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 №7

З дисципліни: Технології розроблення програмного забезпечення

Тема: “Паттерн Bridge”

Виконав:

студент групи ІА-14

Рисаков Богдан

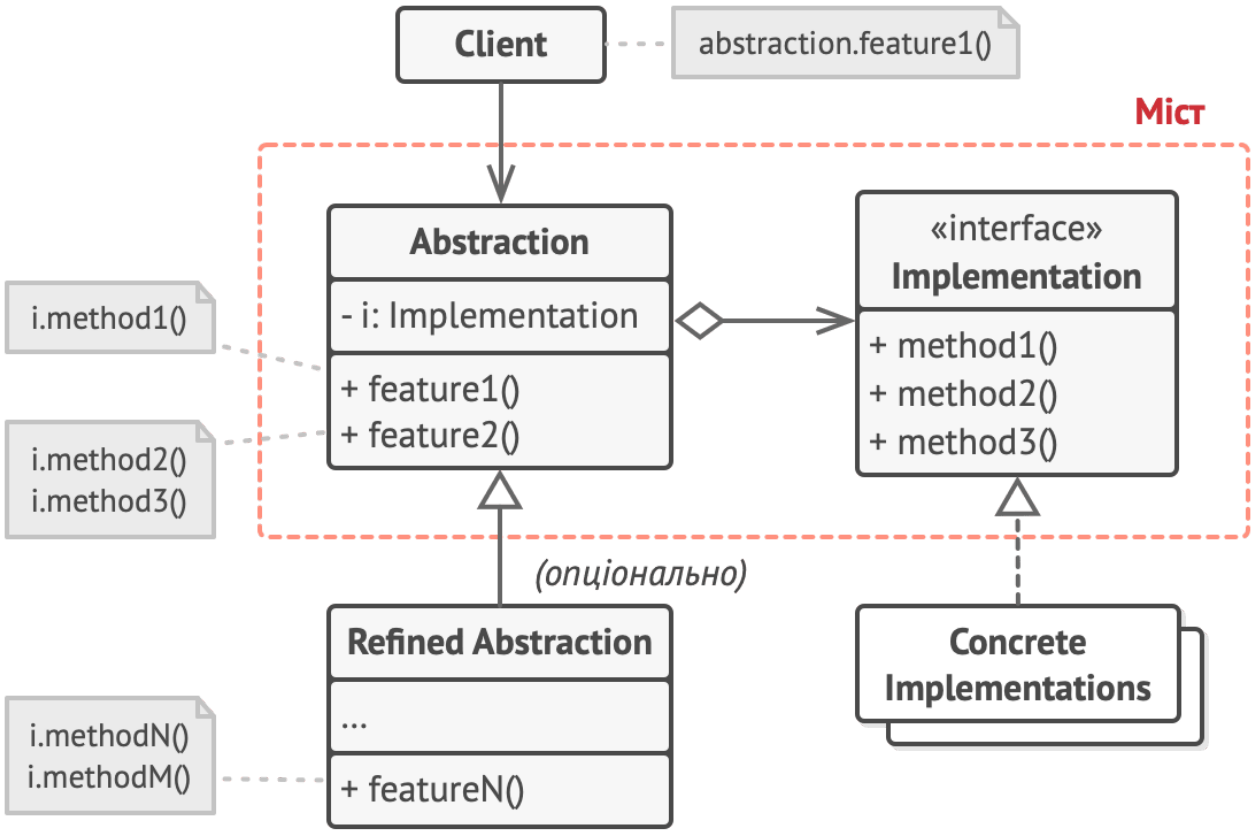
Перевірив:

Мягкий Михайло Юрійович

Київ, 2023

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

Міст — це структурний патерн проектування, який розділяє один або кілька класів на дві окремі ієрархії — абстракцію та реалізацію, дозволяючи змінювати код в одній гілці класів, незалежно від іншої.



Приклад структури реалізації та використання патерну. [Reference]

Переваги:

- Дозволяє будувати платформо-незалежні програми.
- Приховує зайві або небезпечні деталі реалізації від клієнтського коду.
- Реалізує *принцип відкритості/закритості*.

Недоліки:

- Ускладнює код програми внаслідок введення додаткових класів.

Реалізація:

1)Визначимо Абстракцію та Реалізацію

Абстракція:

```
interface ReportCreator {
   Report generateScheduledReport(LocalDateTime start, LocalDateTime end);

   Report generateReportByTime(LocalDateTime end);

   Report startReporting();

   Report stopReporting();
}
```

Це буде наша абстракція - саме її буде викликати кліент щоб зробити якіїсь звіт

Реалізація:

```
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();
}
```

Нащадкам цього класу абстракція буде делегувати формування звітів
(Трохи порушено SRP - але дуже не критично як на мене бо моніторинг в реальному часі та формування звітів будуть завжди разом - це закладено на ідейному рівні)

Приклади використання абстракції:

```
public class DefaultReportCreator implements ReportCreator {
   public Monitoring monitoring;

   public DefaultReportCreator(Monitoring monitoring) {
       this.monitoring = monitoring;
   }

   @Override
   public Report generateScheduledReport(LocalDateTime start, LocalDateTime end) {
       return monitoring.generateScheduledReport(start, end);
   }

   @Override
   public Report generateReportByTime(LocalDateTime end) {
       return monitoring.generateReportByTime(end);
```

```
    }

    @Override
    public Report startReporting() {
        return monitoring.startReporting();
    }

    @Override
    public Report stopReporting() {
        return monitoring.stopReporting();

    }
}
```

Це звичайна реалізація.

Інша реалізація:

```
public class ReportWithEmail implements ReportCreator {
    public Monitoring monitoring;
    public String email;

    public ReportWithEmail(Monitoring monitoring) {
        this.monitoring = monitoring;
    }

    @Override
    public Report generateScheduledReport(LocalDateTime start, LocalDateTime end) {
        return null;
    }

    @Override
    public Report generateReportByTime(LocalDateTime end) {
        return null;
    }

    @Override
    public Report startReporting() {
        return null;
    }

    @Override
    public Report stopReporting() {
        return null;
    }
}
```

цей клас буде надсилати звіті по пошті а не просто їх віддавати користувачу.

Таким чином ми можемо незалежно одна від одній змінювати Абстракцію на Реалізацію. Тому якщо Реалізація наприклад зміне ОС - то абстракція цього не побачить.
Також з іншого боку ми можемо змінювати поведінку абстракції та підміняти їх на ходу.

Висновок: Я реалізував патерн Bridge