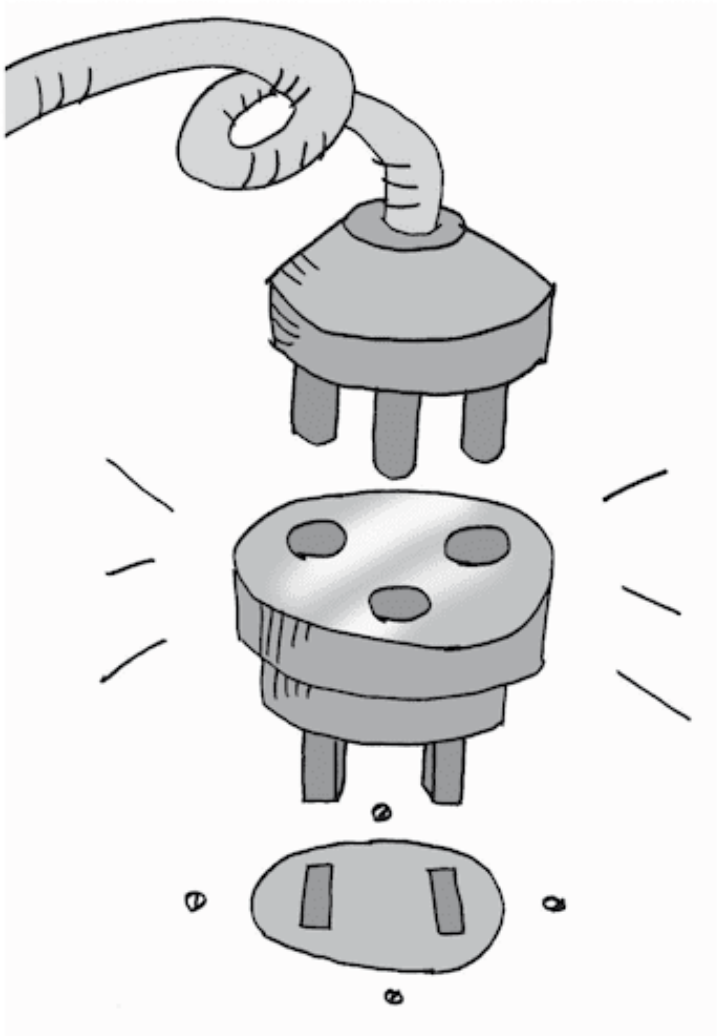
Adapter Pattern

Reuse through Adaptation



Adapter Pattern

Adapter Pattern

- **Power Adapter** → Converts U.S. 110V to EU 220V
- **HDMI to VGA** → Connects new laptop to old projector

Adapt one interface to another, making them compatible.

The Problem

- Old class: `Banner("Hello").show_with_paren()`
- New interface: `Print("Hello").print_weak()`
- Goal: Reuse old code (Banner) without rewriting to match the new interface (Print).

The challenge: how to make **incompatible interfaces** work together and **reuse existing code** without modifying it?

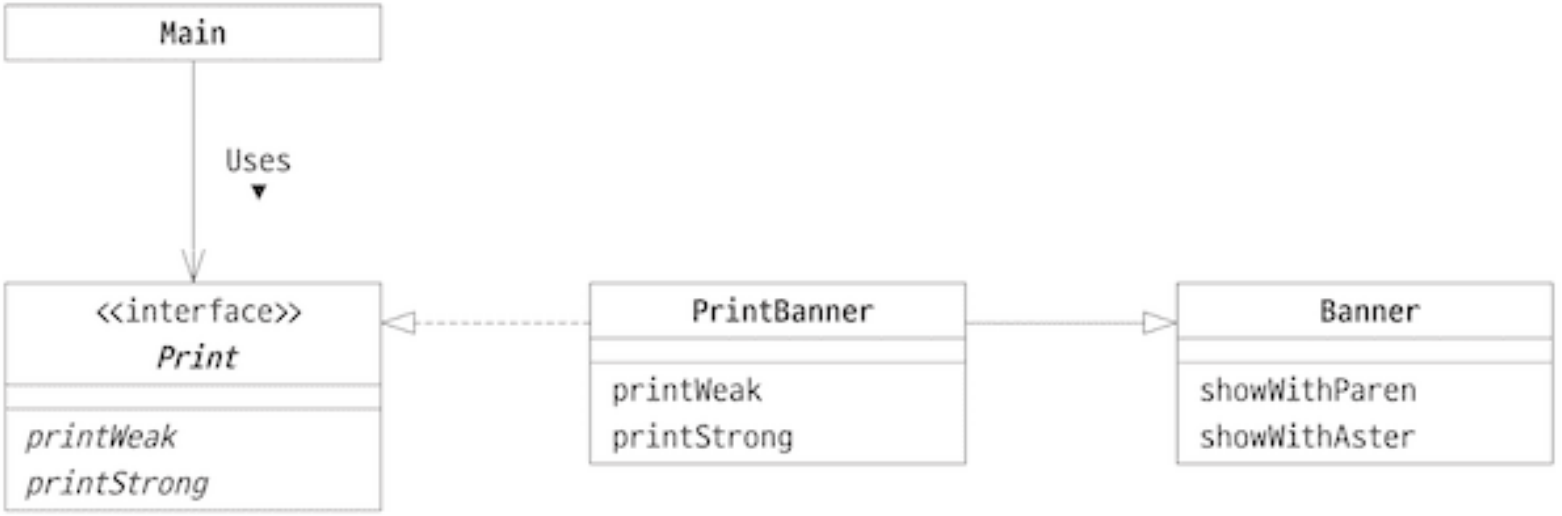
The Solution

We use the Adapter DP.

```
# We want to use the new Print interface
print_obj = PrintBanner(Banner, Print)  # Adapter
print_obj.print_weak()  # New API
```

- The **PrintBanner** is an *adapter* that allows us to use the new *print_weak* interface with the old **Banner** class.
- We can *reuse* existing code through *adaptation*.
    - For this, we should use the same *interface* with the new class, but reuse the old *code/implementation* with the old class.

The Design



- We have two interfaces: Banner (old) and Print (new).
- We need the Adator PrintBanner that knows both interfaces.
    - In this example, the adapter uses the **new interface** and old code through **inheritance**.

Example

Step 1: The Players

In this design, we have players:

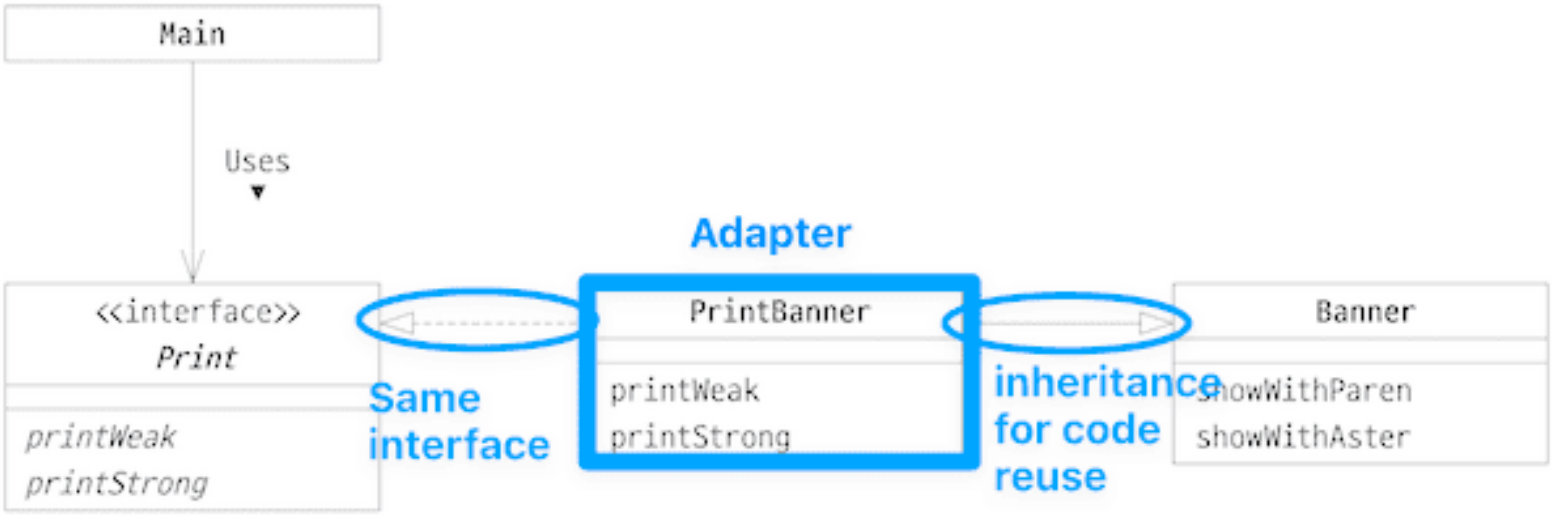
- *Target* (new interface - Print)
- **Adaptee** (old class - Banner)
- **Adapter** (bridges Target and Adaptee - PrintBanner)

The client uses the new (not old) interface with old code.

- **Client** uses the new interface

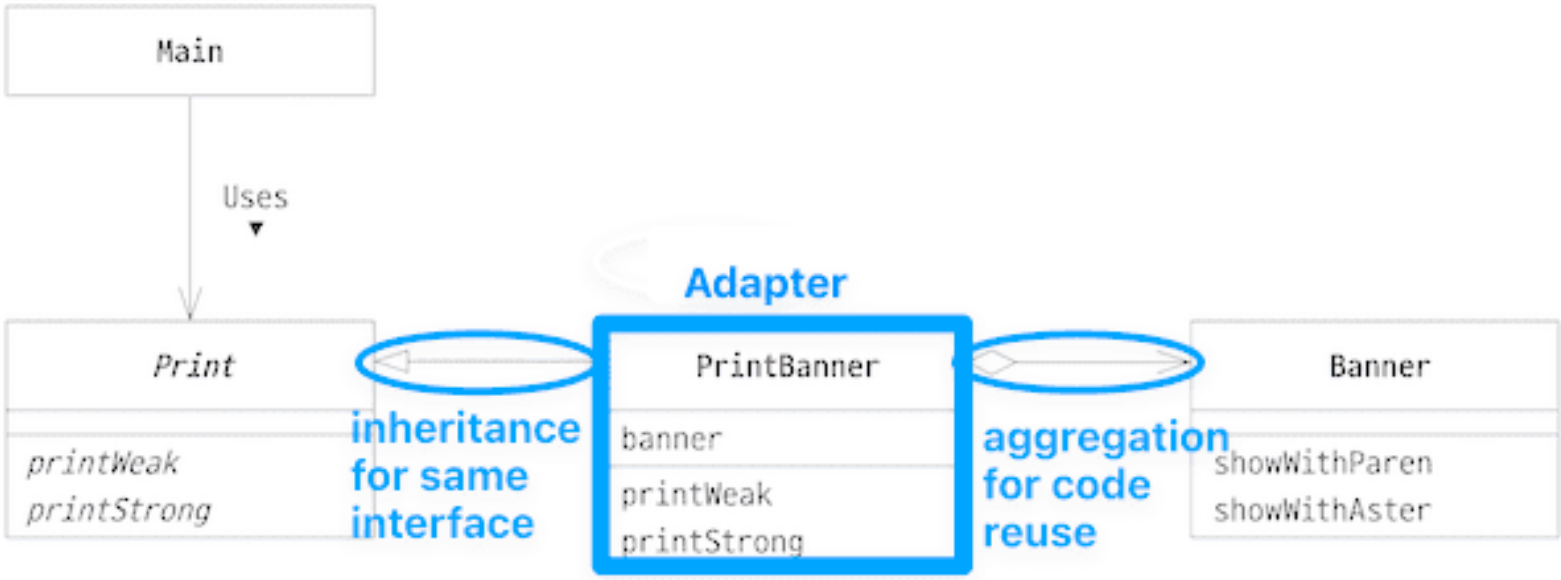
Step 2: Two Implementations

1. Inheritance-based Adapter



- **Adapter** uses the same interface of the **Target**, and use the old code from **Adaptee**.
- **Adapter** can inherit from both *Target* and **Adaptee**.

2. Delegation-based Adapter



- **Adapter** implements *Target* for reusing the interface.
- **Adapter** aggregates (owns) the **Adaptee** for code reuse.

Code

- Main Method
- Banner (Adaptee)
- Print Interface (Target)
- PrintBanner (Adapter)

Main Method

```
from print_banner import PrintBanner

def main():
    print("=== Adapter Pattern Example ===\n")

    p = PrintBanner("Hello")

    print("Client code using Print interface:")
    p.print_weak()   # Adapted to Banner's show_with_paren()
    p.print_strong() # Adapted to Banner's show_with_aster()

if __name__ == "__main__":
    main()
```

Step 1: Create adapter instance

```
p = PrintBanner("Hello")
```

- **PrintBanner** is the adapter that implements *Print* interface.
- It internally uses **Banner** functionality.

Step 2: Use the new interface

```
p.print_weak()   # Uses Banner's show_with_paren()
p.print_strong() # Uses Banner's show_with_aster()
```

- Client code uses the new *Print* interface methods.
- The adapter translates these calls to the old **Banner** methods.

Old API (Adaptee)

```
class Banner:
    def __init__(self, string: str):
        self._string = string

    def show_with_paren(self):
        print(f"({self._string})")

    def show_with_aster(self):
        print(f"*{self._string}*")
```

New API (Target)

```
from abc import ABC, abstractmethod

class Print(ABC):
    def print_weak(self): pass
    def print_strong(self): pass
```

Adapter Implementation

```python
from banner import Banner
from print_interface import Print

class PrintBanner(Banner, Print):
    def __init__(self, string: str):
        super().__init__(string)

    def print_weak(self):
        self.show_with_paren()  # Adapt to old method

    def print_strong(self):
        self.show_with_aster()  # Adapt to old method
```

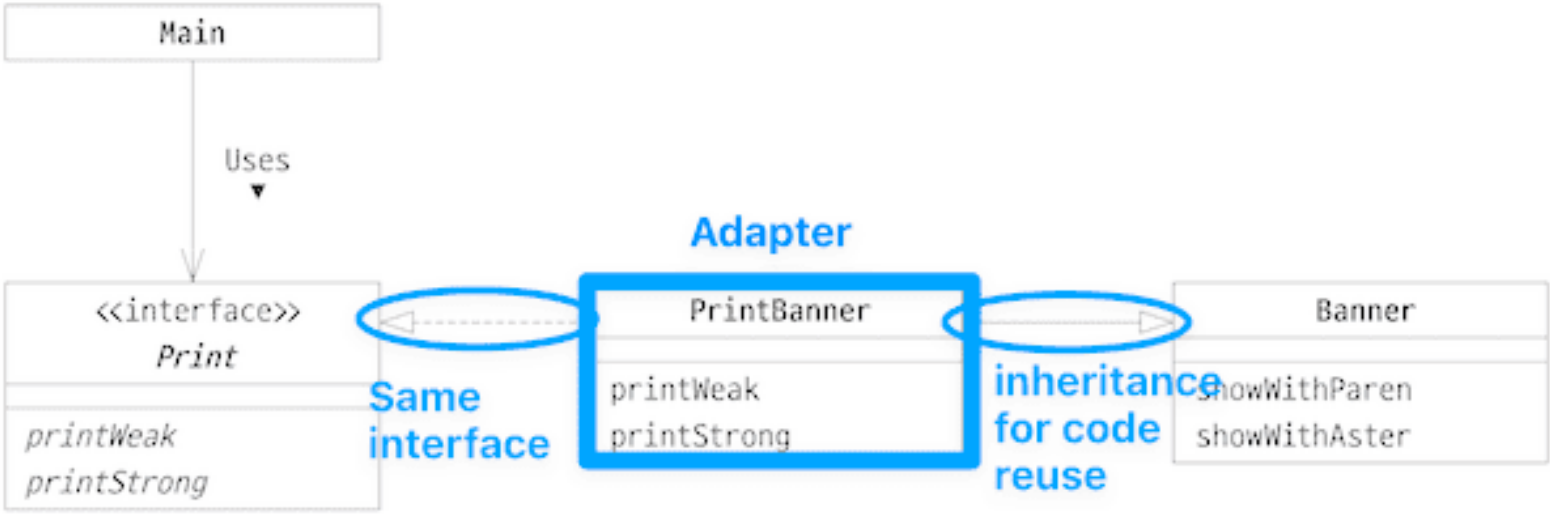
Delegation-based Alternative

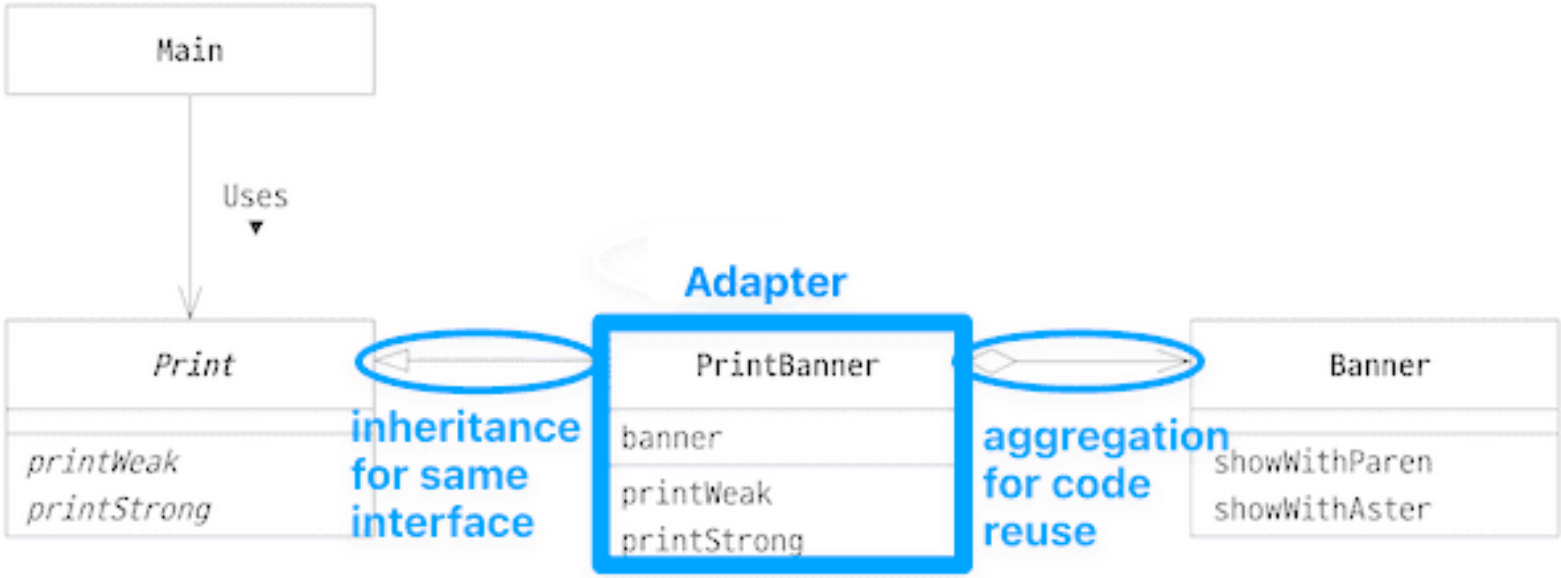
```
class PrintBanner(Print):
    def __init__(self, string: str):
        self._banner = Banner(string)  # Aggregation

    def print_weak(self):
        self._banner.show_with_paren()

    def print_strong(self):
        self._banner.show_with_aster()
```

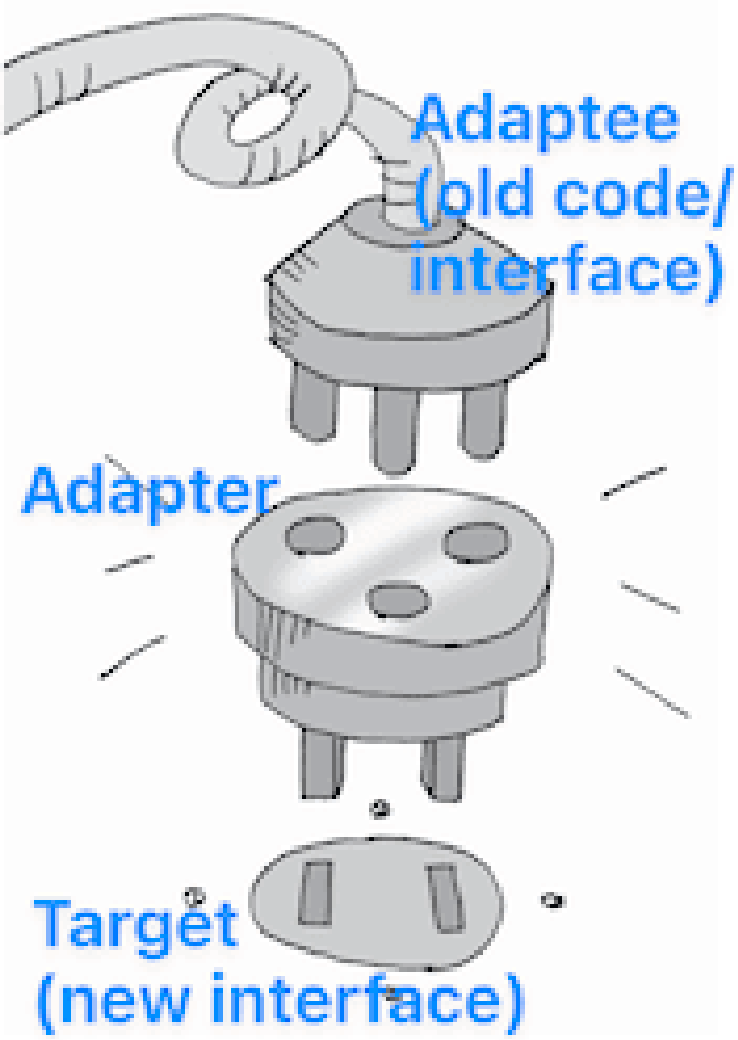
Discussion

- The Adapter pattern can be used even when we do not have the source code: we just call the old method to implement the new interface.
- The Adapter pattern is used for supporting legacy code by making the legacy code the Target and the new code the Adaptee.
- When the Adaptee and Target are too different, we can't use this pattern.

UML


Main
Uses
Adapter
<<interface>>
Print
printWeak
printStrong
Same
interface
PrintBanner
printWeak
printStrong
inheritance
for code
reuse
Banner
showWithParen
showWithAster


Main
Uses
Adapter
Print
printWeak
printStrong
inheritance
for same
interface
PrintBanner
banner
printWeak
printStrong
aggregation
for code
reuse
Banner
showWithParen
showWithAster


Adaptee
(old code/
interface)
Adapter
Target
(new interface)