Паттерн Міст

Виконав: Шкільний Владислав КС31

Паттерн "Міст" (Bridge) – це структурний шаблон проектування (тобто його основне завдання — створення повноцінної структури із класів та об'єктів), основна мета якого полягає в тому, щоб відокремити **абстракцію** від її **реалізації**. Це дозволяє змінювати як абстракцію, так і реалізацію незалежно одну від одної, що робить систему більш гнучкою і розширюваною.

Ідея паттерна

Ключова ідея паттерна полягає в тому, щоб створити дві окремі ієрархії:

1. **Абстракція** – представляє інтерфейс для користувача або основну функціональність.
2. **Реалізація** – конкретний механізм, що виконує реальну роботу. Абстракція містить посилання на реалізацію.

Завдяки цьому підходу, будь-які зміни в одній ієрархії не впливають на іншу. Це забезпечує можливість додавання нових реалізацій або абстракцій без зміни існуючого коду.

Застосування паттерна Bridge

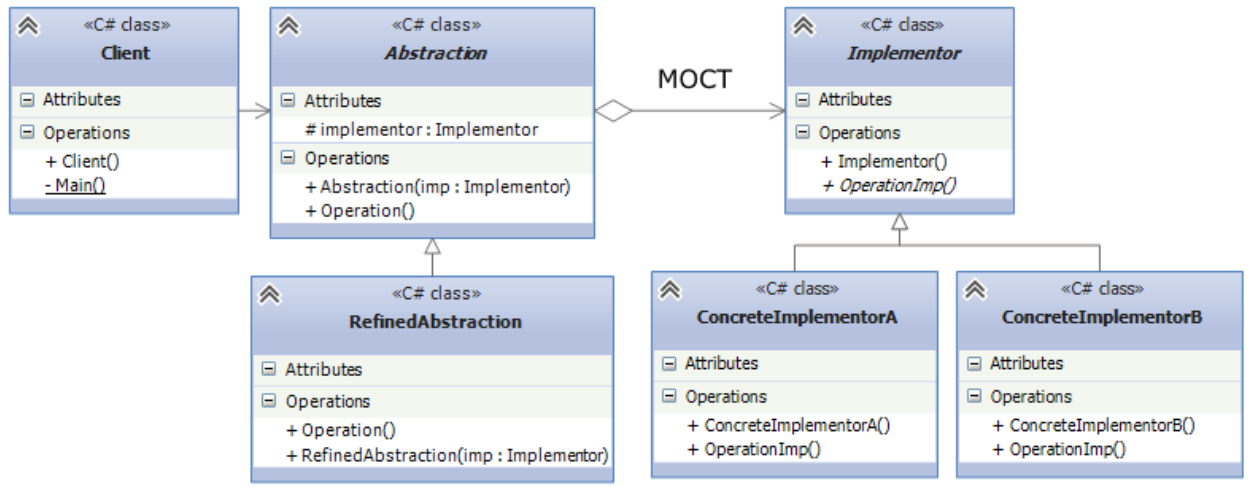
Паттерн **Bridge** корисний у випадках, коли є потреба:

- Працювати з **різними платформами** (наприклад, підтримка декількох операційних систем).
- Використовувати **різні бази даних** або інші джерела даних (SQL, NoSQL).
- Інтегруватися з **різними API** (наприклад, хмарні сервіси, соціальні мережі).

Паттерн також застосовується тоді, коли потрібно уникнути жорсткої прив’язки абстракції до її реалізації, або коли абстракцію і реалізацію необхідно змінювати незалежно одну від одної.

Структура паттерна Bridge (рис. 1)

1. **Абстракція** (Abstraction) – основний інтерфейс або клас, що представляє логіку на високому рівні. Вона містить посилання на об'єкт реалізації (Implementor).
2. **Реалізація** (Implementor) – інтерфейс або базовий клас для реалізацій.
3. **Конкретна абстракція** (Refined Abstraction) – розширений клас абстракції, що додає додаткову функціональність.
4. **Конкретна реалізація** (Concrete Implementor) – клас, що конкретно реалізує методи реалізації.



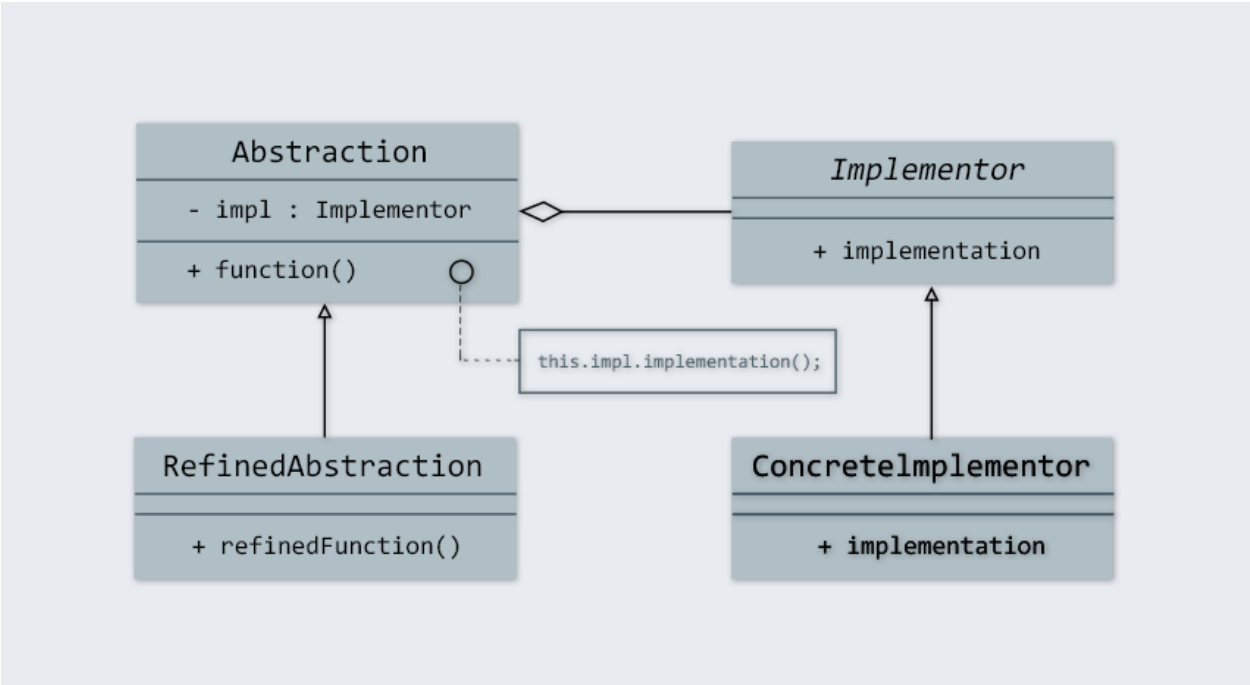


Рисунок 1 – UML подання паттерна Міст.

Зв’язок агрегації між класами Abstraction та Implementor фактично і представляє деякий міст між двома паралельними ієрархіями класів. Саме тому патерн отримав назву Міст.

Приклад реалізації паттерна Bridge на мові Ruby

Створимо простий приклад реалізації паттерна на прикладі автомобілів.

В якості реалізації буде загальний клас автомобіля **Car**.

В якості конкретної реалізації будуть види автомобілів **Mercedes, BMW, Volkswagen.**

В якості абстракції буде загальний клас кольору автомобіля **CarColor**.

В якості конкретної абстракції будуть види кольорів автомобілів **Red, Blue, Yellow.**

Кожне авто матиме метод **Info** який виводить інформацію про неї (тип авто, колір швидкість).

Реалізація та абстракція поєднуються завдяки полю **@_color** в класі реалізації **Car**, що є нашим "містом" між ієрархіями.

Така реалізація паттерну **Bridge** в структурі нашої програми дозволяє нам створювати безліч класів типів авто та додавати безліч класів видів кольору, при цьому ці зміни не будуть впливати один на одного (є незалежними).

```
# Реалізація (інтерфейс або базовий клас для різних типів автомобілів)
class Car
      def initialize(color, maxSpeed)
            @_color = color
            @_maxSpeed = maxSpeed
      end

      def Info
            raise NotImplementedError, 'This method should be overridden in a
subclass'
      end
end

# Конкретна реалізація 1 - Mercedes
class Mercedes < Car
      def Info
            "Це Mercedes кольору #{@_color.Paint}, максимальна швидкість
#{@_maxSpeed} км/год"
      end
end

# Конкретна реалізація 2 - BMW
class BMW < Car
      def Info
            "Це BMW кольору #{@_color.Paint}, максимальна швидкість
#{@_maxSpeed} км/год"
      end
end
```

```ruby
# Конкретна реалізація 3 - Volkswagen
class Volkswagen < Car
      def Info
            "Це Volkswagen кольору #{@_color.Paint}, максимальна швидкість
#{@_maxSpeed} км/год"
      end
end

# Абстракція - колір автомобіля
class CarColor
      def Paint
            raise NotImplementedError, 'This method should be overridden in a
subclass'
      end
end

# Розширена абстракція 1 - Червоний колір
class Red < CarColor
      def Paint
            'Red'
      end
end

# Розширена абстракція 2 - Синій колір
class Blue < CarColor
      def Paint
            'Blue'
      end
end

# Розширена абстракція 3 - Жовтий колір
class Yellow < CarColor
      def Paint
            'Yellow'
      end
end

# Приклад використання

redMercedes = Mercedes.new(Red.new, 160)
blueVolkswagen = Volkswagen.new(Blue.new, 150)
yellowBMW = BMW.new(Yellow.new, 170)

puts redMercedes.Info
# Виведе: "Це Mercedes кольору Red, максимальна швидкість 160 км/год"
puts blueVolkswagen.Info
# Виведе: "Це Volkswagen кольору Blue, максимальна швидкість 150 км/год"
puts yellowBMW.Info
# Виведе: "Це BMW кольору Yellow, максимальна швидкість 170 км/год"
```

Переваги паттерна Bridge:

1. **Незалежність абстракції та реалізації** – Легко додавати нові реалізації або абстракції без змін у наявних класах.
2. **Гнучкість** – Можна вільно комбінувати різні абстракції з різними реалізаціями.
3. **Зменшення дублювання коду** – Одна і та ж логіка може використовуватися в різних контекстах.

Ознаки застосування паттерна:

- Чітко виражене розділення між класами, що відповідають за управління і виконання, коли керуючі об'єкти делегують функції виконання іншим класам (реалізаціям).

Висновки

Паттерн "Міст" (Bridge) дозволяє відокремити абстракцію від реалізації, що робить систему більш гнучкою і легко розширюваною. Його використання доцільне, коли потрібно забезпечити незалежну змінюваність двох або більше ієрархій класів — наприклад, коли є кілька варіантів реалізації або різні аспекти функціональності. Це зменшує зв'язність коду та запобігає дублюванню, оскільки кожна ієрархія може розвиватися окремо. Такий підхід особливо корисний при роботі з різними платформами, типами пристроїв чи API, командній розробці, забезпечуючи легку інтеграцію нових компонентів без порушення існуючої архітектури.

Джерела:

https://abitap.com/4-6-patern-mist-bridge/

https://javarush.com/ua/groups/posts/uk.2570.pattern-proektuvannja-mst-bridge-pattern

https://refactoring.guru/uk/design-patterns/bridge/ruby/example