Structural Design Pattern

Hung Tran

Fpt software

October 27, 2021

Outline

1. Structural Pattern Overview

2. Adapter pattern

Structural Pattern Overview

How classes and objects are composed fo form larger structure.

- **Adapter**: Convert the interface of a class into another interface.
- **Bridge**: Decouple an abstraction from its implementation.
- **Composite**: Compose objects into tree structure.
- **Decorator**: Attach additional responsibilities to an object dynamically.
- **Facade**: Provide a unified interface to a set of interfaces.
- **Flyweight**: Use sharing to support large numbers of fine-grained objects efficiently.
- **Proxy**: Provide a surrogate or placeholder for another object to control access to it.

Why we need Adapter Design Pattern?

Class wrapper

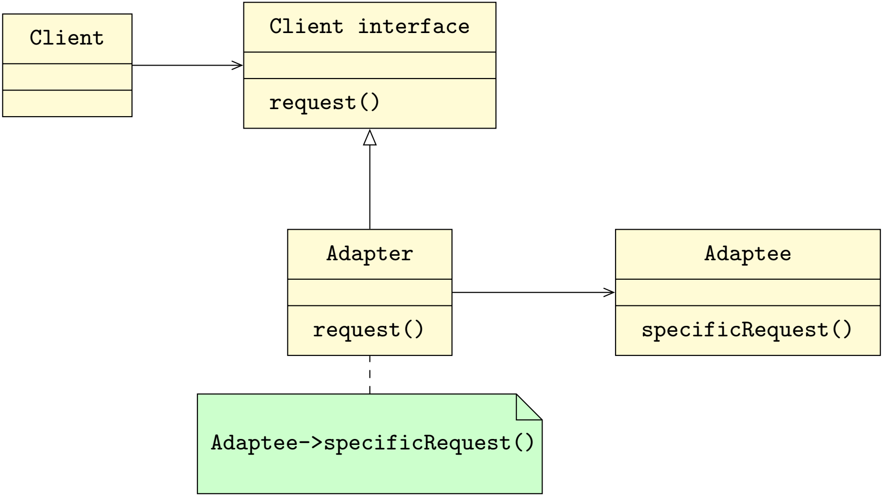
The Intent of Adapter Design Pattern

Convert the interface of a class into another interface clients expect. Adapter lets classes work together that could not otherwise because of incompatible interfaces.

How to implement Adapter Design Pattern?

-
-
-

Structure of Adapter Pattern: Object adapter



Basic implementation: Rectangle class

rectangle.h

```cpp
#ifndef _RECTANGLE_H_
#define _RECTANGLE_H_
class Rectangle {
  int a;
  int b;
public:
  Rectangle();
  Rectangle(int a, int b);
  virtual int width() const;
  virtual int height() const;
  virtual int area() const;
};

#endif // _RECTANGLE_H_
```

rectangle.cpp

```cpp
#include "rectangle.h"

Rectangle::Rectangle() : a{0}, b{0} {
}

Rectangle::Rectangle(int a, int b) : a{a}, b{b} {
}

int Rectangle::width() const {
  return a;
}

int Rectangle::height() const {
  return b;
}

int Rectangle::area() const {
  return a*b;
}
```

Basic implementation: Square class

square.h

```cpp
#ifndef _SQUARE_H_
#define _SQUARE_H_
class Square {
  int a;
public:
  Square();
  Square(int a);
  int getEdge() const;
  int area() const;
};
#endif // SQUARE_H_
```

square.cpp

```cpp
#include "square.h"

Square::Square() : a{0} {
}

Square::Square(int a) : a{a} {
}

int Square::getEdge() const {
  return a;
}

int Square::area() const{
  return a*a;
}
```

Basic implementation: Adapter class

adapter.h

```cpp
#ifndef _ADAPTER_H_
#define _ADAPTER_H_
#include "rectangle.h"
#include "square.h"
class Adapter : public Rectangle{
  Square& s;
public:
  Adapter(Square& s);
  int width() const override;
  int height() const override;
  int area() const override;
};
#endif // _ADAPTER_H_
```

adapter.cpp

```cpp
#include "adapter.h"

Adapter::Adapter(Square& s) : s{s}{
}

int Adapter::width() const {
  return s.getEdge();
}

int Adapter::height() const {
  return s.getEdge();
}

int Adapter::area() const {
  return s.area();
}
```

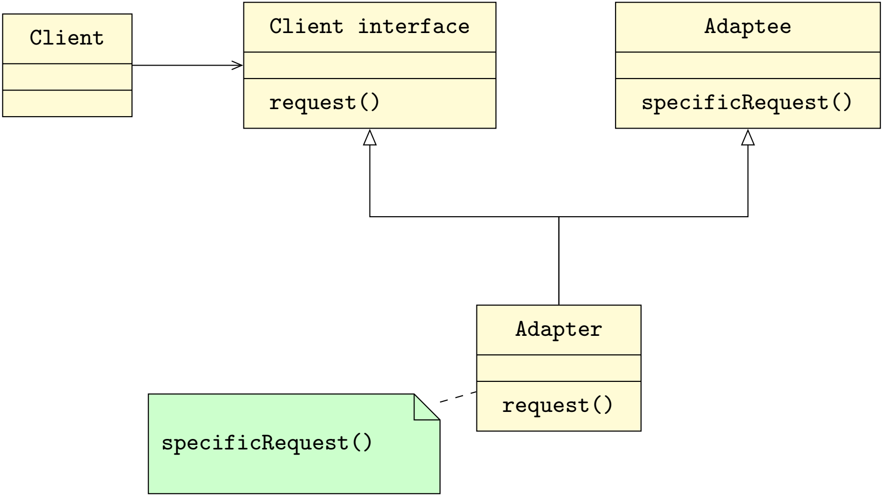
Basic implementation: client code

main.cpp

```cpp
#include "adapter.h"
#include <iostream>

// client code
void doSomething(Rectangle& r) {
  int w = r.width();
  int h = r.height();

  std::cout << "width: " << w << "\nheight: " << h << std::endl;
}

int main() {
  Square s(2);
  Rectangle r(2,3);
  doSomething(r);
  //doSomething(s);
  Adapter a(s);
  doSomething(a);
  return 0;
}
```

- Object adapter

Structure of Adapter Pattern: Class adapter



Basic implementation: Rectangle class

rectangle.h

```cpp
#ifndef _RECTANGLE_H_
#define _RECTANGLE_H_
class Rectangle {
  int a;
  int b;
public:
  Rectangle();
  Rectangle(int a, int b);
  virtual int width() const;
  virtual int height() const;
  virtual int area() const;
};

#endif // _RECTANGLE_H_
```

rectangle.cpp

```cpp
#include "rectangle.h"

Rectangle::Rectangle() : a{0}, b{0} {
}

Rectangle::Rectangle(int a, int b) : a{a}, b{b} {
}

int Rectangle::width() const {
  return a;
}

int Rectangle::height() const {
  return b;
}

int Rectangle::area() const {
  return a*b;
}
```

Basic implementation: Square class

square.h

```
#ifndef _SQUARE_H_
#define _SQUARE_H_
class Square {
  int a;
public:
  Square();
  Square(int a);
  int getEdge() const;
  int area() const;
};
#endif // _SQUARE_H_
```

square.cpp

```
#include "square.h"

Square::Square() : a{0} {
}

Square::Square(int a) : a{a} {
}

int Square::getEdge() const {
  return a;
}

int Square::area() const{
  return a*a;
}
```

Basic implementation: Adapter class

adapter.h

```cpp
#ifndef _ADAPTER_H_
#define _ADAPTER_H_
#include "rectangle.h"
#include "square.h"
class Adapter : public Rectangle, private Square{
public:
  Adapter(Square& s);
  int width() const override;
  int height() const override;
  int area() const override;
};
#endif // _ADAPTER_H_
```

adapter.cpp

```cpp
#include "adapter.h"

Adapter::Adapter(Square& s) : Square(s) {
}

int Adapter::width() const {
  return this->getEdge();
}

int Adapter::height() const {
  return this->getEdge();
}

int Adapter::area() const {
  return this->area();
}
```

Basic implementation: client code

main.cpp

```cpp
#include "adapter.h"
#include <iostream>

// client code
void doSomething(Rectangle* r) {
  int w = r->width();
  int h = r->height();

  std::cout << "width: " << w << "\nheight: " << h << std::endl;
}

int main() {
  Square s(2);
  Rectangle* r = new Rectangle(2,3);
  doSomething(r);
  //doSomething(s);
  Rectangle* a = new Adapter(s);
  doSomething(a);
  return 0;
}
```

- class adapter

Where to use?

-

Thank You!