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

Лабораторна робота № 5

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Варіант - 17

Виконав
Студент групи ІА-31:
Лисенко В. Є

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

1. Мета:

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

2. Теоретичні відомості:

Adapter: Адаптує інтерфейс одного об’єкта до іншого, дозволяючи несумісним компонентам працювати разом (наприклад, уніфікація інтерфейсів бібліотек принтерів). Спрощує інтеграцію без змін у коді.
Builder: Відокремлює процес створення об’єкта від його представлення, доречний для складних або багатоформних конструкцій (наприклад, відповіді веб-сервера). Забезпечує гнучкий контроль.
Command: Перетворює виклик методу в об’єкт, дозволяючи гнучкі системи команд із відміною, логуванням чи плануванням (наприклад, дії в інтерфейсі). Покращує модульність.
Chain of Responsibility: Передає запити по ланцюжку обробників, поки один не виконає (наприклад, контекстні меню). Зменшує зв’язки та спрощує зміни.
Prototype: Створює об’єкти клонуванням прототипу, корисний для складних об’єктів або динамічних змін (наприклад, редактор рівнів гри). Зменшує ієрархію спадкування.

3. Завдання:

1) Ознайомитись з короткими теоретичними відомостями.
2) Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3) Реалізувати один з розглянутих шаблонів за обраною темою.
4) Реалізувати не менше 3-х класів відповідно до обраної теми.
5) 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.

4. Хід роботи:

Мій варіант:

17. **System activity monitor** (iterator, command, abstract factory, bridge, visitor,

SOA) Монітор активності системи повинен зберігати і запам'ятовувати статистику використовуваних компонентів системи, включаючи навантаження на процесор, обсяг займаної оперативної пам'яті, натискання клавіш на клавіатурі, дії миші (переміщення, натискання), відкриття вікон і зміна вікон; будувати звіти про використання комп'ютера за різними критеріями (% часу перебування у веб-браузері, середнє навантаження на процесор по годинах, середній час роботи

комп'ютера по днях і т.д.); правильно поводитися з «простоюванням» системи – відсутністю користувача.

Паттерн Command дозволяє:

1. Інкапсулювати запит на виконання дії (генерація/очищення) у вигляді окремого об'єкта.
2. Відокремити об'єкт, що ініціює операцію (кнопка на MainWindow), від об'єкта, що знає, як її виконати (SystemController).
3. Забезпечити можливість розширення системи новими командами (наприклад, "Архівація логів") без зміни коду інтерфейсу користувача.
4. Зберігати історію виконаних команд (через CommandInvoker), що в майбутньому дозволить реалізувати функцію скасування дій (Undo).

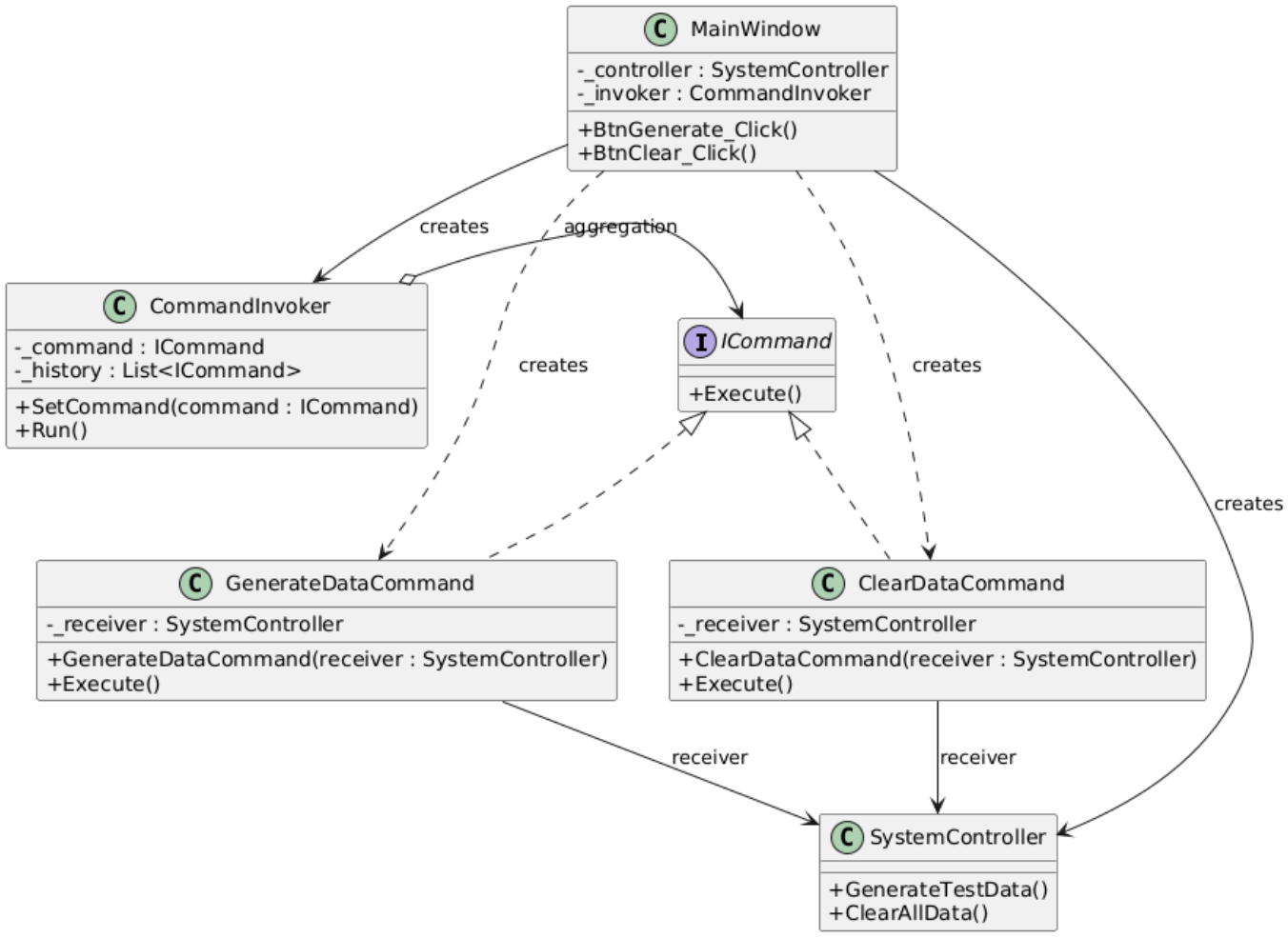


Рисунок 1 - Структура патерну Command

ICommand	Інтерфейс команди	Оголошує метод Execute(), спільний для всіх операцій.
SystemController	Одержувач	Клас бізнес-логіки, який містить реалізацію операцій роботи з базою даних (методи GenerateTestData та ClearAllData). Він знає, *як* виконувати роботу.

GenerateDataCommand / ClearDataCommand	Конкретні команди	Реалізують інтерфейс ICommand. Вони пов'язують дію з конкретним одержувачем (SystemController) і викликають відповідні методи.
CommandInvoker	Ініціатор	Відповідає за запуск команди. Він не знає деталей реалізації команди, лише викликає метод Execute(). Також зберігає історію виконаних команд.
MainWindow	Клієнт	Створює конкретну команду, передає їй одержувача і призначає її ініціатору (Invoker) при натисканні кнопки.

Програмна реалізація:

ICommand.cs:

```
namespace SystemActivityMonitor.Data.Patterns.Command
{
    public interface ICommand
    {
        void Execute();
    }
}
```

SystemController.cs:

```
public class SystemController
    {
        public void ClearAllData()
        {
            using (var db = new MonitorDbContext())
            {
                db.ResourceLogs.RemoveRange(db.ResourceLogs);
                db.Sessions.RemoveRange(db.Sessions);
                db.SaveChanges();
            }
        }
        public void GenerateDataWithFactory(IMonitorFactory factory)
        {
            ICpuSensor cpuSensor = factory.CreateCpuSensor();
            IRamSensor ramSensor = factory.CreateRamSensor();

            using (var db = new MonitorDbContext())
            {
                var admin = db.Users.FirstOrDefault(u => u.Username == "admin");
                if (admin == null) return;
```

```
            var session = new Session
            {
                UserId = admin.Id,
                MachineName = "FULL-PC",
                OSVersion = "Windows 11 Pro"
            };
            db.Sessions.Add(session);
            db.SaveChanges();

            var rnd = new Random();
            string[] windows = { "Google Chrome", "Visual Studio", "Google Chrome", "Telegram", "Word" };
            for (int i = 0; i < 10; i++)
            {
                db.ResourceLogs.Add(new ResourceLog
                {
                    SessionId = session.Id,
                    CpuLoad = cpuSensor.GetCpuLoad(),
                    RamUsage = ramSensor.GetFreeRam(),
                    ActiveWindow = windows[rnd.Next(windows.Length)],
                    CreatedAt = DateTime.UtcNow.AddSeconds(i * 2)
                });
            }
            string[] keyActions = { "Ctrl+C", "Ctrl+V", "Enter", "Alt+Tab", "Space" };
            string[] mouseActions = { "Left Click", "Right Click", "Scroll Down", "Double Click" };
            for (int i = 0; i < 5; i++)
            {
                bool isKeyboard = rnd.Next(0, 2) == 0;
                db.InputEvents.Add(new InputEvent
                {
                    SessionId = session.Id,
                    EventType = isKeyboard ? "Keyboard" : "Mouse",
                    Details = isKeyboard
                        ? keyActions[rnd.Next(keyActions.Length)]
                        : mouseActions[rnd.Next(mouseActions.Length)],
                    CreatedAt = DateTime.UtcNow.AddSeconds(i * 3)
                });
            }
            db.SaveChanges();
        }
    }
```

GenerateDataCommand.cs:

```
public class GenerateDataCommand : ICommand
    {
```

```
        private readonly SystemController _receiver;
        private readonly IMonitorFactory _factory;

        public GenerateDataCommand(SystemController receiver, IMonitorFactory factory)
        {
            _receiver = receiver;
            _factory = factory;
        }

        public void Execute()
        {
            _receiver.GenerateDataWithFactory(_factory);
        }
    }
```

CommandInvoker.cs:

```
public class CommandInvoker
    {
        private ICommand _command;

        private List<ICommand> _history = new List<ICommand>();

        public void SetCommand(ICommand command)
        {
            _command = command;
        }

        public void Run()
        {
            if (_command != null)
            {
                _command.Execute();
                _history.Add(_command);
            }
        }
    }
```

MainWindow.xaml.cs:

```
private void BtnGenerate_Click(object sender, RoutedEventArgs e)
        {
            IMonitorFactory selectedFactory;

            if (cmbFactoryMode.SelectedIndex == 0)
                selectedFactory = new StandardFactory();
            else
```

```
            selectedFactory = new CriticalFactory();

        ICommand generateCmd = new GenerateDataCommand(_controller, selectedFactory);

        _invoker.SetCommand(generateCmd);
        _invoker.Run();

        MessageBox.Show($"Дані згенеровано! Режим: {((ComboBoxItem)cmbFactoryMode.SelectedItem).Content}");
        BtnLoadIterator_Click(null, null);
    }
```

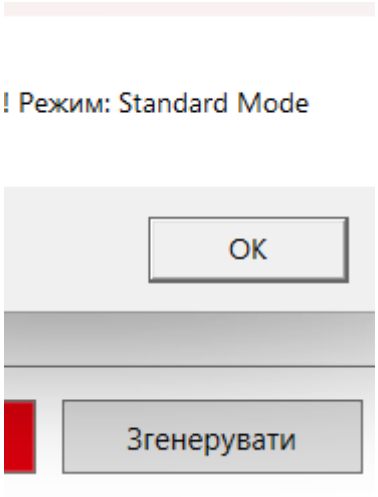


Рисунок 2 – Результат виконання команди генерації даних через Invoker.

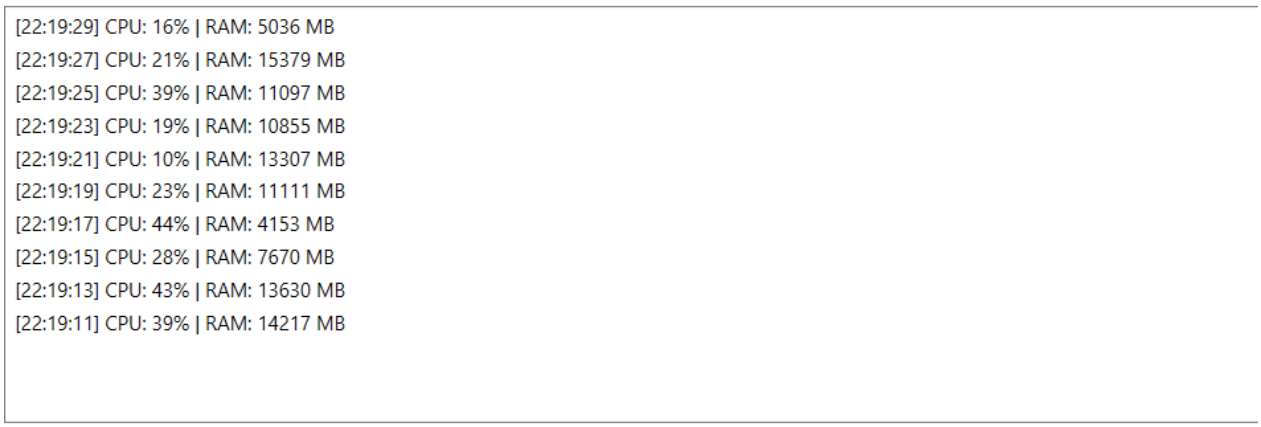


Рисунок 3 – Результат виконання

5. Висновок

У ході виконання лабораторної роботи було вивчено та реалізовано структурний патерн проєктування Command. Даний патерн було застосовано для організації взаємодії між інтерфейсом користувача та бізнес-логікою системи моніторингу.

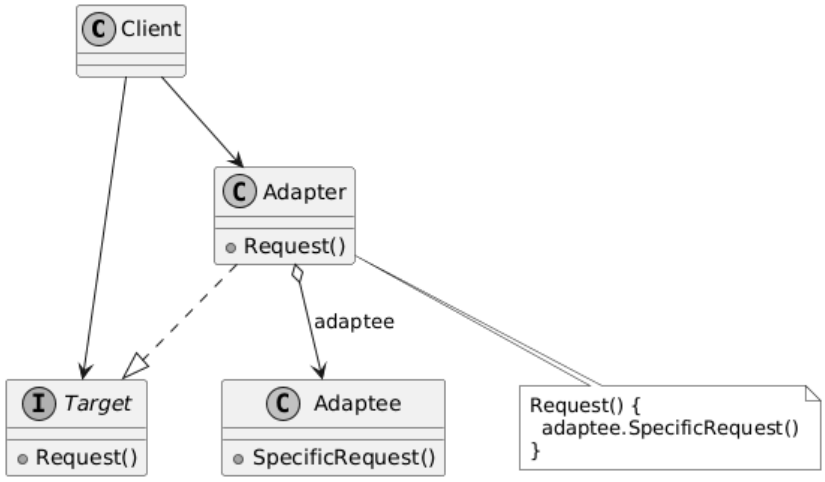
В межах системи було створено власну реалізацію патерну без використання вбудованого інтерфейсу System.Windows.Input.ICommand, що дозволило детально розібрати механізм його роботи. Було розроблено:

1. Клас SystemController (Receiver), який містить безпосередню логіку роботи з даними (генерація та очищення). 2. Класи команд GenerateDataCommand та ClearDataCommand, що інкапсулюють виклики методів контролера .
2. Клас CommandInvoker, який ініціює виконання команд та зберігає історію їх викликів.

Використання патерну Command дозволило повністю відокремити графічний інтерфейс від логіки обробки даних, усунути дублювання коду та забезпечити гнучкість системи для додавання нових операцій у майбутньому.

6. Контрольні питання:

1) Яке призначення шаблону «Адаптер»? Адаптує інтерфейс одного об’єкта до іншого, дозволяючи несумісним компонентам працювати разом (наприклад, уніфікація бібліотек принтерів).
2) Нарисуйте структуру шаблону «Адаптер».

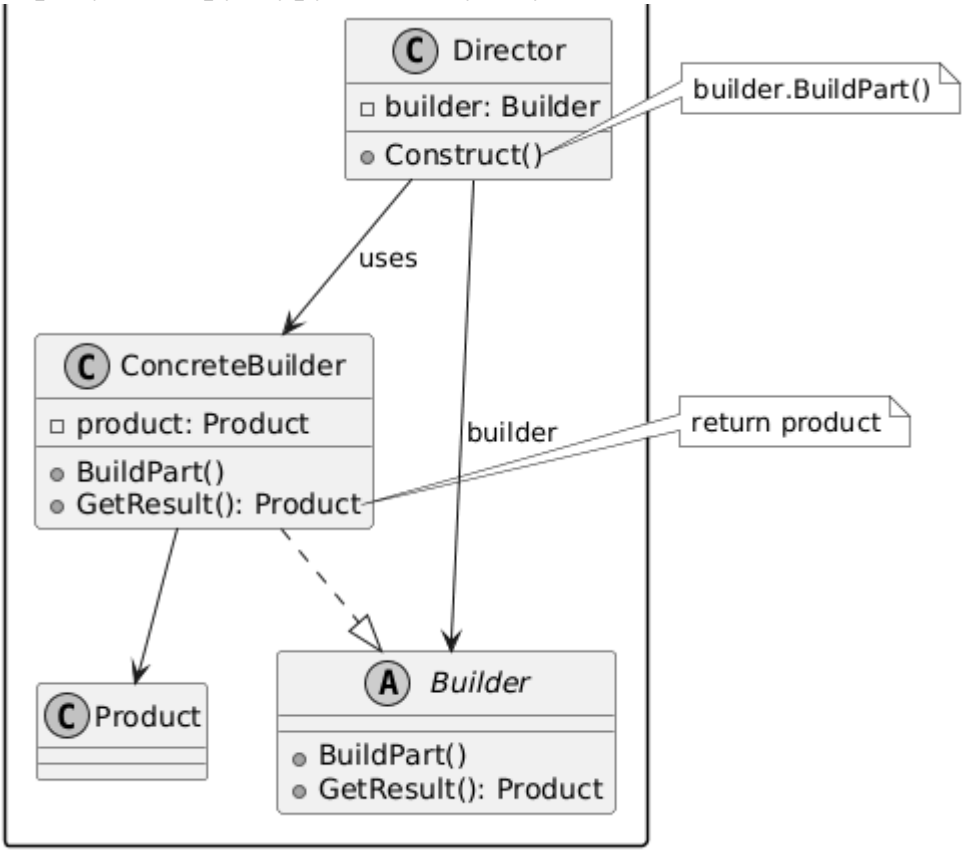


3) Які класи входять в шаблон «Адаптер», та яка між ними взаємодія?

Client, Target, Adapter, Adaptee. Client працює з Target через адаптований інтерфейс, Adapter перенаправляє виклики до Adaptee.

4) Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів? Об’єктний адаптер використовує композицію, класовим — успадкування.
5) Яке призначення шаблону «Будівельник»? Відокремлює процес створення об’єкта від його представлення, придатний для складних або багатоформних конструкцій.

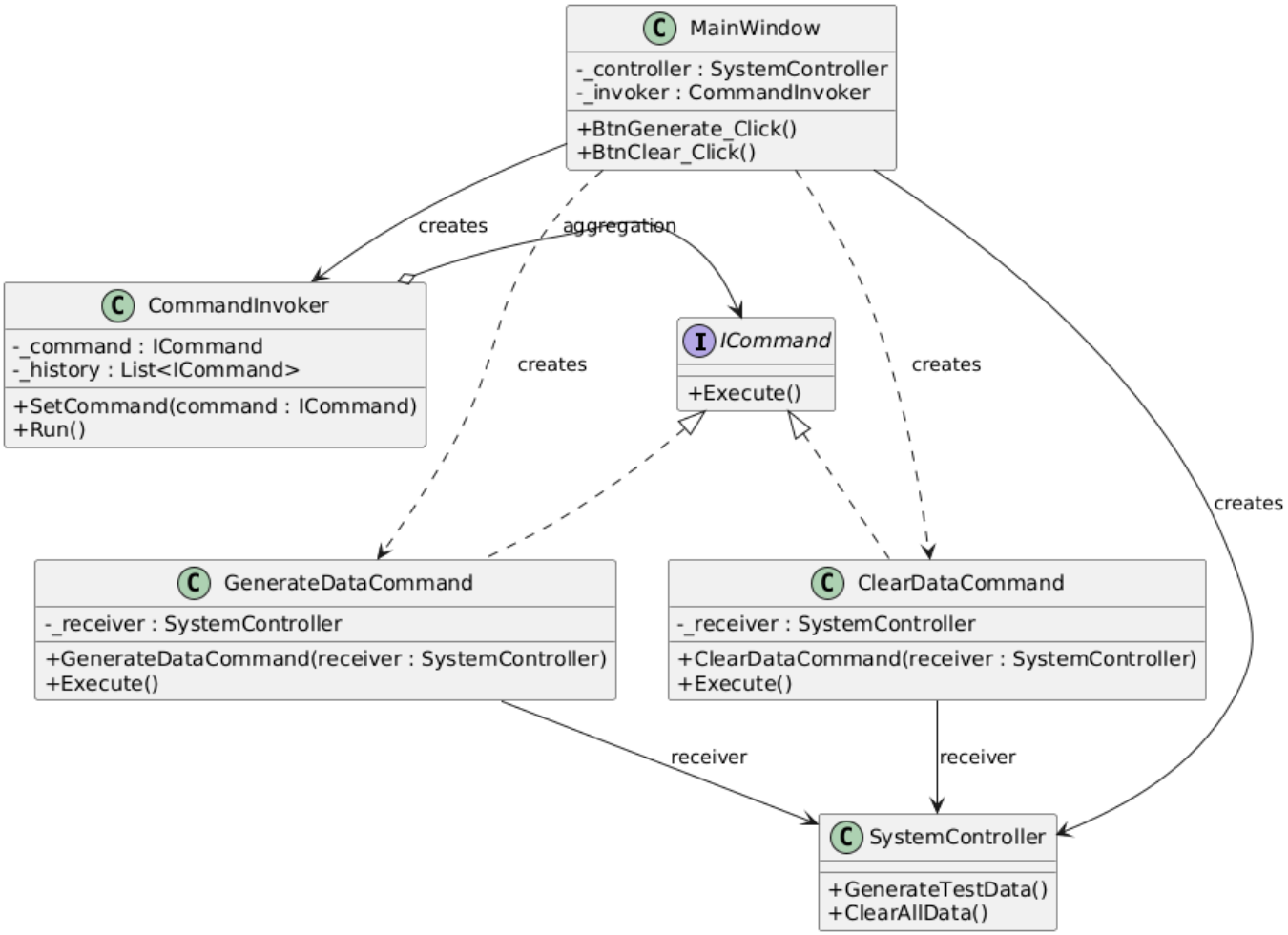
6) Нарисуйте структуру шаблону «Будівельник».



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

Director, Builder, ConcreteBuilder, Product. Director керує будівництвом, Builder визначає інтерфейс, ConcreteBuilder реалізує його, Product — результат.

8) У яких випадках варто застосовувати шаблон «Будівельник»? При складному процесі створення або необхідності різних форм об’єкта.
9) Яке призначення шаблону «Команда»? Перетворює виклик методу в об’єкт для гнучкої системи команд із відміною, логуванням чи плануванням.
10) Нарисуйте структуру шаблону «Команда».



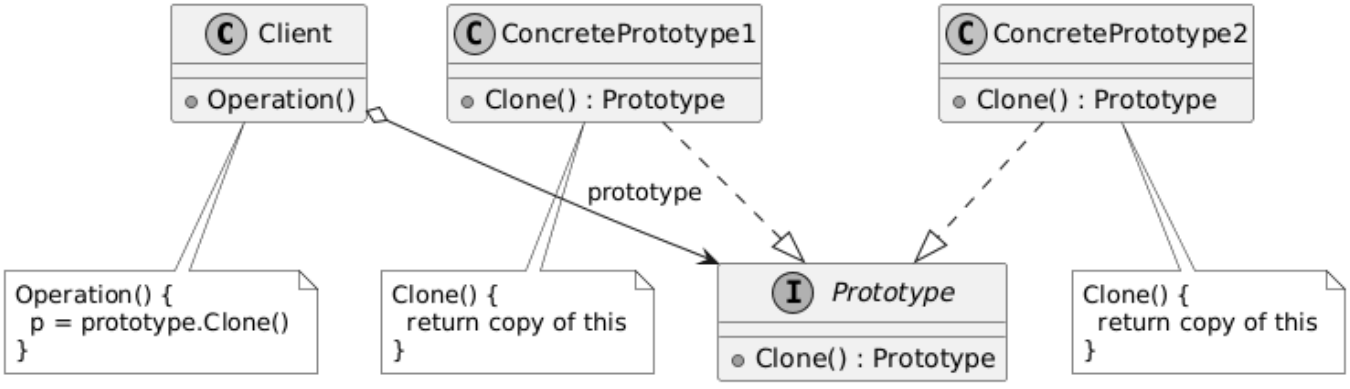
11) Які класи входять в шаблон «Команда», та яка між ними взаємодія?

Client, Invoker, Command, ConcreteCommand, Receiver. Client створює команду, Invoker виконує її, Receiver реалізує дію.

12) Розкажіть як працює шаблон «Команда». Команда інкапсулює запит як об’єкт, який передається Invoker до Receiver для виконання, підтримуючи додаткові функції (скасування, логування).

13) Яке призначення шаблону «Прототип»? Створює об’єкти клонуванням прототипу, зменшуючи ієрархію спадкування.

14) Нарисуйте структуру шаблону «Прототип».



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

Client, Prototype. Client клонувати об’єкт через метод Prototype.

16) Які можна привести приклади використання шаблону «Ланцюжок відповідальності»? Формування контекстного меню в UI, обробка запитів у ієрархії документів.