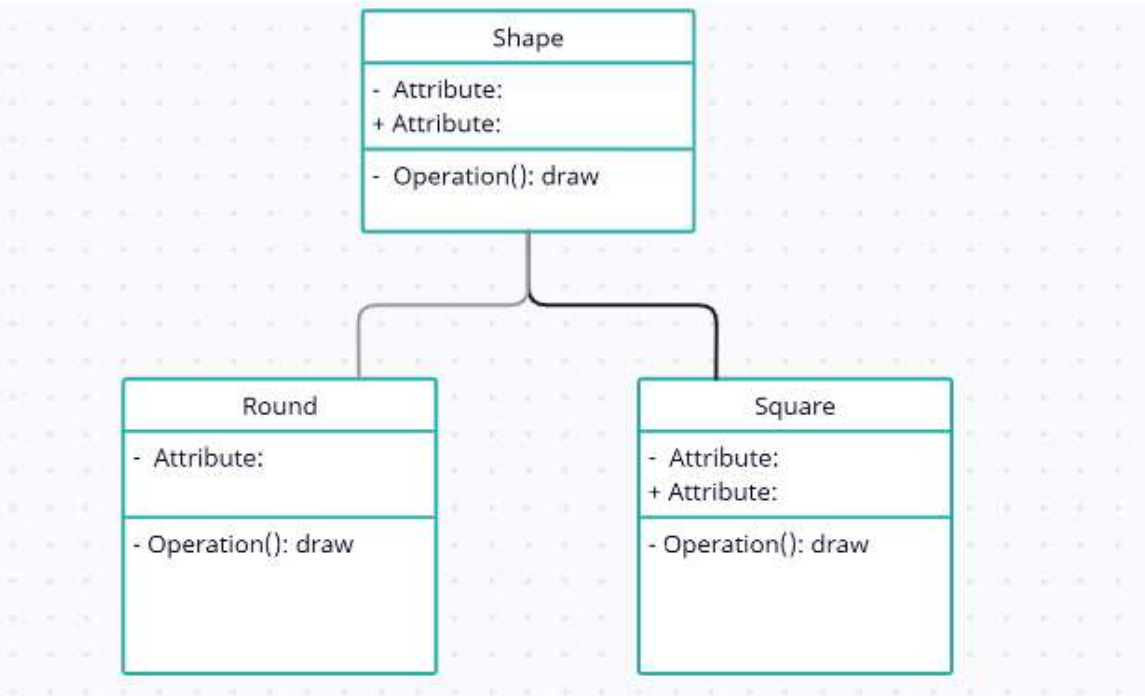
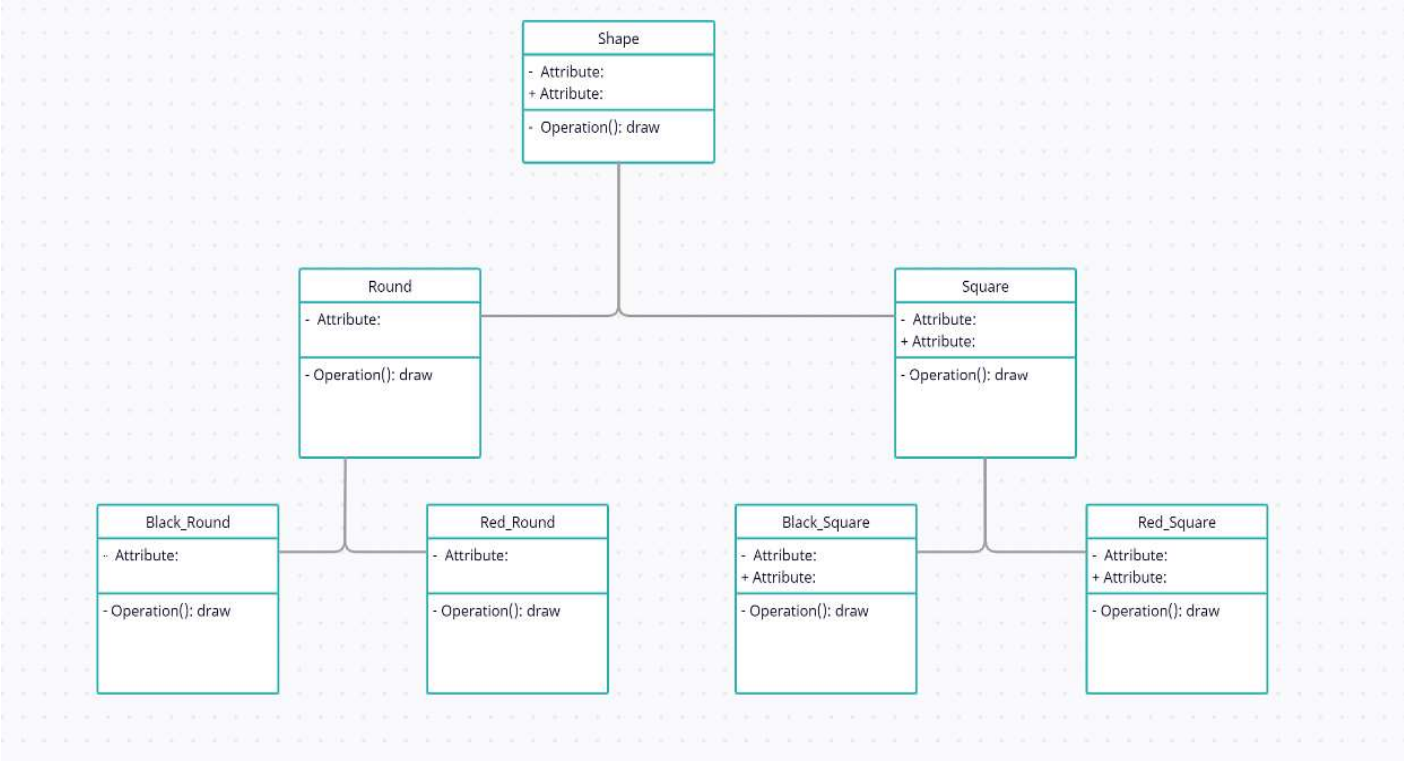
Bridge Pattern

- Проблема: Ми хочемо створити за допомогою ООП застосунок, де будуть такі класи як Фігура (абстрактний клас, для прикладу), який будуть наслідувати класи Коло та Квадрат. Наче все непогано, поки що... Якщо нам не треба буде маштабувати програму далі, то все взагалі чудово. Але в житті нічого просто не буває. Та ми раптом хочемо додати кольорів у наше життя, а саме до наших класів.



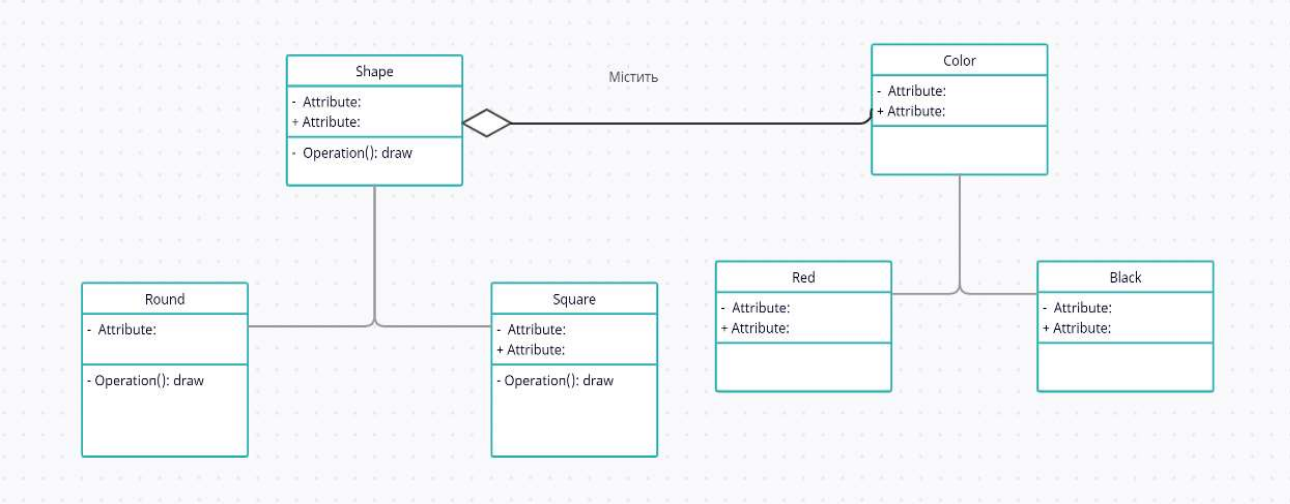
Bridge Pattern

- Отож, ми створили ще 4 класи: 2 кола та 2 квадратики, чорні та червоні. І тут вже починає бути зрозуміла наша проблема. Бо кольорів може бути не 2, а 20. А що якщо ми додамо ще одну фігуру, коли в нас вже реалізовано більше 20 кольорів? Це буде надзвичайна велика к-ть класів при масшабуванні та безперечно є проблемою. То ж, що ми можемо зробити?



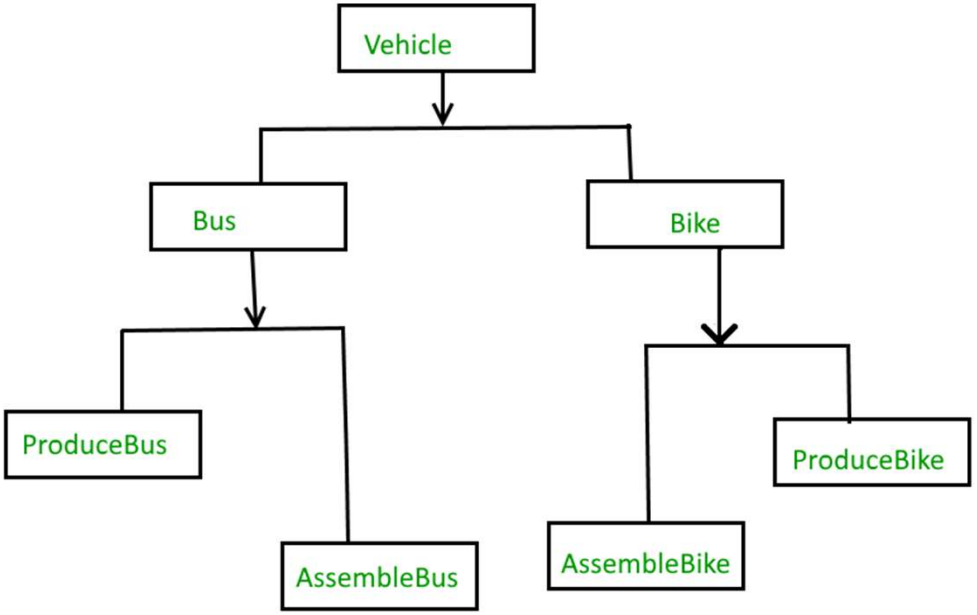
Bridge Pattern

- Гарне рішення – паттерн Міст. 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.



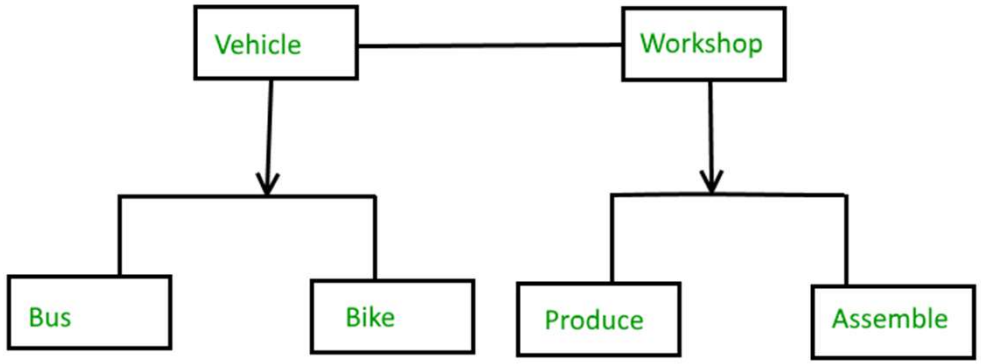
Bridge Pattern

- Також можна навести приклад з транспортними засобами. Напраклад як зображено на діаграмі. Цей приклад більш точний так як більш пояснює суть і проблему.



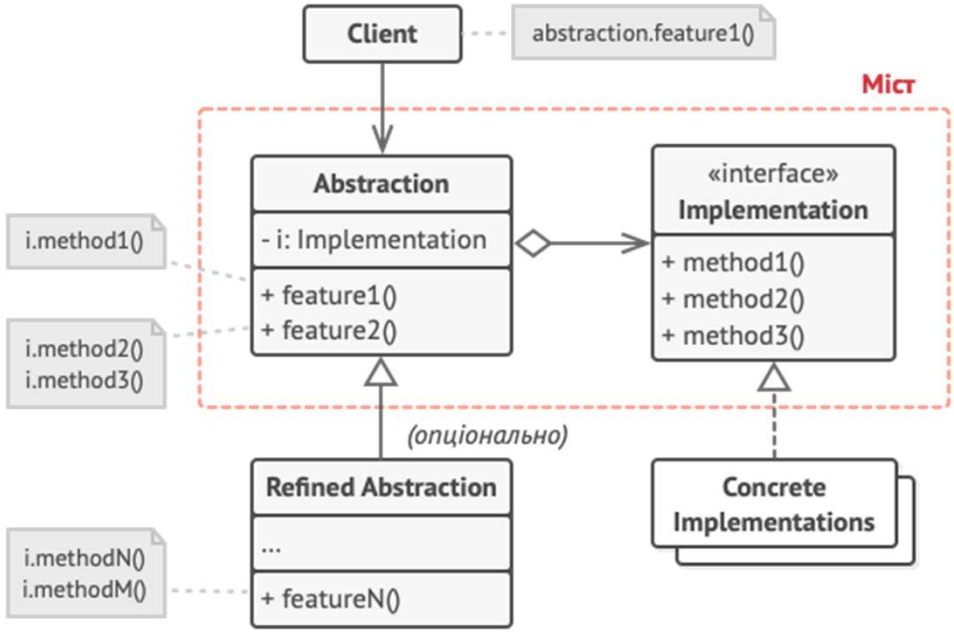
Bridge Pattern

- Тож рішення за допомогою патерном Міст буде наступним.



Структура

1. **Абстракція** містить керуючу логіку. Код абстракції делегує реальну роботу пов'язаному об'єктові реалізації.
2. **Реалізація** описує загальний інтерфейс для всіх реалізацій. Всі методи, які тут описані, будуть доступні з класу абстракції та його підкласів.Інтерфейси абстракції та реалізації можуть або збігатися, або бути абсолютно різними. Проте, зазвичай в реалізації живуть базові операції, на яких будуються складні операції абстракції.
3. **Конкретні реалізації** містять платформо-залежний код.
4. **Розширені абстракції** містять різні варіації керуючої логіки. Як і батьківский клас, працює з реалізаціями тільки через загальний інтерфейс реалізацій.
5. **Клієнт** працює тільки з об'єктами абстракції. Не рахуючи початкового зв'язування абстракції з однією із реалізацій, клієнтський код не має прямого доступу до об'єктів реалізації.

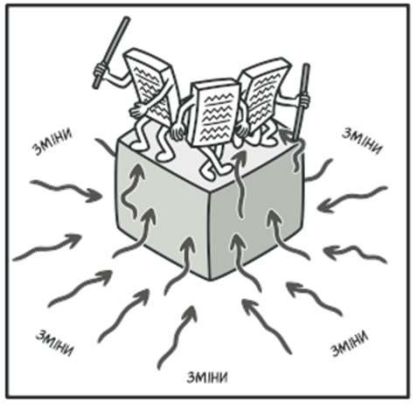


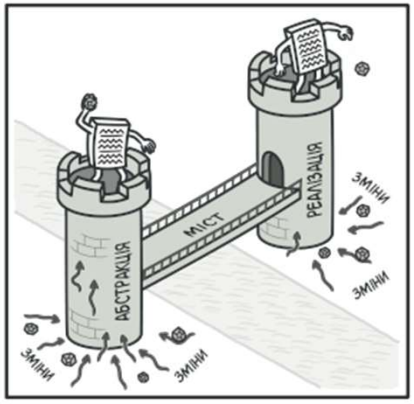
Bridge Pattern

Якщо говорити про реальні програми, то абстракцією може виступати графічний інтерфейс програми (GUI), а реалізацією — низькорівневий код операційної системи (API), до якого графічний інтерфейс звертається, реагуючи на дії користувача.

Ви можете розвивати програму у двох різних напрямках:

- мати кілька різних GUI (наприклад, для звичайних користувачів та адміністраторів).
- підтримувати багато видів API (наприклад, працювати під Windows, Linux і macOS).
- Така програма може виглядати як один великий клубок коду, в якому змішано умовні оператори рівнів GUI та API.





- Ви можете спробувати структурувати цей хаос, створивши для кожної з варіацій інтерфейсу-платформи свої підкласи. Але такий підхід призведе до зростання класів комбінацій, і з кожною новою платформою їх буде все більше й більше.
- Ми можемо вирішити цю проблему, застосувавши Міст. Патерн пропонує розплутати цей код, розділивши його на дві частини:
- Абстракцію: рівень графічного інтерфейсу програми.
- Реалізацію: рівень взаємодії з операційною системою.

Bridge Pattern

Застосування

Якщо ви хочете розділити монолітний клас, який містить кілька різних реалізацій якої-небудь функціональності (наприклад, якщо клас може працювати з різними системами баз даних).

- Чим більший клас, тим важче розібратись у його коді, і тим більше це розтягує час розробки. Крім того, зміни, що вносяться в одну з реалізацій, призводять до редагування всього класу, що може викликати появу несподіваних помилок у коді.
- Міст дозволяє розділити монолітний клас на кілька окремих ієрархій. Після цього ви можете змінювати код в одній гілці класів незалежно від іншої. Це спрощує роботу над кодом і зменшує ймовірність внесення помилок.

Bridge Pattern

Застосування

- *Якщо клас потрібно розширювати в двох незалежних площинах.*

Міст пропонує виділити одну з таких площин в окрему ієрархію класів, зберігаючи посилання на один з її об'єктів у початковому класі.

Якщо ви хочете мати можливість змінювати реалізацію під час виконання програми.

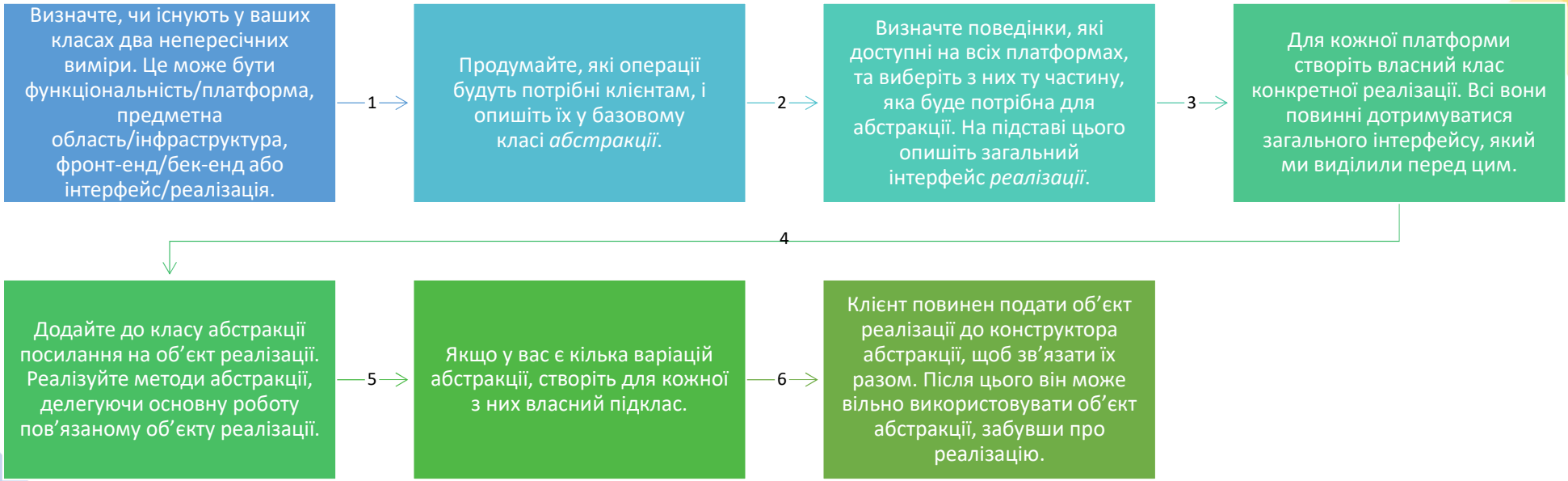
Міст дозволяє замінювати реалізацію навіть під час виконання програми, оскільки конкретна реалізація не «зашита» в клас абстракції.

Bridge Pattern

Взаємодія з іншими патернами

Паттерн Bridge дозволяє відокремити абстракцію від реалізації, тобто він дозволяє створювати дві різні ієрархії класів та зв'язувати їх через міст (Bridge). Це дозволяє змінювати реалізацію без зміни інтерфейсу абстракції. Паттерн Bridge може бути поєднаний з іншими паттернами, такими як Factory Method або Abstract Factory, щоб створити об'єкти різних класів, які взаємодіють між собою через міст.

Кроки реалізації



Переваги та недоліки

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 (більш скалдна архітектура, так як ми розділяємо абстракції та пов’язуємо їх з реалізаціями) .

Використання цього патерну

1. Microsoft використовує паттерн Bridge в операційній системі Windows, для забезпечення різних рівнів взаємодії між драйверами та апаратними засобами.
2. Oracle використовує паттерн Bridge в своїх програмних продуктах для забезпечення взаємодії між клієнтською та серверною частинами.