Object-Oriented Programming

Design Pattern Adapter

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

Adapter

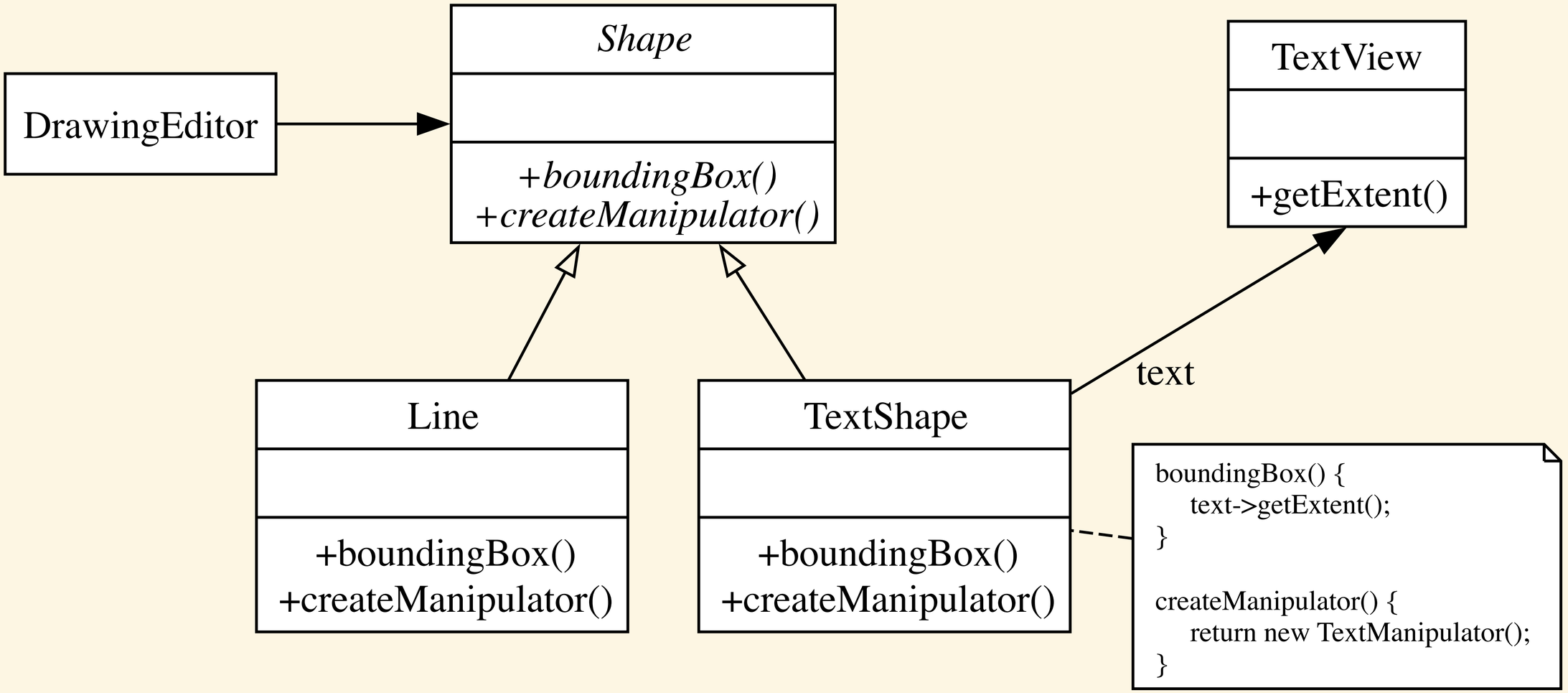
Convert the interface of a class into another interface clients expect

Adapter lets classes work together with incompatible interfaces

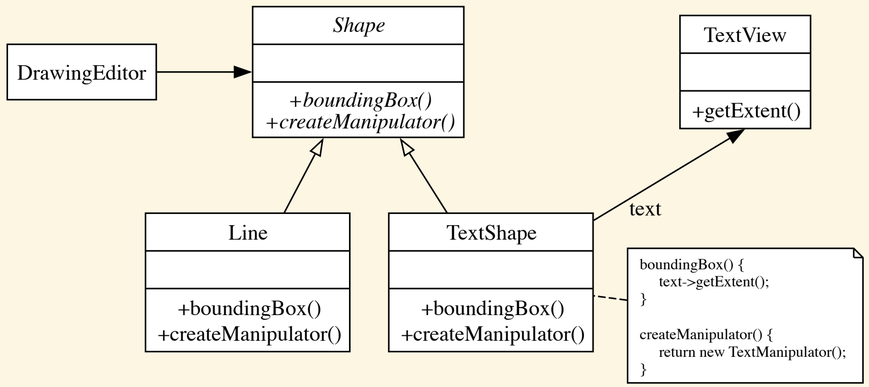
- Structural Pattern

- AKA: Wrapper

Adapter: Motivation



Adapter



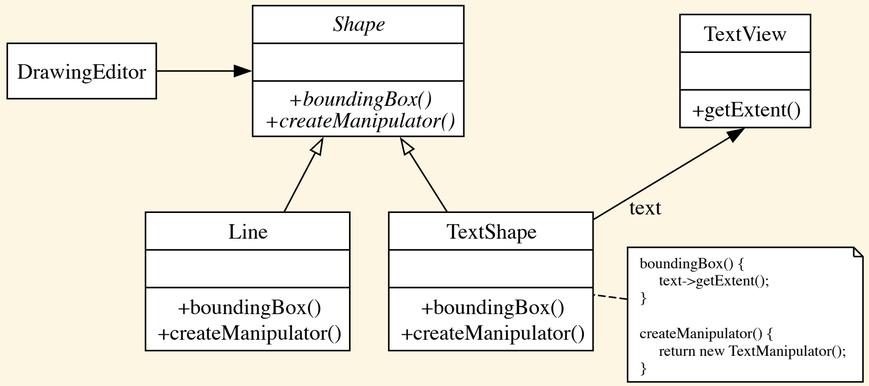
- Converts requests to the client into requests for the target class

- Allows a current class to be used by a client expecting a different interface

- Often, the first step towards replacement/major changes to a class with the "wrong" interface

- May add functionality missing in the target class

Adapter: Applicability

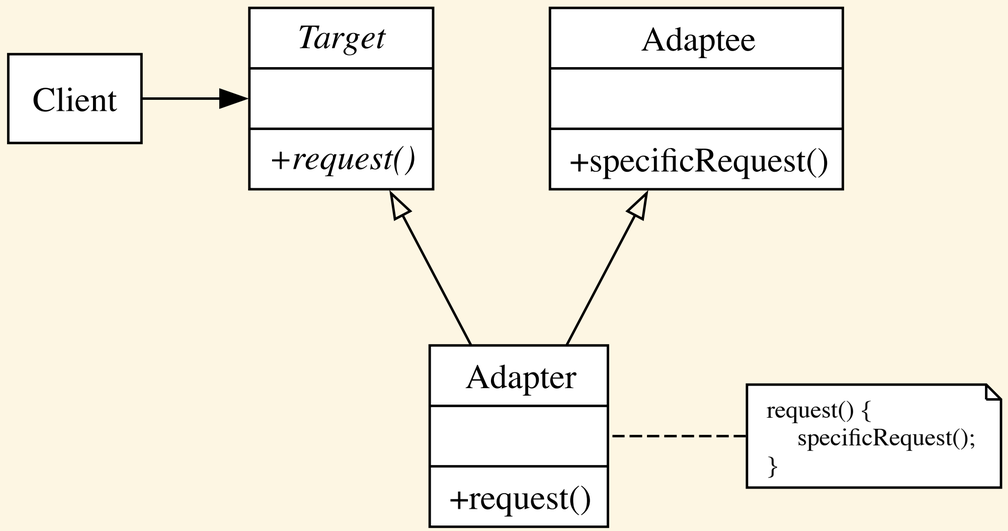


- Want to use an existing class, and its interface does not match the one you need

- Want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that don’t necessarily have compatible interfaces

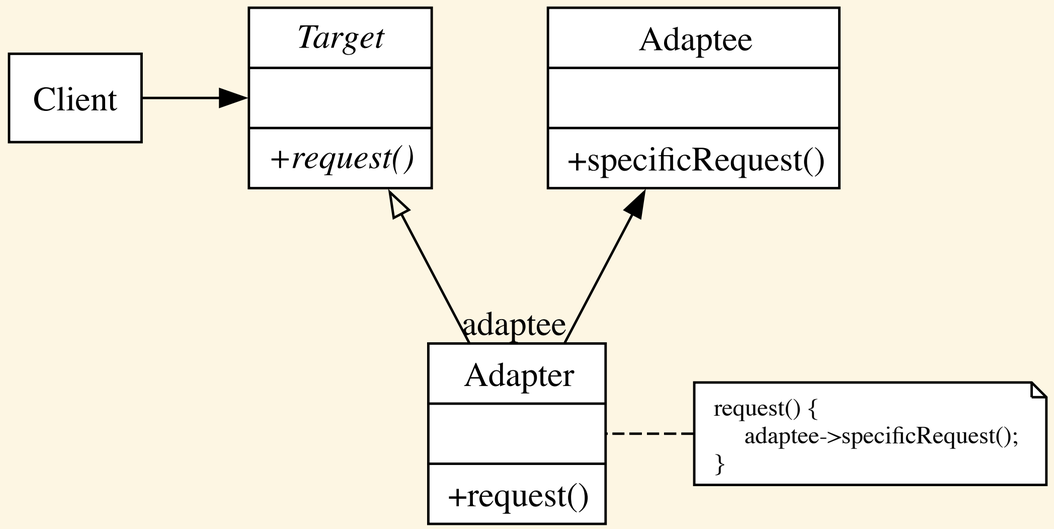
- *object adapter* We Need to use several existing subclasses, but it’s impractical to adapt their interface by subclassing every one

Adapter: Class Adapter Structure



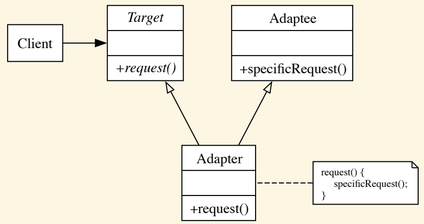

```
class Adaptee {
public:
    void specificRequest();
};

class Target {
public:
    virtual void request() = 0;
};

class Adapter : public Target, Adaptee {
public:
    Adapter()
        : Target(), Adaptee()
    {}

    void request() override {
        specificRequest();
    }
};
```

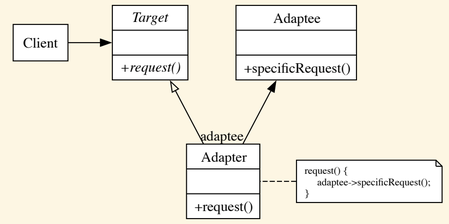
Adapter: Object Adapter Structure




```
class Adaptee {
public:
    void specificRequest();
};

class Target {
public:
    virtual void request() = 0;
};

class Adapter : public Target {
public:
    void request() override {
        adaptee.specificRequest();
    }

private:
    Adaptee adaptee;
};
```

Adapter: Structure Comparison



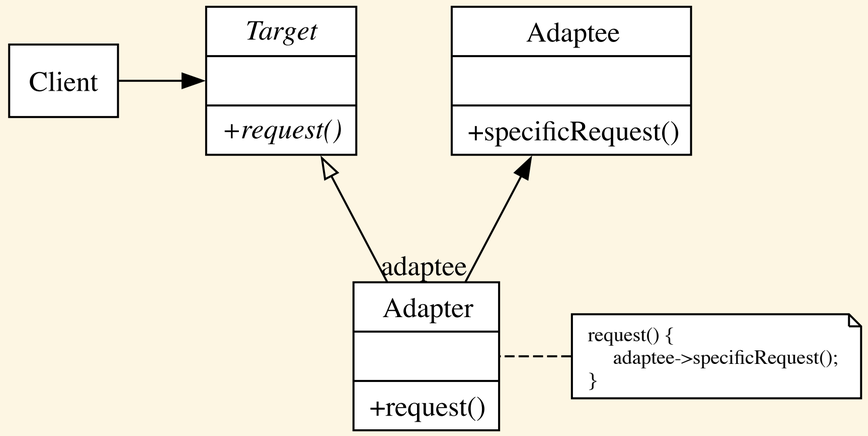


Adapter: Code Comparison

```
class Adaptee {
public:
    void specificRequest();
};

class Target {
public:
    virtual void request() = 0;
};

class Adapter : public Target, Adaptee {
public:
    Adapter()
        : Target(), Adaptee()
    {}

    void request() override {
        specificRequest();
    }
};
```

```
class Adaptee {
public:
    void specificRequest();
};

class Target {
public:
    virtual void request() = 0;
};

class Adapter : public Target {
public:
    void request() override {
        adaptee.specificRequest();
    }

private:
    Adaptee adaptee;
};
```

Adapter: Participants



- Target (e.g., Shape)

  Defines the domain-specific interface that the Client uses

- Client (e.g., DrawingEditor)

  Collaborates with objects conforming to the Target interface

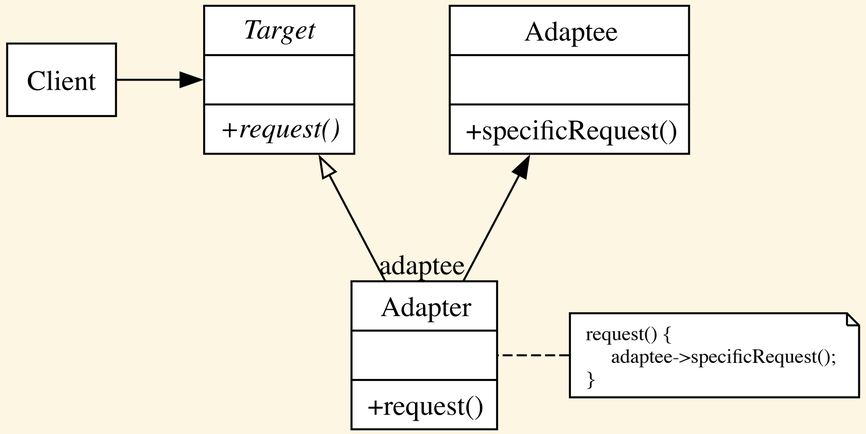
- Adaptee (e.g., TextView)

  Defines an existing interface that needs adapting

- Adapter (e.g., TextShape)

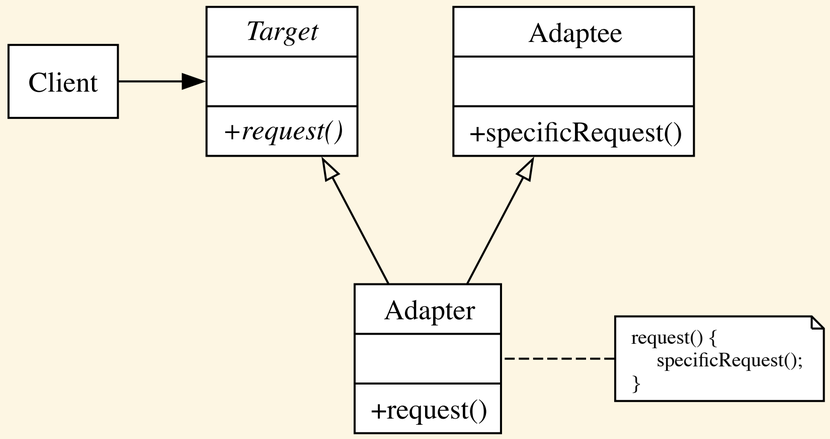
  Adapts the Adaptee interface to the Target interface

Adapter: Collaborations



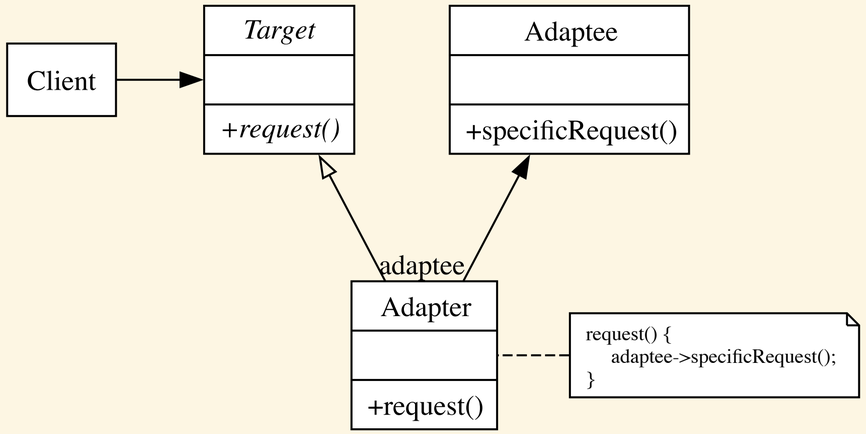
- Clients call operations on an Adapter instance
- The adapter calls Adaptee operations that carry out the request

Adapter: Class Adapter Consequences



- Adapts Adaptee to Target by committing to a concrete Adaptee class, which does not allow adaptation of a class and its subclasses
- Since the Adapter is a subclass of Adaptee, the Adapter can override some of Adaptee's behavior
- Introduces only one object, with no further pointer indirection

Adapter: Object Adapter Consequences



- A single Adapter can work with a class and all of its subclasses
- Does not directly allow subclassing Adaptee behavior

Adapter: Questions

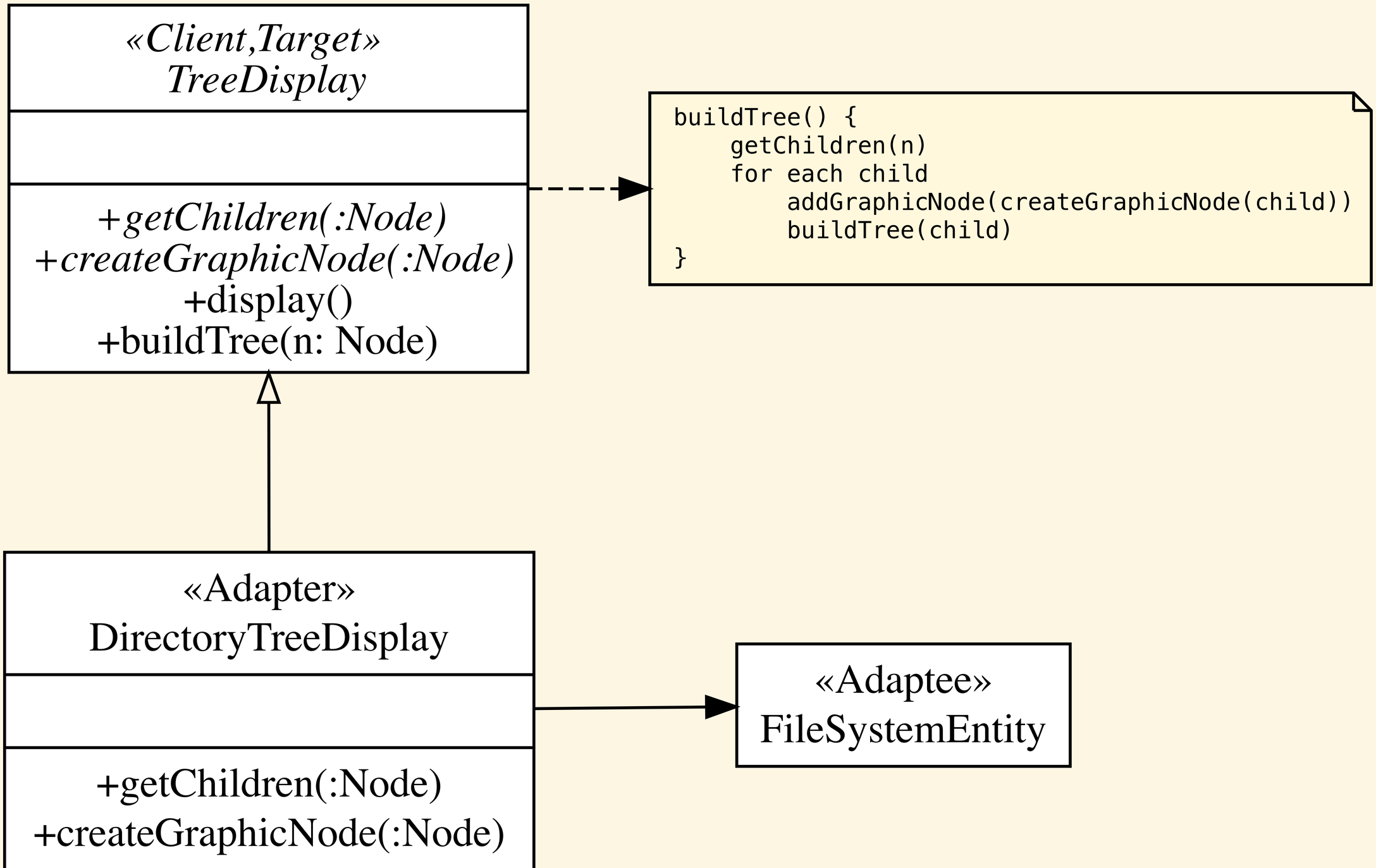
- How much adapting does the Adapter do?

- Pluggable adapters

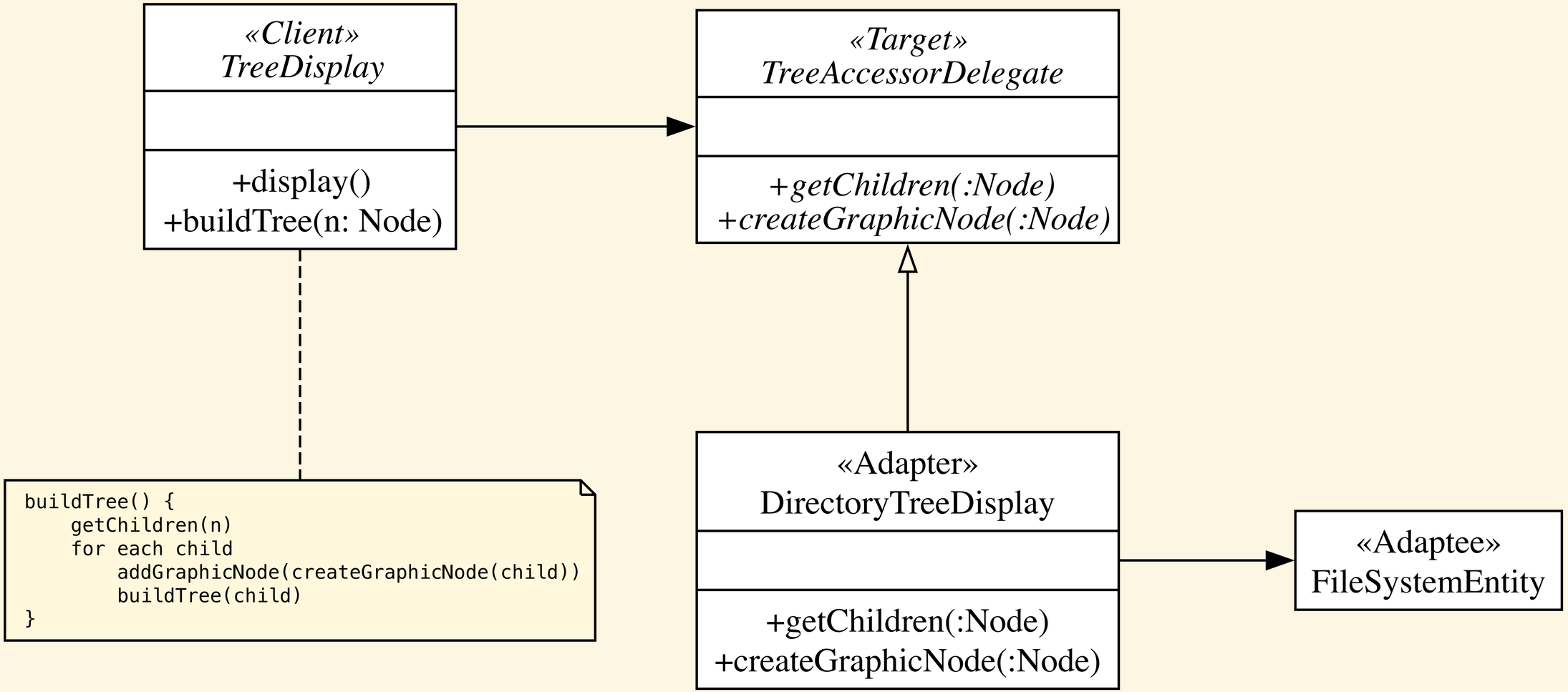
- Two-way adapters

- Note: Avoid class adapters in C++ unless necessary

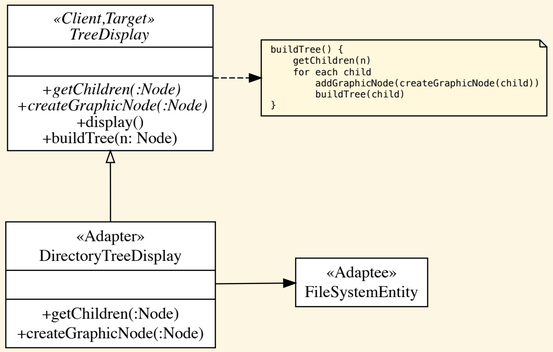
Implementation 1

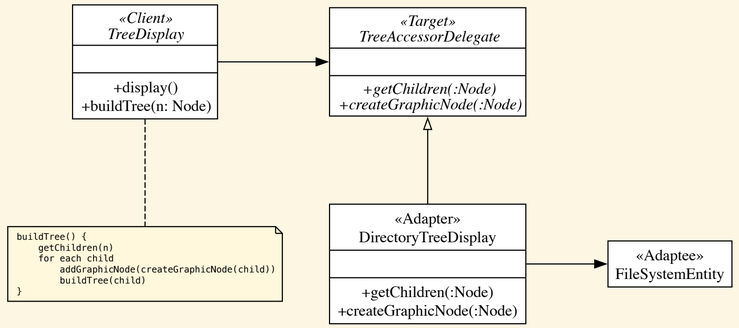


Implementation 2



Implementation Comparison





Related Patterns

- *Bridge*

  A similar structure but different intent

  Separates interface from implementation

- *Decorator*

  Add functionality to an existing object without changing the interface

  Supports recursive composition

- *Proxy*

  A representative or surrogate for another object

  Does not change its interface