Adapter Pattern

Jim Fawcett
CSE687 – Object Oriented Design, Spring 2003

Adapted from a Presentaton by
Matt Smouse and Jeff Ting
CSE776 – Design Patterns, Summer 2001

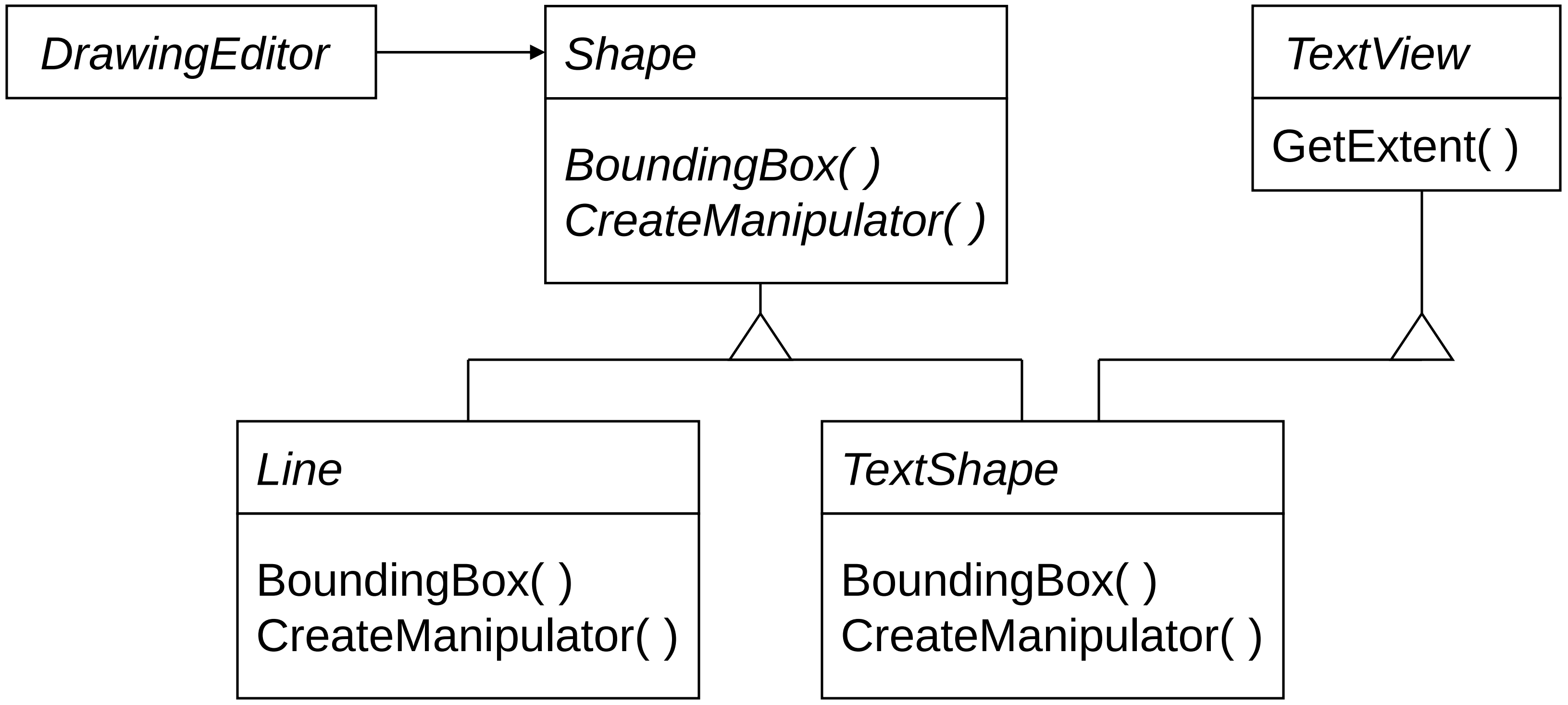
Intent

- Adapt an existing class rather than modify for reuse:
    - Convert interface of a useful class into another interface clients expect
- Also known as the wrapper design pattern; it wraps existing functionality of an adaptee with an adapter's inherited interface.

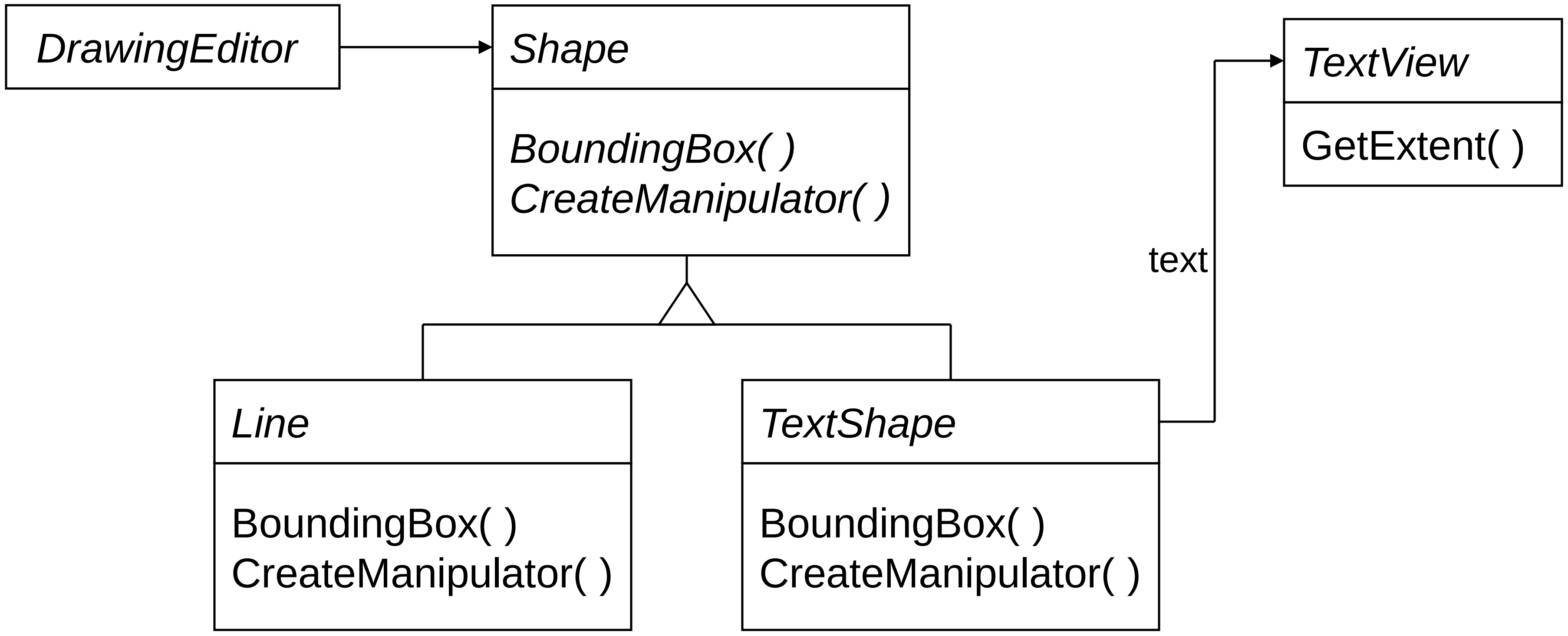
Motivation

- TextView example
  - Client is a drawing editor which wants to accommodate both shapes and text.
  - Adaptee is an existing TextView class which can display and edit text.
  - Target is a Shape class which provides the key abstraction for graphical objects.
  - Adapter is a TextShape class which inherits the Shape interface and adapts the TextView interface to the inherited Shape interface.

Motivation: Class Adapter



Motivation: Object Adapter

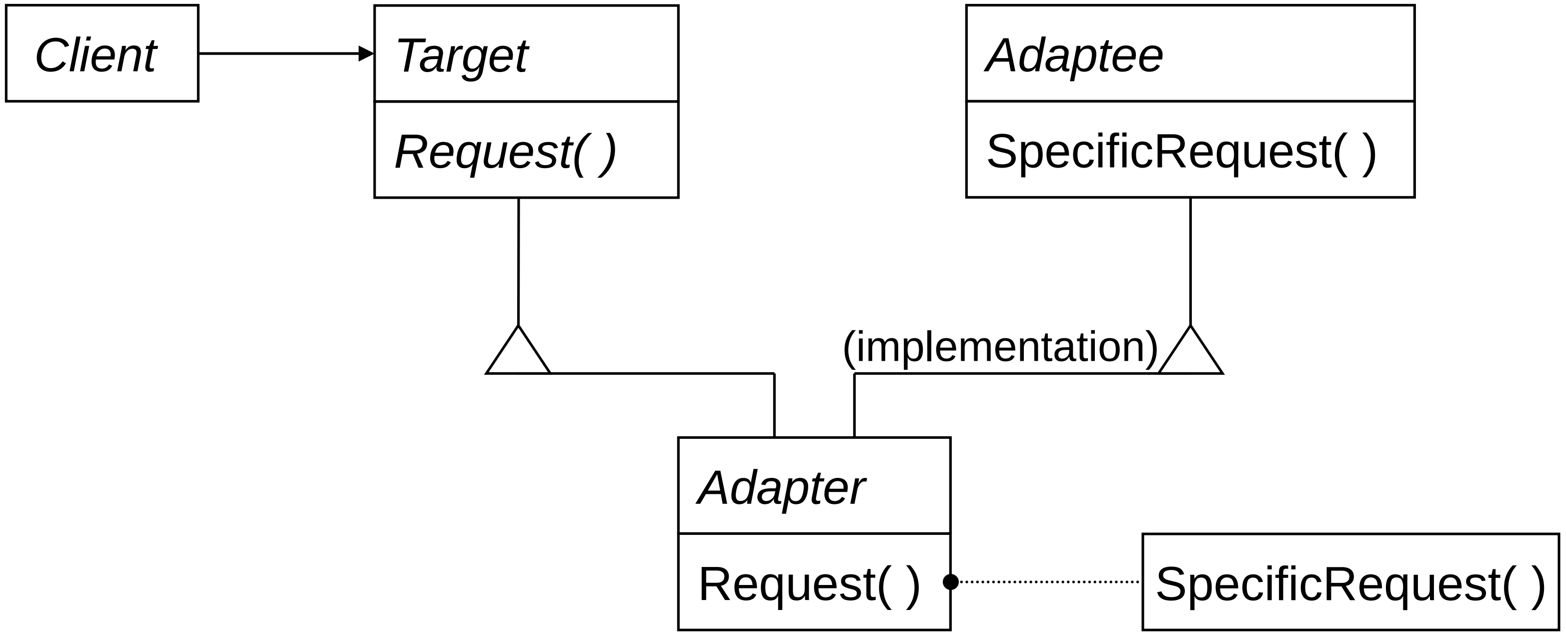


Applicability

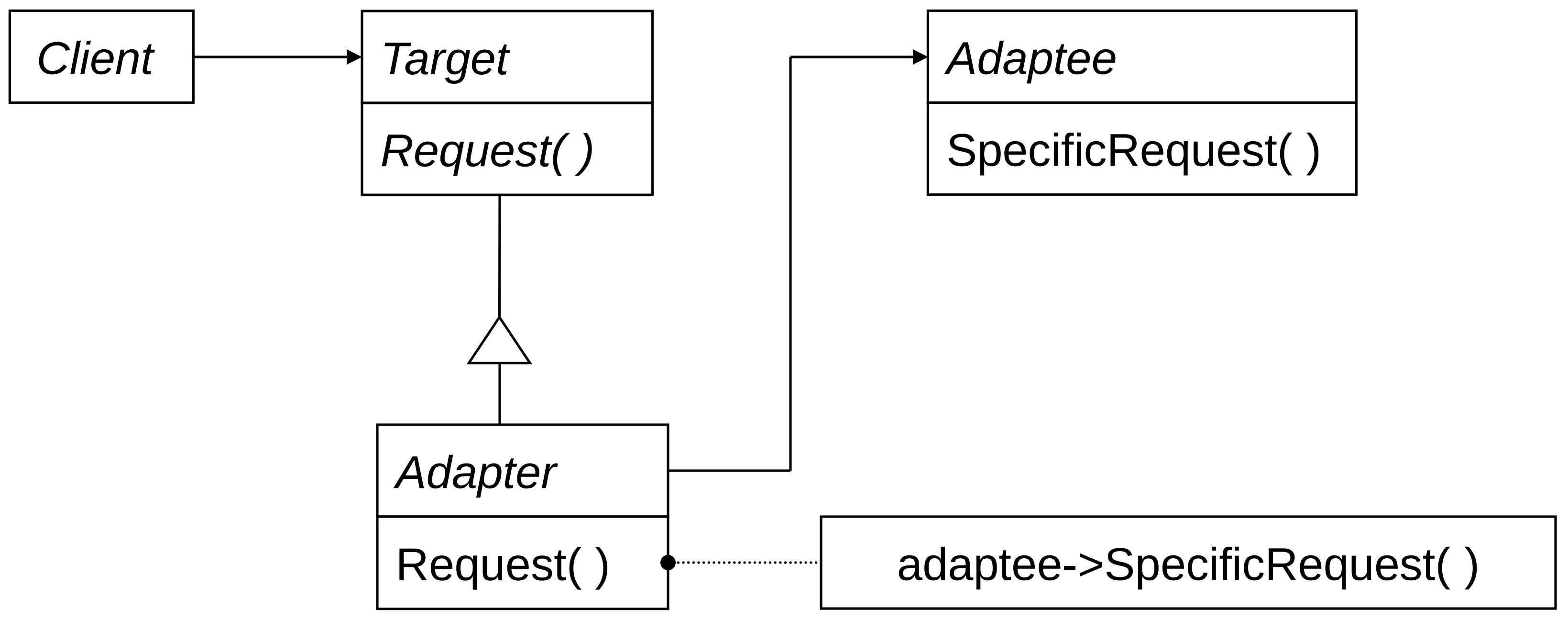
- Use an adapter when you have existing classes and need to add functionality while providing a common interface.
- Use an adapter when you want a reusable class which will handle software that you did not write.
- Use an object adapter when you have to provide for several different adaptee subclasses which inherit from a parent* adaptee.

*Must use object adapter when adaptee is abstract.

Structure: Class Adapter



Structure: Object Adapter



Participants

- Target
  - defines the interface that the Client will use.
- Client
  - creates and interacts with objects which conform to the target interface.
- Adaptee
  - has an interface that needs adapting.
- Adapter
  - adapts the interface of the adaptee to that of the target.

Collaborators

- Clients will create instances of the adapter and any adaptees to be used.
- Clients will call operations within instances of the adapter.
- The adapter will call adaptee operations on behalf of the client.
- The adapter may or may not perform additional processing on any data that the adaptee methods return.
- The adapter is the middleman for all operations involving the adaptee.

Consequences

- Class Adapter
  - Must commit to a concrete adaptee. This won’t work for an abstract adaptee, and it won’t work when adapting multiple adaptees.
  - The adapter can override some of the adaptee’s behavior; the adapter is a subclass (child) of the adaptee.
  - No additional pointer indirection to access adaptee.
- Object Adapter
  - Can work with adaptee subclasses and add functionality to those subclasses.
  - The adapter cannot easily override adaptee behavior; only can do this by referring to an adaptee subclass.

Implementation: Class Adapter

- Based upon multiple inheritance
- Inherit publicly from Target (the interface) and privately from Adaptee (the implementation)
- Adaptor is a subtype of Target but not of Adaptee

Implementation: Object Adapter

- Only inherits (publicly) from Target
- Adapter maintains pointer to Adaptee; Client must initialize this pointer
- Will work with subclasses of Adaptee

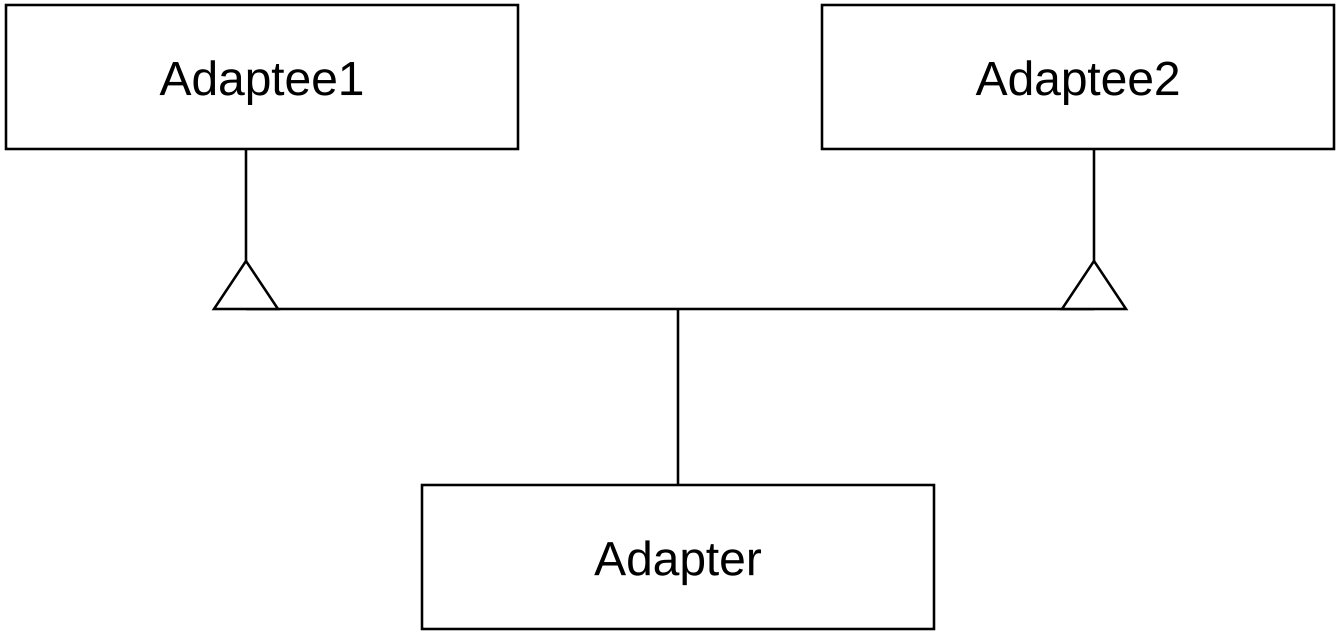
Pluggable Adapter

- Special object adapter
- Make the target contain abstract (pure virtual) methods.
  - Provide concrete operations which are common to any adapter implementation.
  - Provide abstract (pure virtual methods) operations which are required for unique adaptation.
  - Limit the number of abstract operations; it is easier to adapt a few necessary operations than to adapt many.
- The client will instantiate the adapter it wishes to use; since all adapters will conform to the target interface, we can swap them in and out of the client and it won’t know the difference (beyond instantiation).

Two-way Adapter

- Based upon multiple inheritance
- Inherits publicly from both classes (interfaces)
- Two Adaptees, no Target
- Used to combine functionality of two unlike classes

Two-way Adapter



Known Uses

- ET++Draw
- InterViews 2.6
- Pluggable adaptors in Smalltalk
- FixedStack
- ACE ORB (?)
- Java applications (object adapters only)

Related Patterns

- Bridge
  - Separates an interface from its implementation.
  - Adapter works with existing object.
- Decorator
  - Enhances another object without changing interface.
  - Adapter is less transparent to the client.
- Proxy
  - Representative for another object, no interface change
  - Adapter changes (adapts) the object’s interface.

End of Presentation