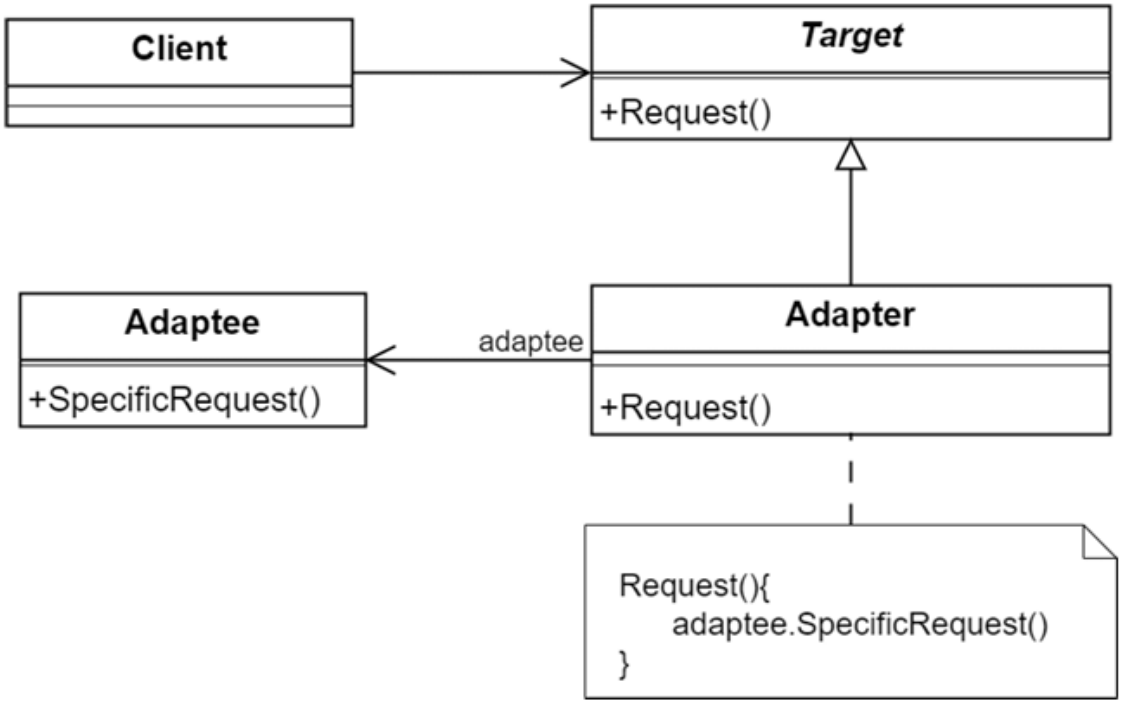
Під час виконання лабораторної роботи було досліджено поняття шаблону проектування. Почалася робота з вибору певного шаблону проектування. Після цього було реалізовано шаблон «адаптер» згідно теми «архіватор», під час реалізації було розроблено загальний інтерфейс ArchiveStrategy, який визначає метод compress для архівування та розроблено класи – наслідники ZipAdapter, TarAdapter, RarAdapter для реалізації логіки архівування різних типів та останнім етапом було модифікації діаграми класів з представленням використання шаблону «адаптер».

Питання до лабораторної роботи

1. **Яке призначення шаблону «Адаптер»?**

   Призначення шаблону «Адаптер» полягає в тому, щоб перетворити інтерфейс одного класу на інтерфейс, який очікує клієнт. Він дозволяє об'єктам з несумісними інтерфейсами працювати разом, виступаючи в ролі "перехідника" або "перекладача" між ними.

2. **Нарисуйте структуру шаблону «Адаптер».**



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

   В шаблон «Адаптер» входять: Target , Client, Adaptee та Adapter (клас-перехідник, який реалізує інтерфейс Target. Взаємодія відбувається так: Client викликає метод у Adapter через інтерфейс Target; Adapter отримує цей виклик, транслює його у один або декілька викликів методів об'єкта Adaptee і повертає результат у Client.

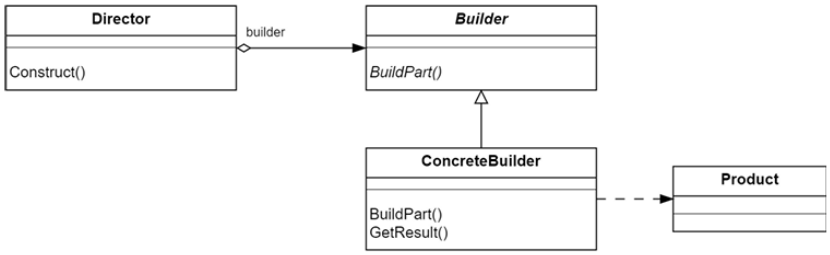
4. **Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів?**

   Різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів полягає у способі зв'язку з Adaptee. Адаптер об’єктів використовує композицію: він містить у собі посилання на екземпляр Adaptee і делегує йому виклики. Адаптер класів використовує множинне успадкування, клас Adapter одночасно успадковує інтерфейс Target і реалізацію класу Adaptee.

5. **Яке призначення шаблону «Будівельник»?**

   Призначення шаблону «Будівельник» — відокремити процес конструювання складного об'єкта від його кінцевого представлення. Це дозволяє використовувати один і той самий процес конструювання для створення різних варіантів об'єкта, а також дає змогу створювати об'єкти покроково, уникаючи перевантажених конструкторів.

6. **Нарисуйте структуру шаблону «Будівельник».**



7. **Які класи входять в шаблон «Будівельник», та яка між ними взаємодія?**

   В шаблон «Будівельник» входять: Product, Builde), ConcreteBuilder та Director. Взаємодія: Клієнт створює об'єкт ConcreteBuilder і передає його Director. Director викликає кроки на Builder. Коли конструювання завершено, Клієнт отримує готовий Product безпосередньо від ConcreteBuilder.

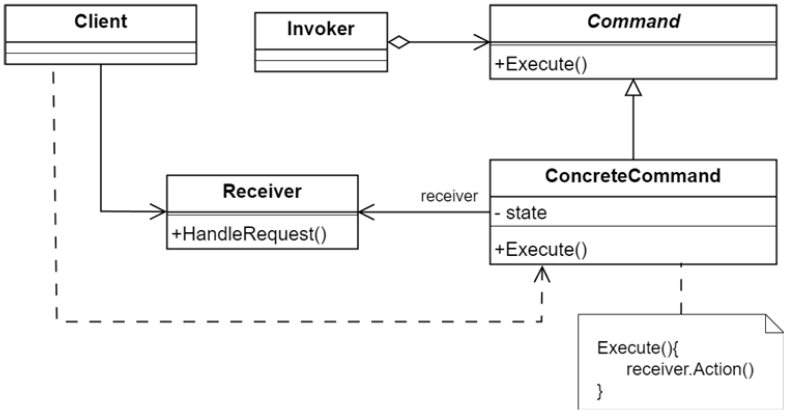
8. **У яких випадках варто застосовувати шаблон «Будівельник»"?**

   Шаблон «Будівельник» варто застосовувати, коли процес створення об'єкта складний, багатоетапний або коли об'єкт має багато опціональних параметрів, що призвело б до великої кількості перевантажених конструкторів. Його також використовують, коли необхідно, щоб один і той самий процес конструювання міг створювати різні представлення (варіації) об'єкта.

9. **Яке призначення шаблону «Команда»?**

   Призначення шаблону «Команда» — інкапсулювати запит або операцію як повноцінний об'єкт. Це дозволяє параметризувати клієнтів різними запитами, ставити запити в чергу, логувати їх.

10. **Нарисуйте структуру шаблону «Команда».**



11. **Які класи входять в шаблон «Команда», та яка між ними взаємодія?**

В шаблон «Команда» входять: Command, ConcreteCommand, Receiver, Invoker та Client.

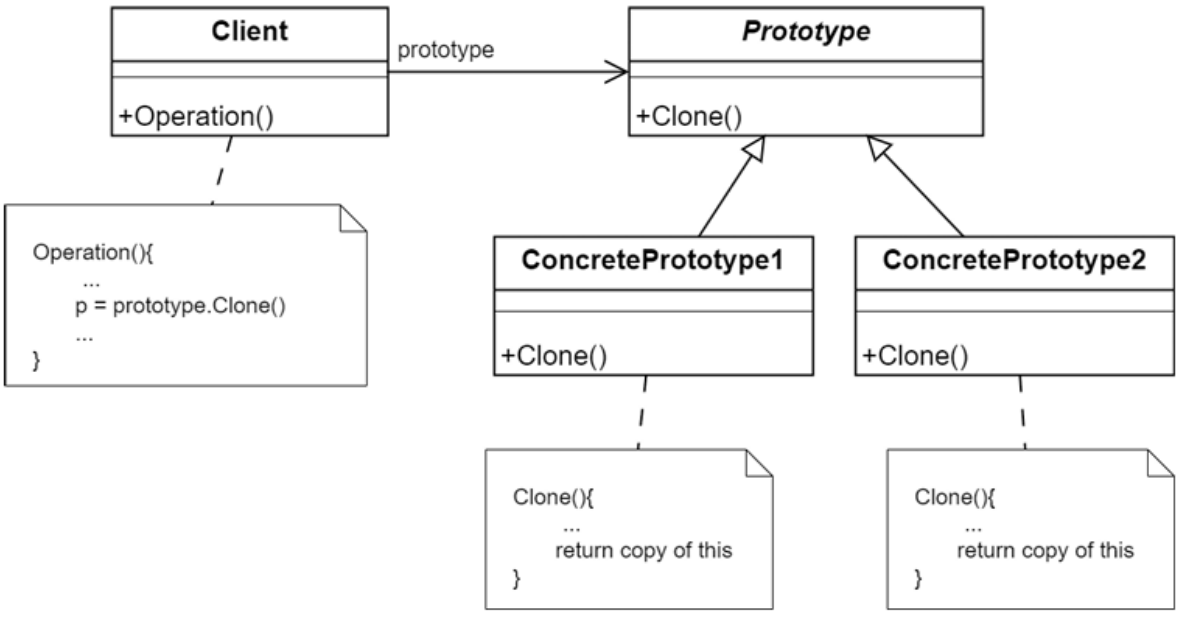
12.Розкажіть як працює шаблон «Команда».

Шаблон «Команда» працює наступним чином - клієнт створює об'єкт ConcreteCommand і налаштовує його, передаючи йому посилання на Receiver. Потім цей об'єкт-команда передається Invoker. Коли Invoker готовий виконати дію, він просто викликає єдиний метод команди. Об'єкт команди, у свою чергу, викликає один або декілька методів у пов'язаного з ним Receiver, який і виконує всю реальну роботу. Таким чином, Invoker повністю відокремлений від Receiver.

13.Яке призначення шаблону «Прототип»?

Призначення шаблону «Прототип» — надати механізм створення нових об'єктів шляхом копіювання існуючого об'єкта-прототипу, замість створення об'єкта "з нуля" через виклик конструктора. Це особливо корисно, коли створення об'єкта є ресурсозатратною операцією або коли потрібно отримати точну копію, включаючи стан приватних полів.

14.Нарисуйте структуру шаблону «Прототип».



15.Які класи входять в шаблон «Прототип», та яка між ними взаємодія?

В шаблон «Прототип» входять два основних учасники - Prototype та ConcretePrototype . Також є Client, який використовує прототип для створення нових об'єктів. Взаємодія дуже проста - Client, маючи

посилання на об'єкт ConcretePrototype, викликає його метод clone(), щоб отримати новий, незалежний екземпляр цього об'єкта з тим самим станом.

16. Які можна привести приклади використання шаблону «Ланцюжок відповідальності»?

Розробка системи зі складними UI формами, які містять багато вкладених один в один візуальних компонентів.