Адаптер

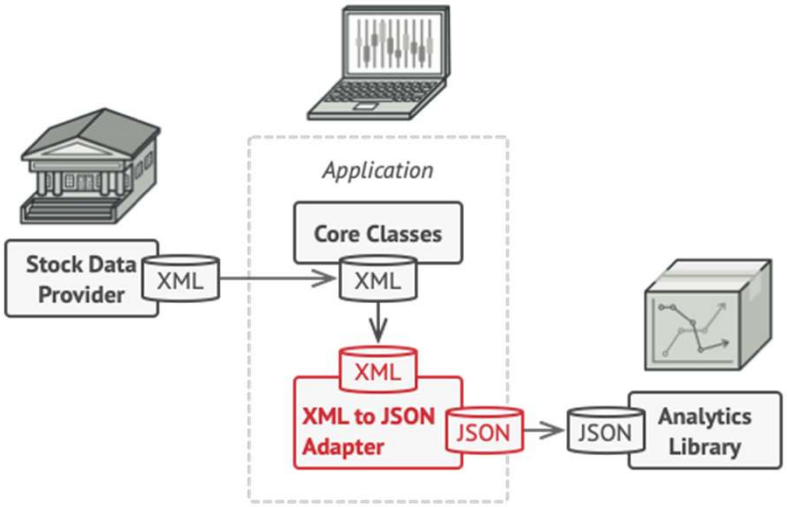
«Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.»

Gang of Four

Не ці)

Адаптери у реальному житті

Мотивація

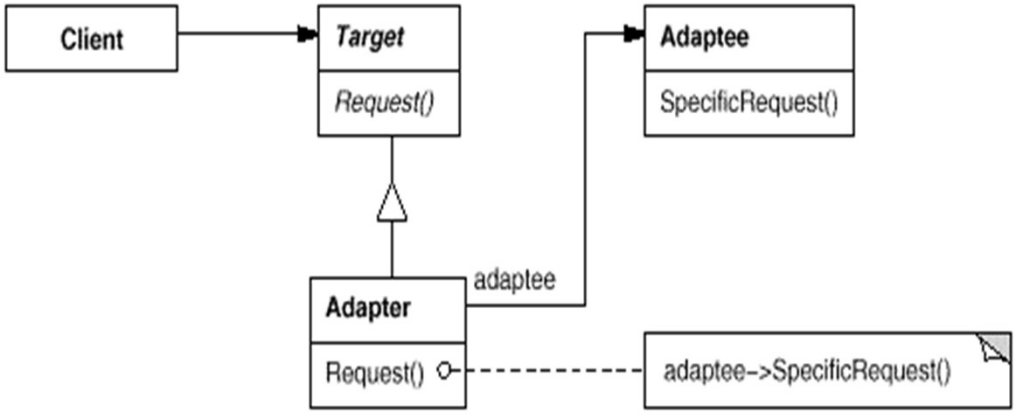


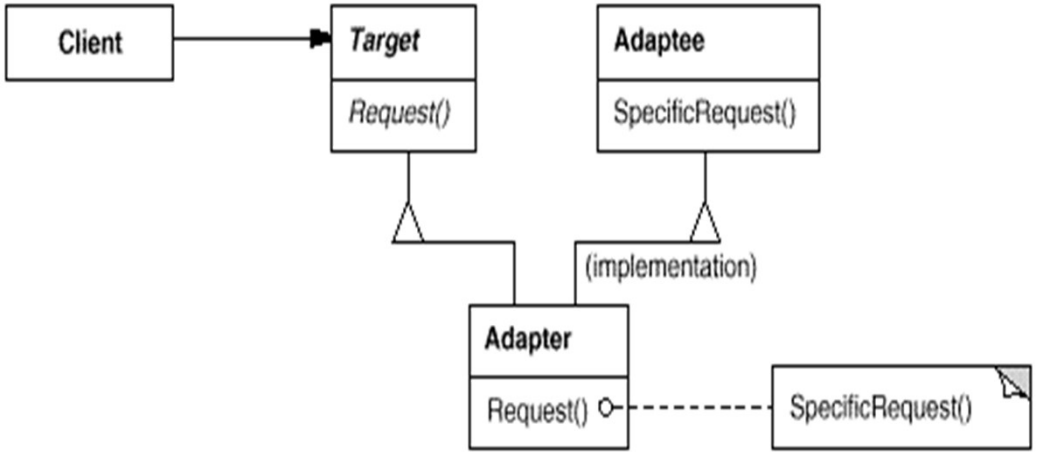
Проблема: потрібно забезпечити взаємодію частин програми, які використовують несумісні *об'єкти* для своєї роботи.

Рішення – створення проміжних об'єктів, котрі забезпечать передачу даних від однієї частини програми до іншої у потрібному форматі.

Реалізація

- Виділити загальний клієнтський інтерфейс, який необхідно реалізувати
- Створити клас Adapter, що реалізуватиме визначений інтерфейс
- Створити конструктор, який на вхід буде приймати об’єкт з іншої частини коду або унаслідувати той клас
- Використовувати методи іншого класу при реалізації інтерфейсу





Невже все настільки просто?

Що робити з бібліотечним класом?

Зберігати вказівник/відсилку?

```
class Adapter : public IClientObject {
private:

const SomeOtherLibClass& _target;

public:
Adapter(const SomeOtherLibClass&);

void action() const override;
};
```

Наслідувати обидва класи?

```
class ClassAdapter : private
SomeOtherLibClass, public IClientObject
{
public:

void action() const override;

};
```

Що робити з бібліотечним класом?

Агрегація:

- Переваги
  - Підтримується на усіх об’єктно-орієнтованих мовах програмування
  - Дозволяє працювати не лише із клієнтським об’єктом, а й із усіма його нащадками
- Недоліки
  - Ускладнюється процес реалізації інтерфейсу іншого класу, оскільки немає доступу до приватних полів

Множинне наслідування

- Переваги
  - Легша реалізація необхідного інтерфейсу за рахунок доступу до приватних полів і наслідування
  - Легко перетворити на двосторонній адаптер
- Недоліки
  - Множинне наслідування підтримується не на усіх мовах програмування
  - Необхідність ініціалізувати поля обох класів, що може спричинити дублювання данних

А які іще адаптери бувають?

Pluggable adapter – адаптер, який забезпечує реалізацію інтерфейсів різних класів бібліотеки.

```
class PluggableAdapter : public IClientObject {
private:
std::function<void()> _requiredMethod;
public:

PluggableAdapter(const SomeOtherLibClass&);
PluggableAdapter(const JustForPluggableAdapter&);

void action() const override {
this->_requiredMethod();
}

};
```

Two-way adapter(двосторонній адаптер) – адаптер, що містить реалізацію інтерфейсів як класу користувача, так і класу бібліотеки.

```
class ClassAdapter : public
SomeOtherLibClass, public
IClientObject {
public:

void action() const override;

};
```

Загальні переваги та недоліки цього шаблону проектування

Переваги:

- Забезпечення принципу єдиної відповідальності
- Забезпечення принципу відкритості і закритості
- Забезпечує взаємодію модулів, які мають різні інтерфейси

Недоліки:

- Додаткова сутність ускладнює читання коду і роботу програми

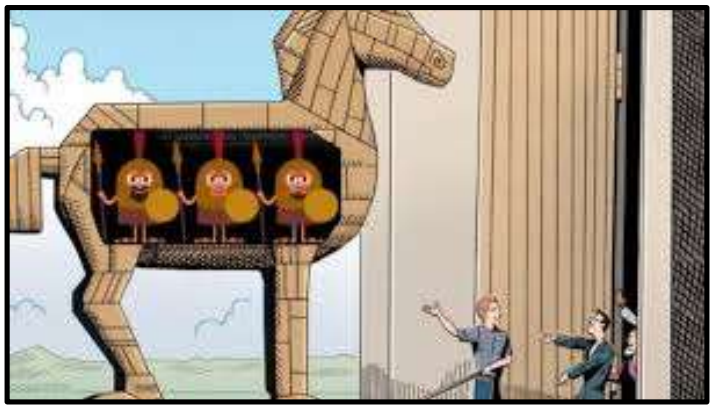
Wrapper vs. Wrapper

Декоратор:

- Підтримує рекурсивну композицію, на відміну від Адаптера

Адаптер:

- Змінює інтерфейс існуючого об’єкта, тоді як Декоратор лише додає нові методи



Адаптер і Замісник

Замісник:

- Повторює весь інтерфейс об’єкта, або його частину

Адаптер:

- Повністю змінює інтерфейс об’єкта

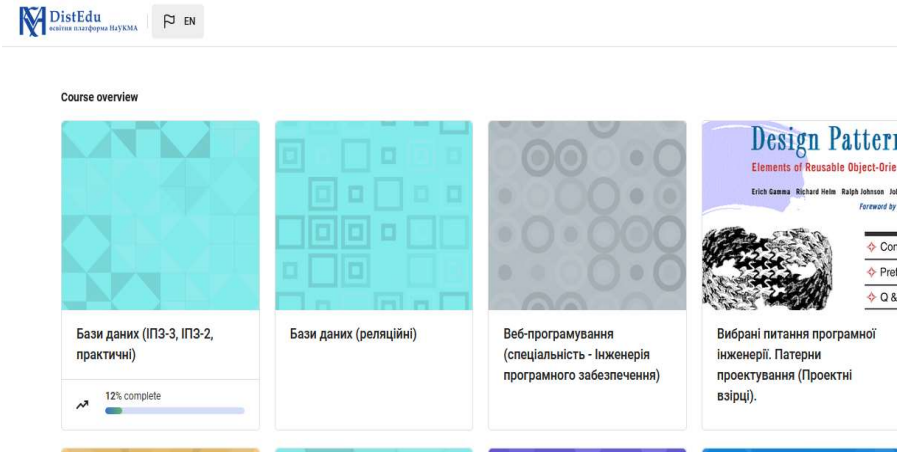
Адаптер і Фасад

Фасад:

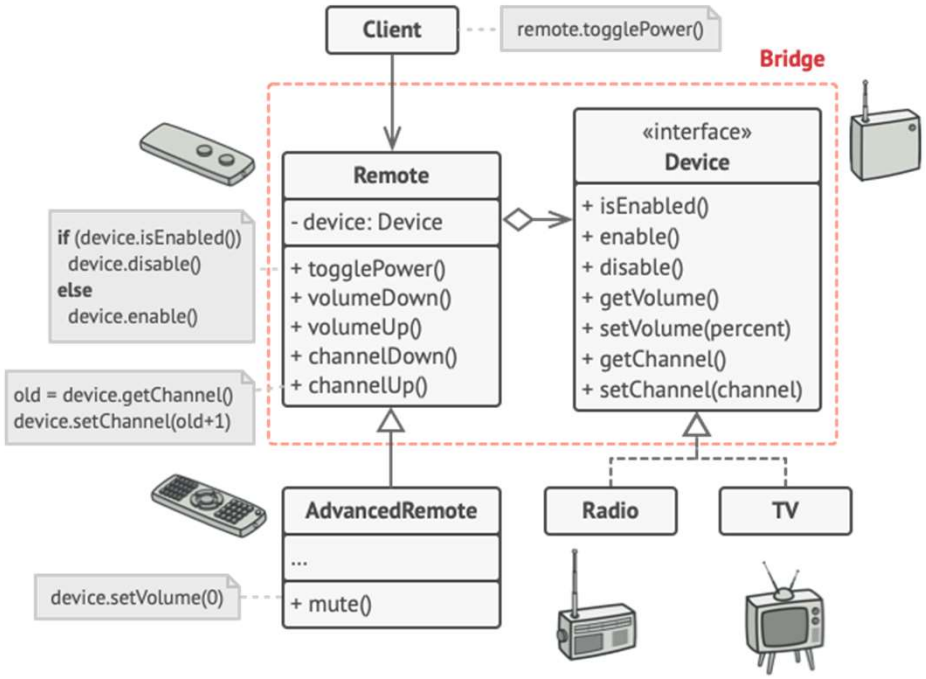
- Змінює інтерфейс *класу*
- Взаємодіє не лише із самим об’єктом класу, а й з усіма нащадками

Адаптер:

- Змінює інтерфейс *об’єкта*
- Змінює інтерфейс на етапі виконання програми



Адаптер і Міст



- Схожа структура і призначення.
- Міст не міняє інтерфейсів, а лише розділює абстракції

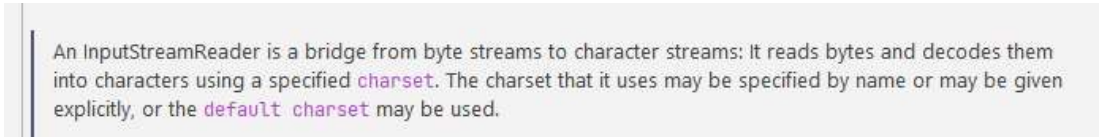
Ну і чим Remote не адаптер?

Коли треба використовувати Адаптер?

- Коли застосувати існуючий клас, але його інтерфейс є незручним для цього
- Коли треба створити клас, який буде використовуватись у подальшій роботі і який повинен взаємодіяти із не пов’язаними з ним класами
- Коли потрібно створити декілька класів, які реалізовуватимуть деякий інтерфейс, але не будуть наслідувати батьківський клас

Використання Адаптера у стандартних бібліотеках

- Колекції у мові Java використовують цей паттерн:
  - public static <T> List<T> asList(T... a) - повертає ArrayList із заданими елементами, який і виконує роль адаптера для масивів
  - public static <T> ArrayList<T> list(Enumration<T> e)
  - InputStreamReader та InputStreamWriter є адаптером для байтових і стрчкових потоків введення/виведення.



Ніякий він не bridge. Це адаптер