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 №8

З дисципліни: Технології розроблення програмного забезпечення

Тема: “Паттерн Visitor”

Виконав:

студент групи ІА-14

Рисаков Богдан

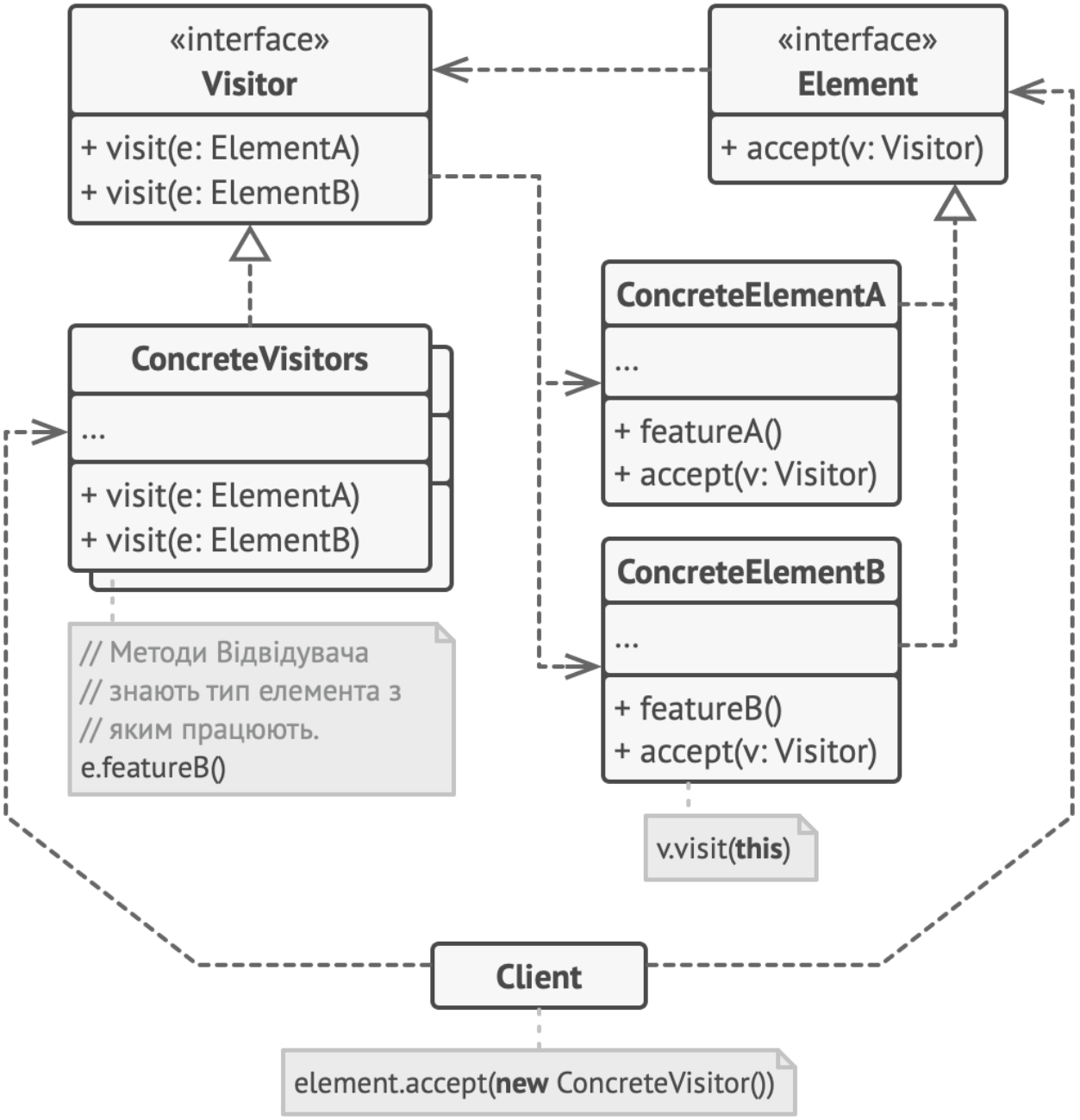
Перевірив:

Мягкий Михайло Юрійович

Київ, 2023

Мета: Реалізувати паттерн Visitor у проекті на тему System Activity Monitor

Відвідувач — це поведінковий патерн проектування, що дає змогу додавати до програми нові операції, не змінюючи класи об'єктів, над якими ці операції можуть виконуватися.



Приклад структури реалізації та використання патерну. Reference

Переваги:

- Спрощує додавання операцій, працюючих зі складними структурами об’єктів.
- Об’єднує споріднені операції в одному класі.
- Відвідувач може накопичувати стан при обході структури елементів.

Недоліки:

- Патерн невиправданий, якщо ієрархія елементів часто змінюється.
- Може призвести до порушення інкапсуляції елементів.

Реалізація:

Спочатку про проблему яку будемо вирішувати:

У минулої лабораторної я зазначив порушення SPR у класі Monitoring:

```
public interface Monitoring {
   void accept(Visitor visitor);

   void startMonitoring(boolean isMonitoringStarted);

   Report generateReportByTime(LocalDateTime end);

   Report generateScheduledReport(LocalDateTime start, LocalDateTime end);

   Report startReporting();

   Report stopReporting();
}
```

Як видно з’явилось 2 проблеми:

- Класи для моніторингу тепер ще і роблять звіти (2 відповідальність)
- Для додавання нового типу звіту треба змінювати цей клас - OCP порушення

Це не вірно з точки зору ООР та для вирішення ідеально підійде паттерн Visitor який даст нам змогу створювати нові генератори та не переписувати класи для моніторингу:

```
public interface Monitoring {
   Report accept(ReportVisitor reportVisitor);

   void startMonitoring(boolean isMonitoringStarted);
}
```

Тепер моніторинг нічого не знає про типи звітів що треба зробити

```
public interface ReportVisitor {
   Report visit(CpuLoadMonitoringService cpuLoadMonitoringService);

   Report visit(KeyLoggerMonitoringService keyLoggerMonitoringService);

   Report visit(MemoryMonitoringService memoryMonitoringService);

   Report visit(MouseTrackerMonitoringService mouseTrackerMonitoringService);

   Report visit(WindowsMonitoringService windowsMonitoringService);
}
```

У свою чергу відвідувач знає про кожний моніторинг та буде знати як з ним працювати.

Приклад реалізації:

```
@Getter
@Setter
public class ReportByTimeGenerator implements ReportVisitor {

   private LocalDateTime dueToTime;

   public ReportByTimeGenerator(LocalDateTime dueToTime) {
       this.dueToTime = dueToTime;
   }

   @Override
   public Report visit(CpuLoadMonitoringService cpuLoadMonitoringService) {
       return null;
   }

   @Override
   public Report visit(KeyLoggerMonitoringService keyLoggerMonitoringService) {
       return null;
   }

   @Override
   public Report visit(MemoryMonitoringService memoryMonitoringService) {
       return null;
   }

   @Override
   public Report visit(MouseTrackerMonitoringService
mouseTrackerMonitoringService) {
       return null;
   }

   @Override
   public Report visit(WindowsMonitoringService windowsMonitoringService) {
       return null;
   }
}
```

Вся логика теперь прихована у класах - відвідувачах.
Тут, наприклад, може бути база даних чи меседж брокер - у минулому підході нам би пришлось прописувати це у кожному моніторингу.

Висновок: Я реалізував патерн Visito