Adapter & Facade

CS356 Object-Oriented Design and Programming
http://cs356.yusun.io
November 19, 2014

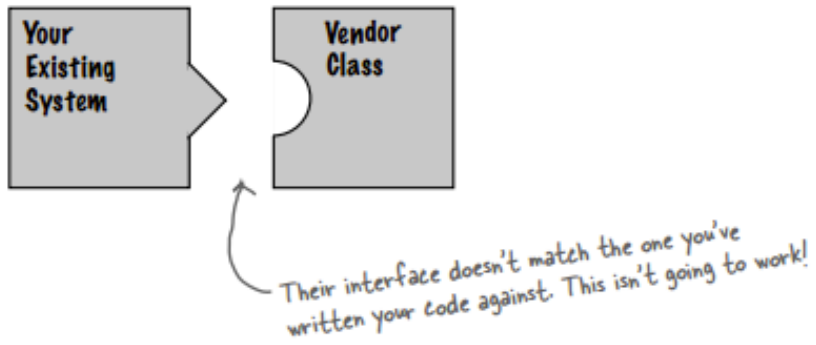
Yu Sun, Ph.D.
http://yusun.io
yusun@csupomona.edu

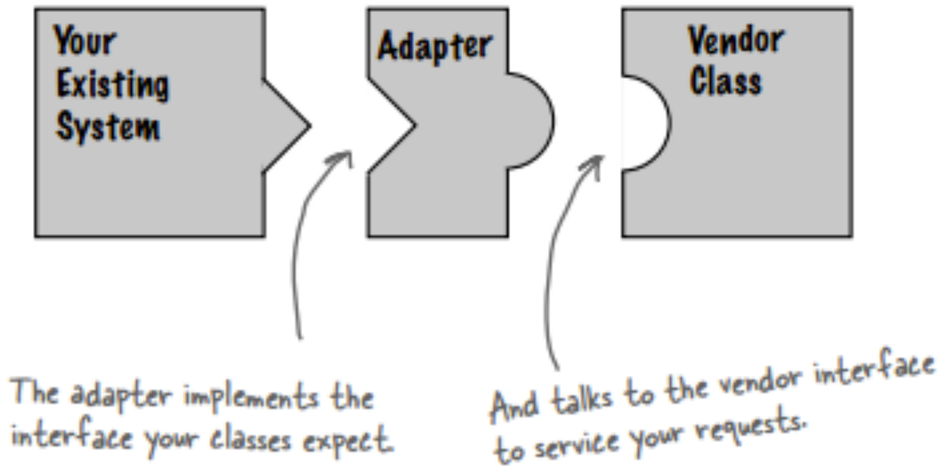
CAL POLY POMONA

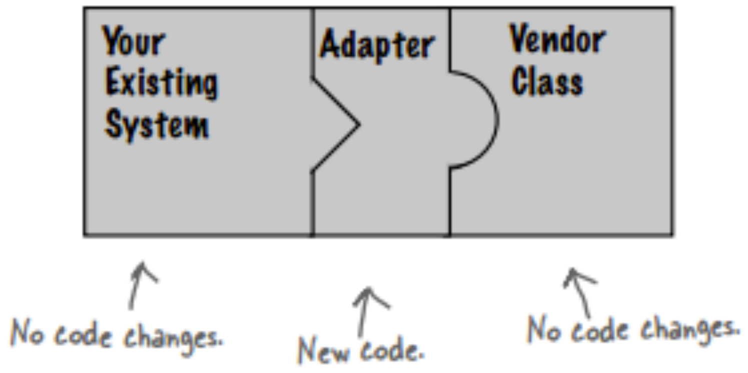
Adapter

- Problem
  - Have an object with an interface that's close to, but not exactly, what we need
- Context
  - Want to re-use an existing class
  - Can't change its interface
  - Impractical to extend class hierarchy more generally
    - May not have source code
- Solution
  - Wrap a particular class or object with the interface needed

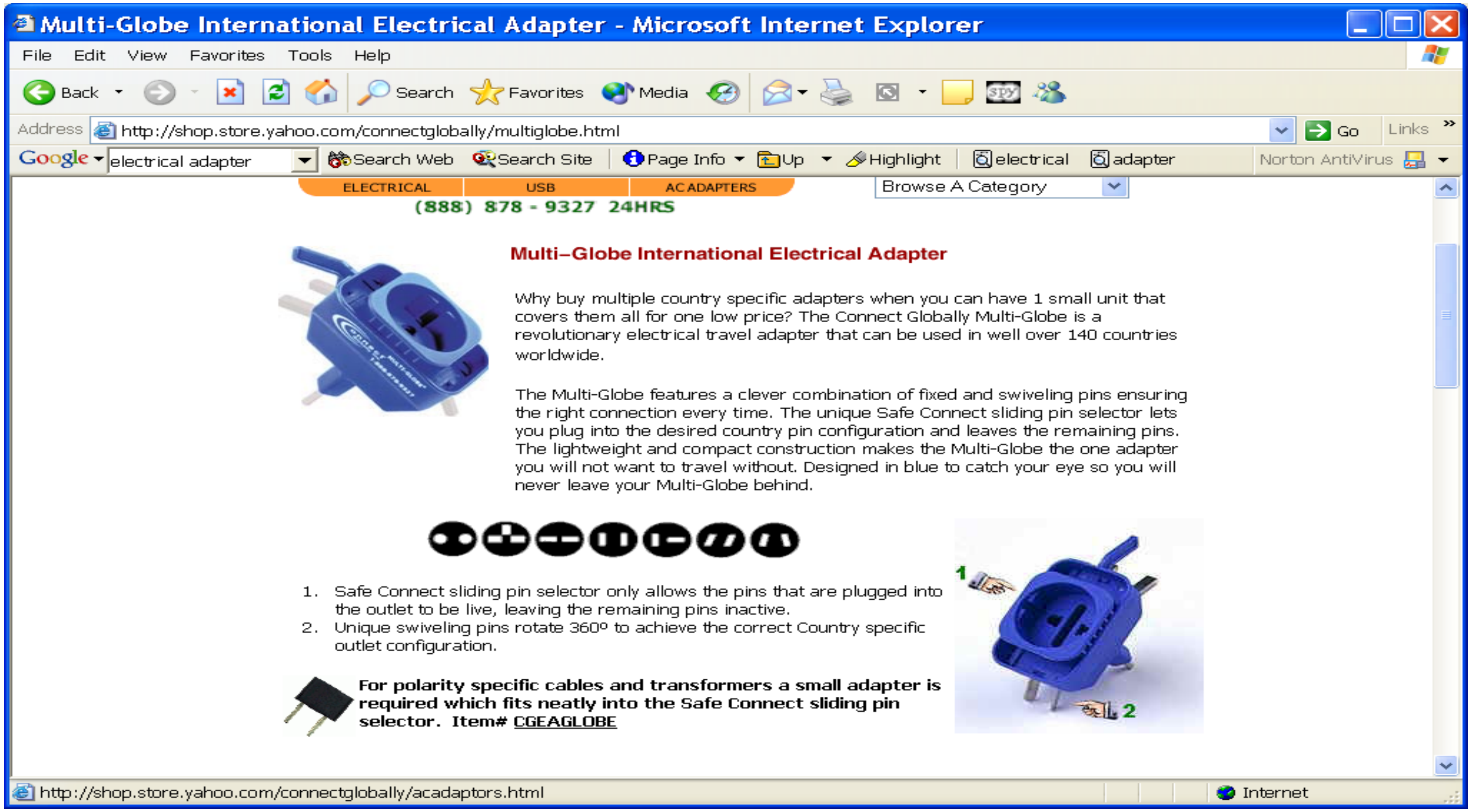
Motivation


Your Existing System
Vendor Class
Their interface doesn't match the one you've written your code against. This isn't going to work!


Your Existing System
Adapter
Vendor Class
The adapter implements the interface your classes expect.
And talks to the vendor interface to service your requests.


Your Existing System
Adapter
Vendor Class
No code changes.
New code.
No code changes.

Electrical Adapter…


Multi-Globe International Electrical Adapter - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://shop.store.yahoo.com/connectglobally/multiglobe.html Go Links
Google electrical adapter Search Web Search Site Page Info Up Highlight electrical adapter Norton AntiVirus
ELECTRICAL USB AC ADAPTERS
Browse A Category
(888) 878 - 9327 24HRS
1
2
http://shop.store.yahoo.com/connectglobally/acadaptors.html
Internet

Multi–Globe International Electrical Adapter

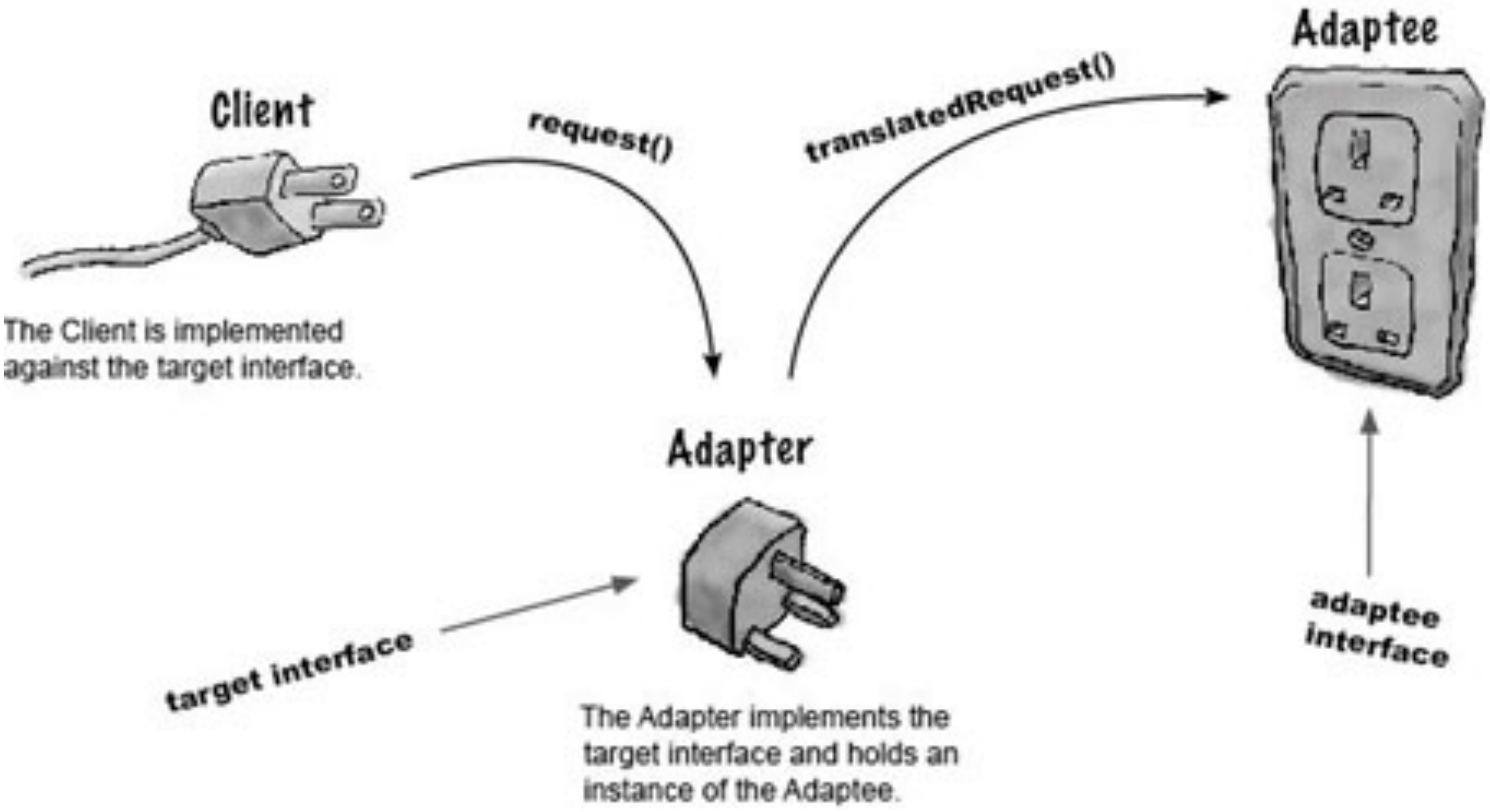
Why buy multiple country specific adapters when you can have 1 small unit that covers them all for one low price? The Connect Globally Multi-Globe is a revolutionary electrical travel adapter that can be used in well over 140 countries worldwide.

The Multi-Globe features a clever combination of fixed and swiveling pins ensuring the right connection every time. The unique Safe Connect sliding pin selector lets you plug into the desired country pin configuration and leaves the remaining pins. The lightweight and compact construction makes the Multi-Globe the one adapter you will not want to travel without. Designed in blue to catch your eye so you will never leave your Multi-Globe behind.

1. Safe Connect sliding pin selector only allows the pins that are plugged into the outlet to be live, leaving the remaining pins inactive.
2. Unique swiveling pins rotate 360º to achieve the correct Country specific outlet configuration.

For polarity specific cables and transformers a small adapter is required which fits neatly into the Safe Connect sliding pin selector. Item# CGEAGLOBE

Electrical Adapter…

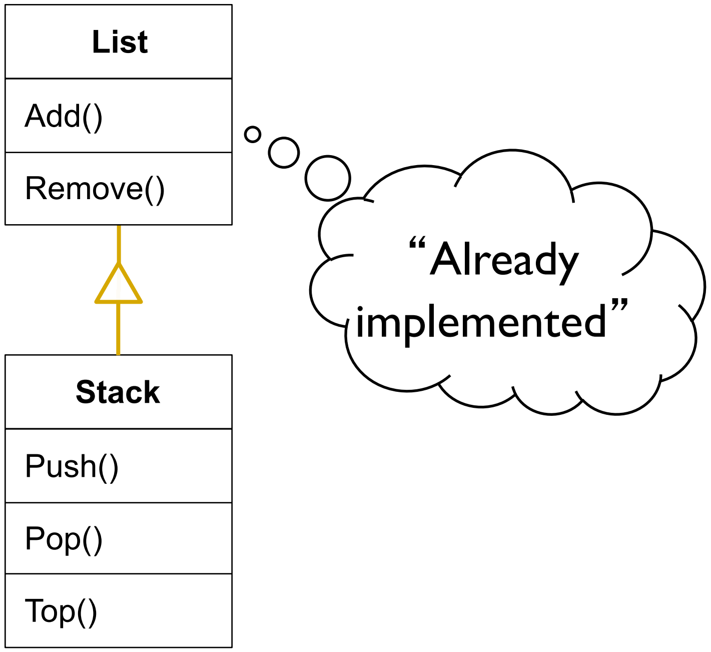

Client
request()
translatedRequest()
Adaptee
The Client is implemented against the target interface.
Adapter
target interface
adaptee interface
The Adapter implements the target interface and holds an instance of the Adaptee.

Reuse

- Main goal
  - Reuse knowledge from previous experience to current problem
  - Reuse functionality already available
- Composition
  - New functionality is obtained by aggregation
  - The new object with more functionality is an aggregation of existing components
- Inheritance
  - New functionality is obtained by inheritance

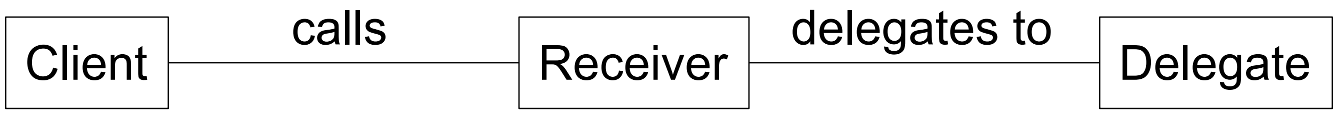
Inheritance

- A very similar class is already implemented that does almost the same as the desired class implementation
- Problem with implementation inheritance
  - Some of the inherited operations might exhibit unwanted behavior. What happens if the Stack user calls Remove() instead of Pop()?



Delegation

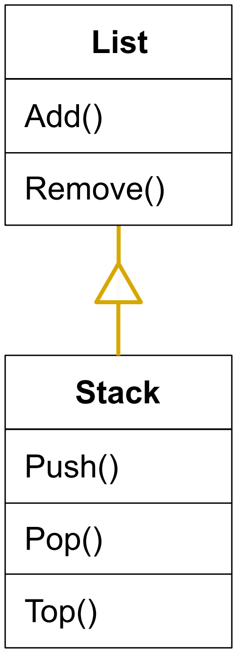
- Delegation is a way of making composition (for example aggregation) as powerful for reuse as inheritance
  - Instead of "inheriting from" a class, we "delegate to" another object
- In Delegation, two objects are involved in handling a request
  - A receiving object delegates operations to its delegate
  - The developer can make sure that the receiving object does not allow the client to misuse the delegate object



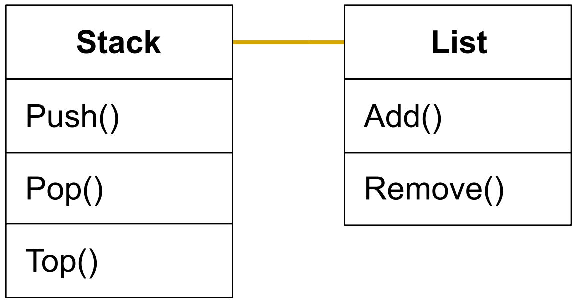
Delegation instead of Inheritance

- Delegation: Catching an operation and sending it to another object

Stack implemented by Inheritance



Stack implemented by Delegation

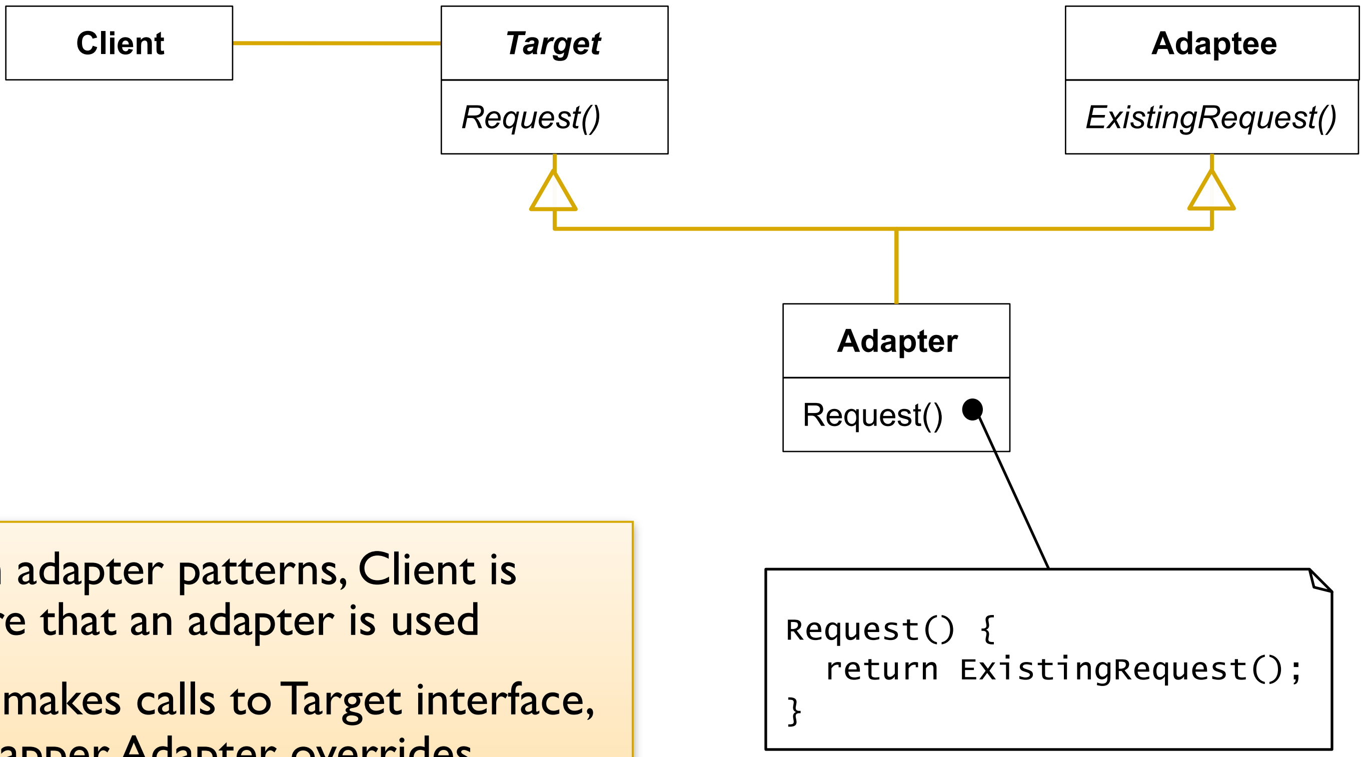


```
public class Stack {
  protected List delegatee;

  public Stack() {
    delegatee = new List();
  }

  public Object push(Object item) {
    delegatee.Add(item);
  }
  ...
```

Adapter Pattern

- “Convert the interface of a class into another interface clients expect.”
  - Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces
- Used to provide a new interface to existing legacy components (Interface engineering, reengineering)
- Also known as a “wrapper”
- Two adapter patterns
  - Class adapter
    - Uses multiple inheritance to adapt one interface to another
  - Object adapter
    - Uses single inheritance and delegation
- We will mostly use object adapters and call them simply adapters

Class Adapter Pattern (Based on Multiple Inheritance)

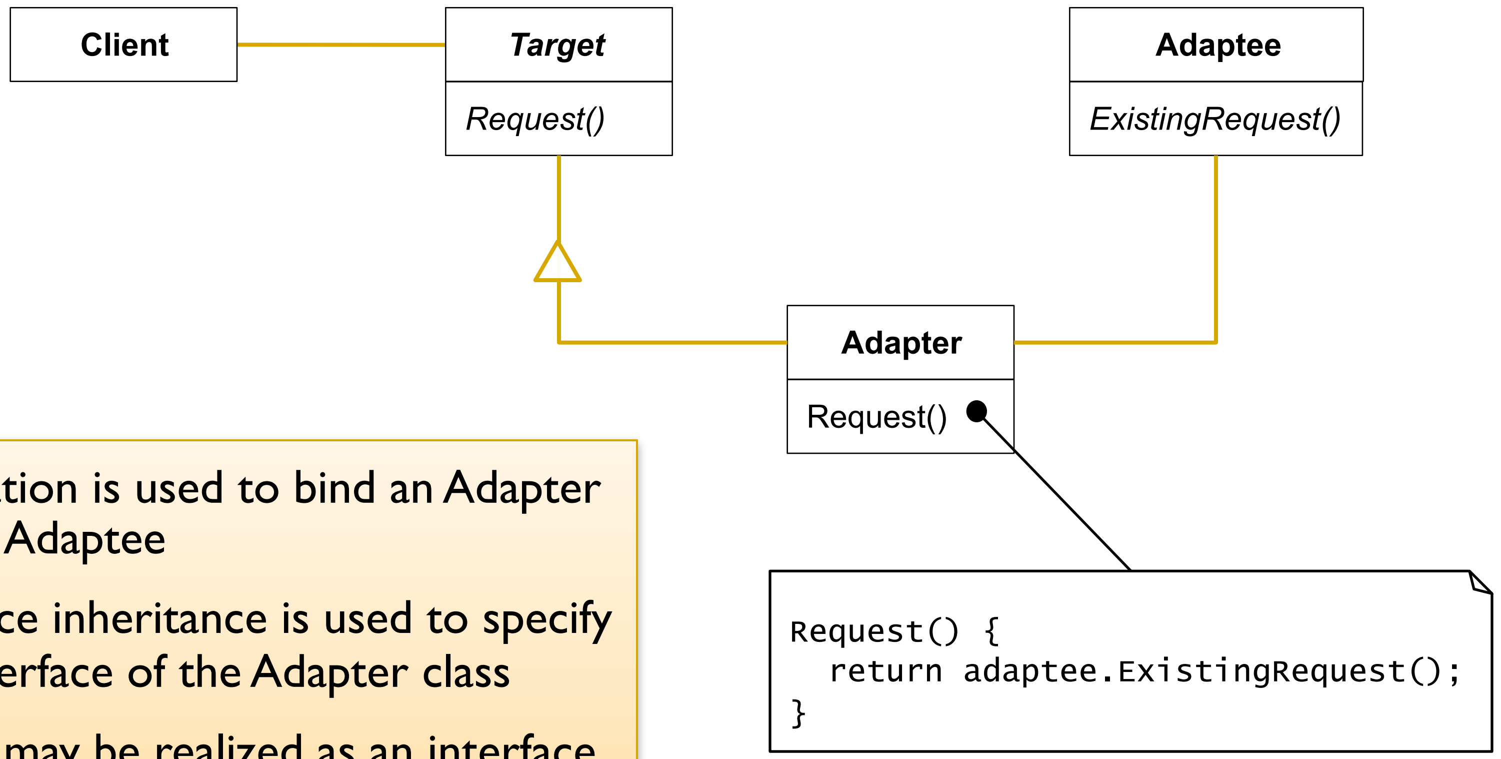


```
Request() {
  return ExistingRequest();
}
```

In both adapter patterns, Client is unaware that an adapter is used

Simply makes calls to Target interface, and wrapper Adapter overrides Request with calls to legacy code

Adapter pattern (Object Adapter)



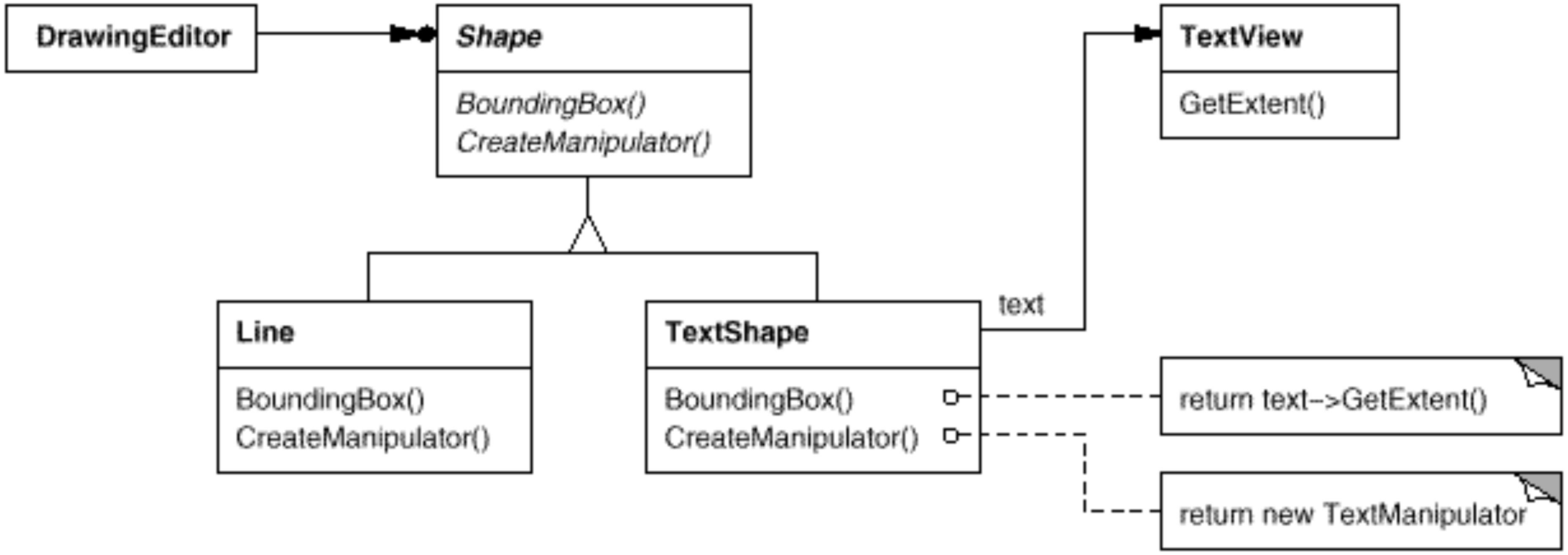
```
Request() {
  return adaptee.ExistingRequest();
}
```

Delegation is used to bind an Adapter and an Adaptee

Interface inheritance is used to specify the interface of the Adapter class

Target may be realized as an interface in Java

Example of the Object Adapter Pattern



Class Shape and TextView

```
class Shape {
public:
  Shape();
  virtual void BoundingBox (Point& bottomLeft, Point& topRight);
  virtual Manipulator* CreateManipulator() const;
};
```

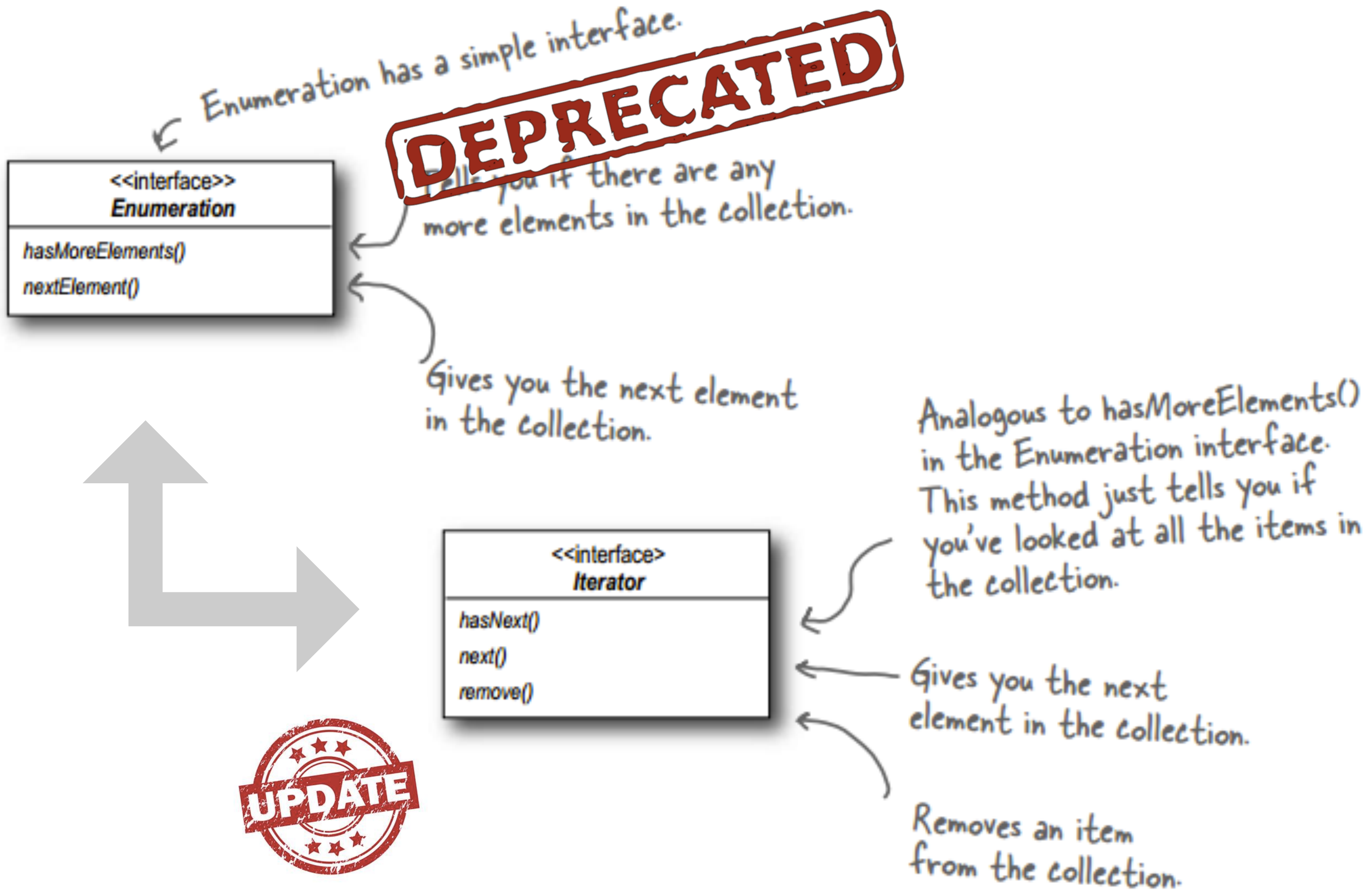
```
class TextView {
public:
  TextView();
  void GetOrigin(Coord& x, Coord& y);
  void GetExtent(Coord& width, Coord& height);
  virtual bool IsEmpty() const;
};
```

Class TextShape and Method BoundingBox

```
class TextShape : public Shape {
public:
  TextShape(TextView*);
  virtual void BoundingBox(Point& bottomLeft, Point& topRight);
  virtual bool IsEmpty();
  virtual Manipulator* CreateManipulator();
private:
  TextView* text;
};
```

```
void TextShape::BoundingBox(Point& bottomLeft, Point& topRight) {
  Coord bottom, left, width, height;
  text->GetOrigin(bottom, left);
  text->GetExtent(width, height);
  bottomLeft = Point(bottom, left);
  topRight = Point(bottom-height, left+width);
}
```

Adapt Enumeration to Iterator



Enumeration has a simple interface.
DEPRECATED
<<interface>>
Enumeration
hasMoreElements()
nextElement()
Tells you if there are any more elements in the collection.
Gives you the next element in the collection.
<<interface>
Iterator
hasNext()
next()
remove()
Analogous to hasMoreElements() in the Enumeration interface. This method just tells you if you've looked at all the items in the collection.
Gives you the next element in the collection.
Removes an item from the collection.
UPDATE

Adapt Enumeration to Iterator

```
public class EnumerationIterator implements Iterator
{
    Enumeration enum;

    public EnumerationIterator(Enumeration enum) {
        this.enum = enum;
    }

    public boolean hasNext() {
        return enum.hasMoreElements();
    }

    public Object next() {
        return enum.nextElement();
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

Since we're adapting Enumeration to Iterator, our Adapter implements the Iterator interface... it has to look like an Iterator.

The Enumeration we're adapting. We're using composition so we stash it in an instance variable.

The Iterator's hasNext() method is delegated to the Enumeration's hasMoreElements() method...

... and the Iterator's next() method is delegated to the Enumerations's nextElement() method.

Unfortunately, we can't support Iterator's remove() method, so we have to punt (in other words, we give up!). Here we just throw an exception.

Adapter Summary

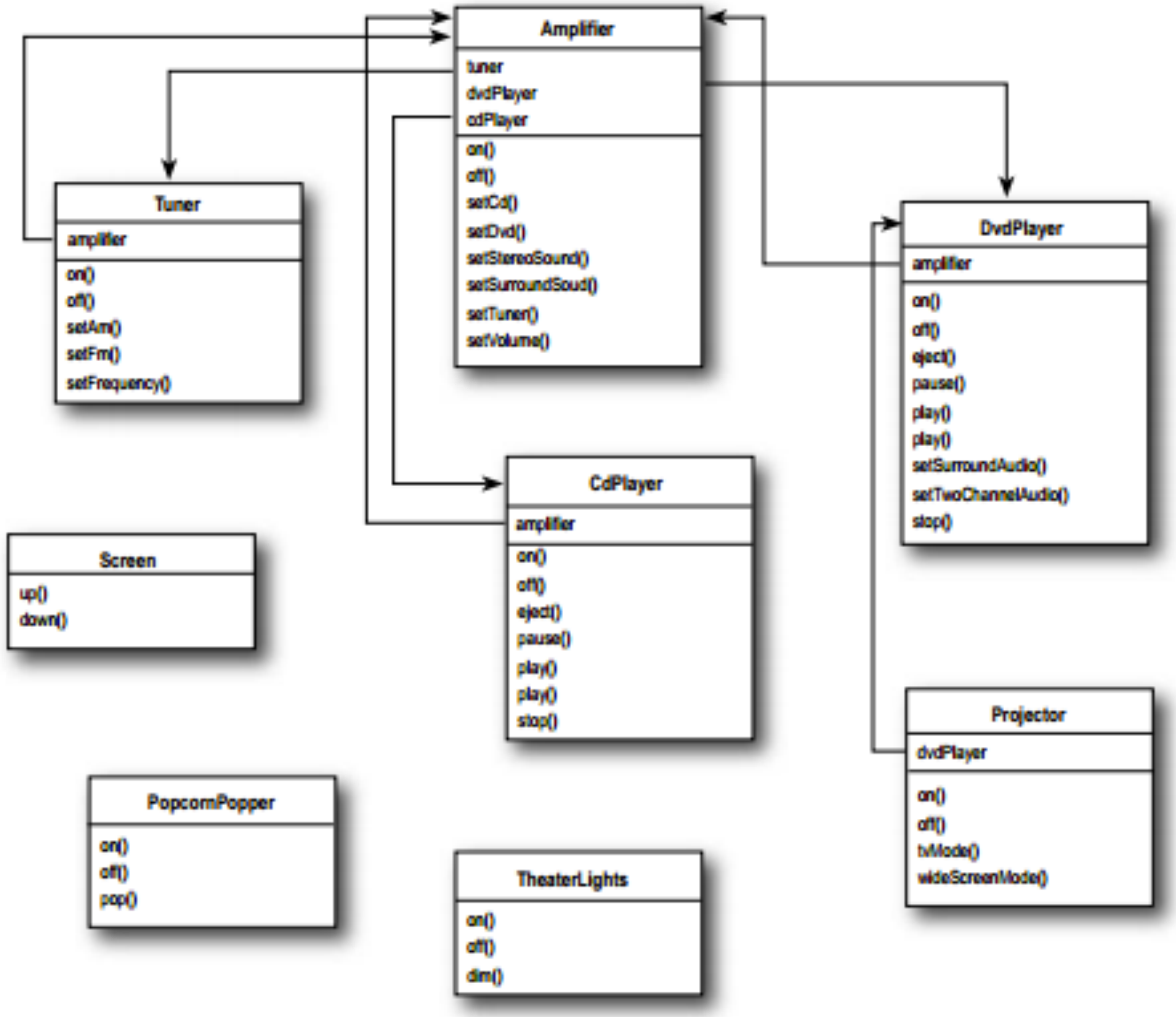
- Adapters are all about interface mapping between two artifacts
- Often, the goal is to find a "narrow" interface for Adaptee; that is, the smallest subset of operations that lets us do the adaptation
- Pay attention to Class Adapter (Inheritance)!

Facade Pattern

Facade

- *Intent*
  - Provides a unified interface to a set of subsystem interfaces
  - A higher-level interface making the subsystem easier to use

Motivating Example – Home Theater

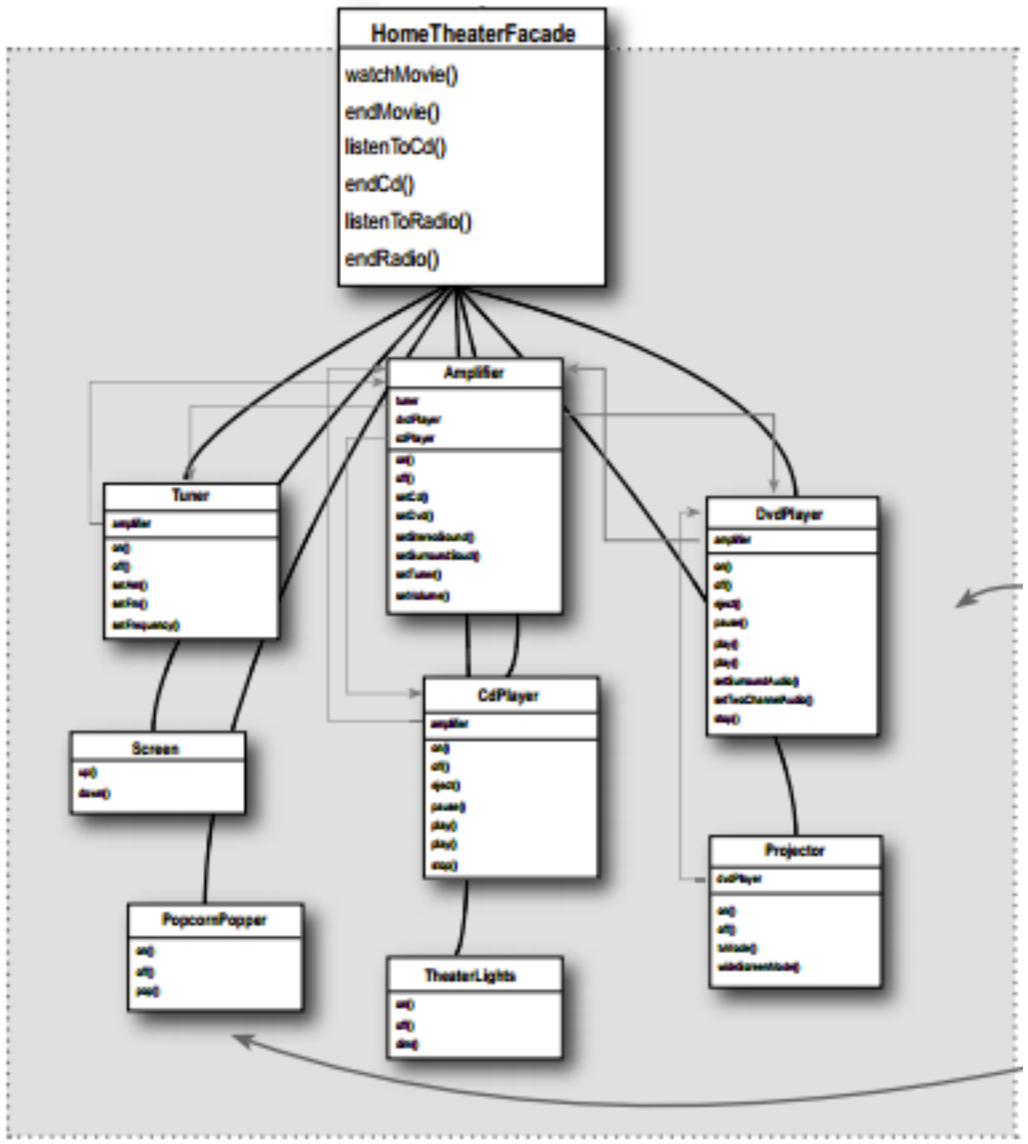


Motivating Example – Home Theater

- To watch a movie, you need to:

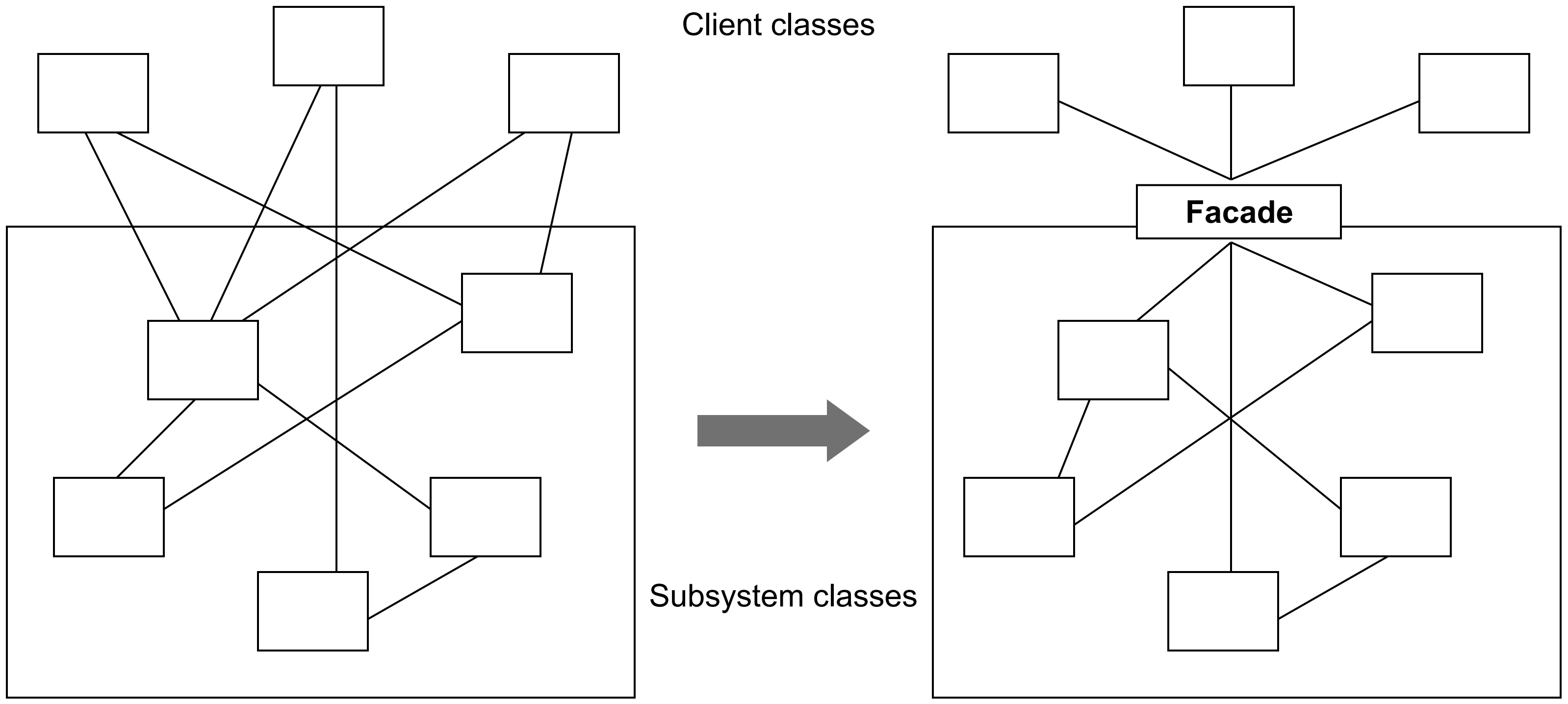
1. Turn on the popcorn popper
2. Start the popper popping
3. Dim the lights
4. Put the screen down
5. Turn the projector on
6. Set the projector input to DVD
7. Put the projector on wide-screen mode
8. Turn the sound amplifier on
9. Set the amplifier to DVD input
10. Set the amplifier to surround sound
11. Set the amplifier volume to medium (5)
12. Turn the DVD Player on
13. Start the DVD Player playing

Motivating Example – Home Theater



Motivation

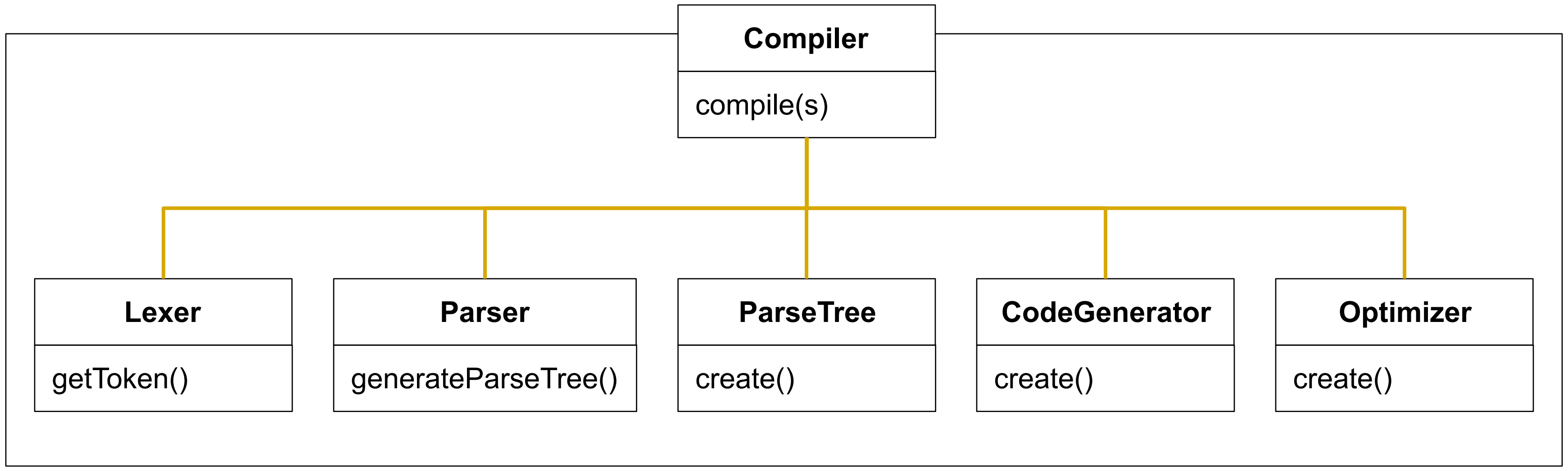
- Making a system into subsystems helps reduce complexity
- Minimize subsystem communications and dependencies
- Facade can provide a single, simplified interface to the more general facilities of a subsystem



Facade

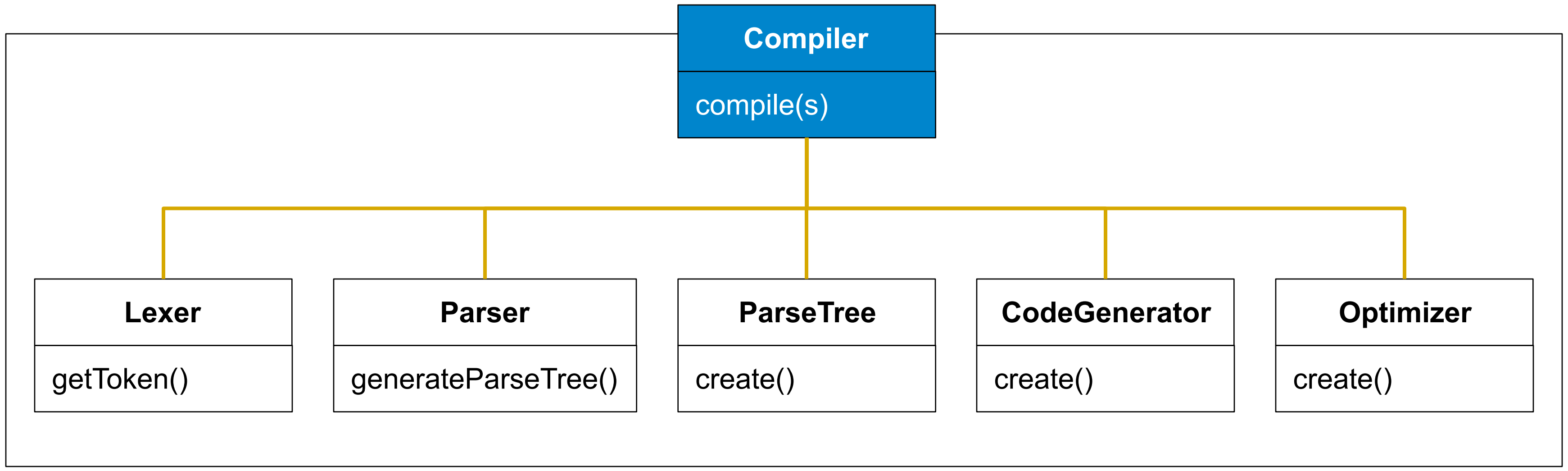
- *Applicability*
  - To provide a simple interface to a complex subsystem
    - Subsystems often get more complex as they evolve
  - To decouple subsystem from clients and other subsystems
    - Promoting subsystem independence and portability
  - To layer subsystems
    - Define an entry point to each subsystem level
    - Minimize subsystem inter-dependencies

Example – Compiler Facade



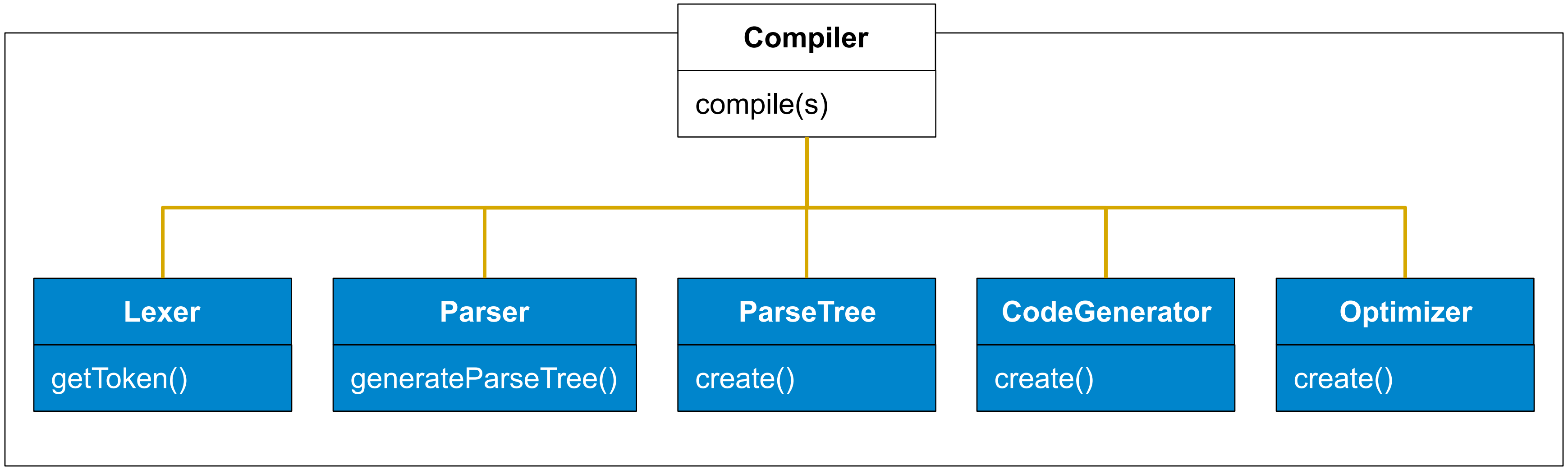
Facade

- Knows which subsystem classes may handle a request
- Delegates client requests to appropriate subsystem objects



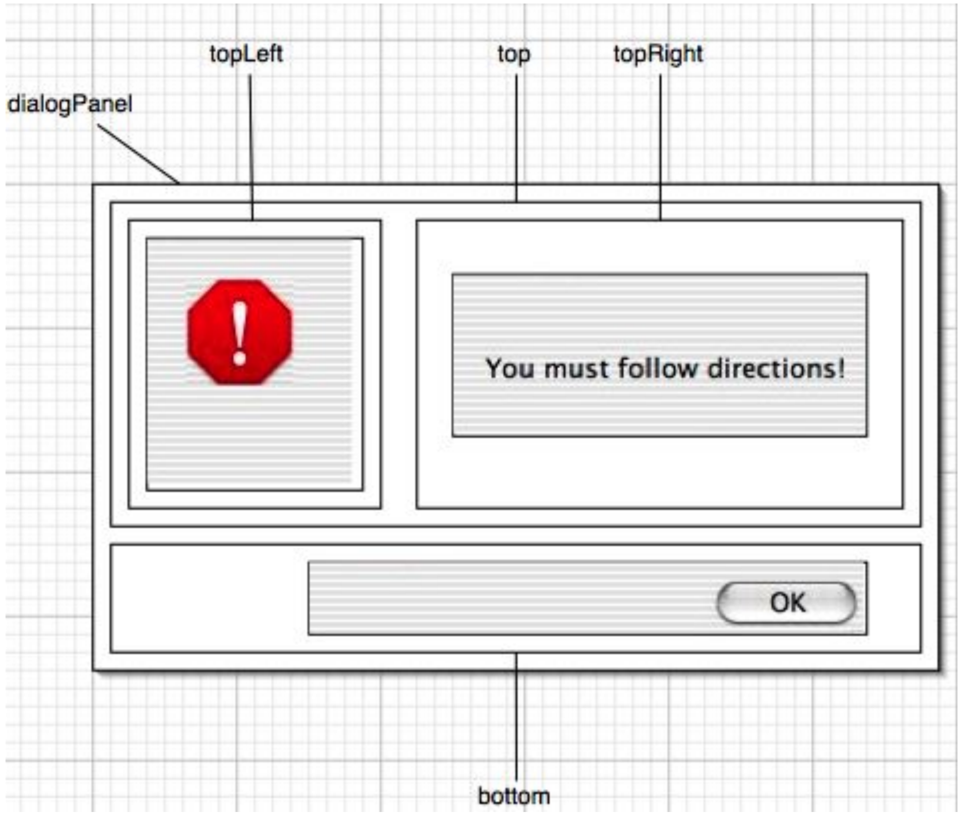
Subsystem Classes

- Implement subsystem functionality
- Have no knowledge of the facade
  - i.e., keep no references to it

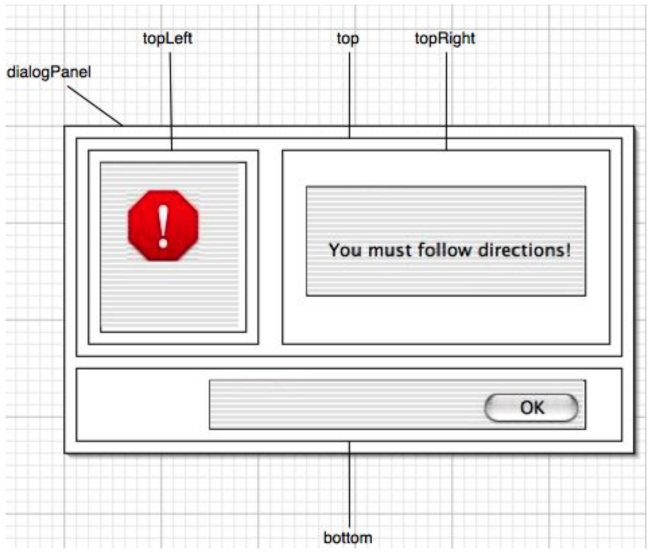


Clients communicate with the subsystem by sending requests to Facade, which forwards them to the appropriate subsystem object(s)

Example: Dialog Boxes

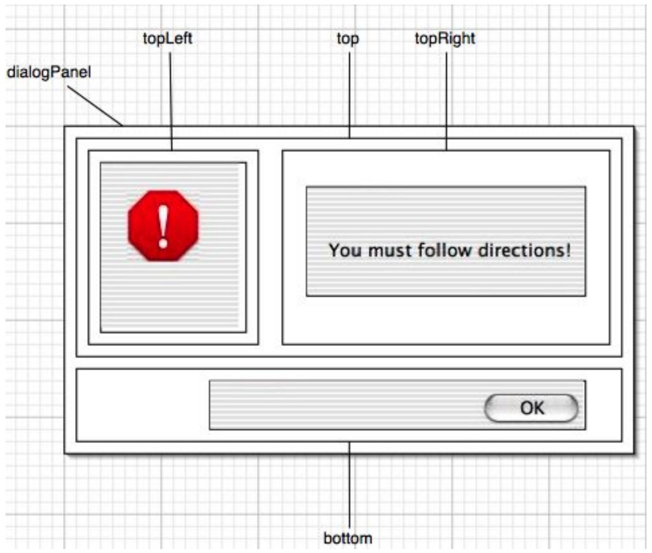


Top-left, Top-right, and Top Panels



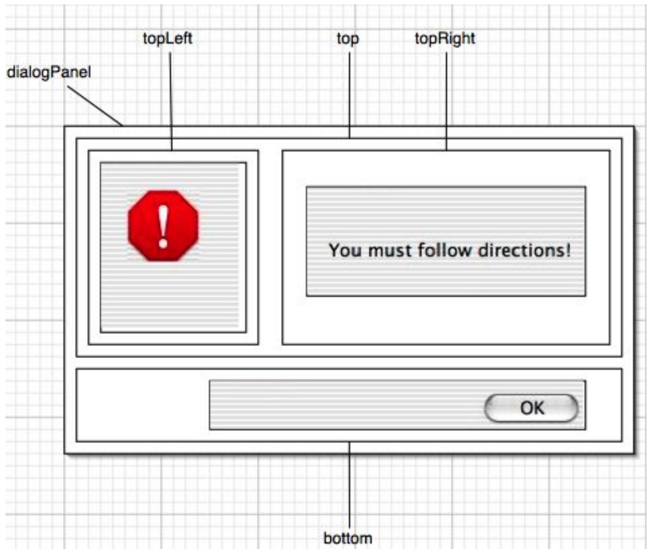
```
...
JDialog dialog = new JDialog(this, "Error!", false);
...
newButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    // Top-left panel:
    JPanel topLeft = new JPanel();
    topLeft.setLayout(new FlowLayout(FlowLayout.LEFT));
    topLeft.add(new JLabel(new
      ImageIcon("../graphics/stopsign.jpg")));

    // Top-right panel:
    JPanel topRight = new JPanel();
    topRight.setLayout(new BorderLayout());
    topRight.add(new
      JLabel("You must follow directions!“,
      JLabel.CENTER), BorderLayout.EAST);

    // Top panel:
    JPanel top = new JPanel();
    top.setLayout(new BorderLayout(15,0));
    top.add(topLeft, BorderLayout.WEST);
    top.add(topRight, BorderLayout.EAST);
```

OK Button, Bottom and Dialog Panels



```
...
// OK button:
JButton button = new JButton("OK");
button.setDefaultCapable(true);
getRootPane().setDefaultButton(button);
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    dialog.hide();
  }
});

// Bottom panel:
JPanel bottom = new JPanel();
bottom.setLayout(new BorderLayout());
bottom.add(button, BorderLayout.EAST);

// Dialog panel:
JPanel dialogPanel = new JPanel();
dialogPanel.setBorder(
  BorderFactory.createEmptyBorder(15,15,15,10));
dialogPanel.setLayout(new BorderLayout());
dialogPanel.add(top, BorderLayout.NORTH);
dialogPanel.add(bottom, BorderLayout.SOUTH);
Container cp = dialog.getContentPane();
...
```

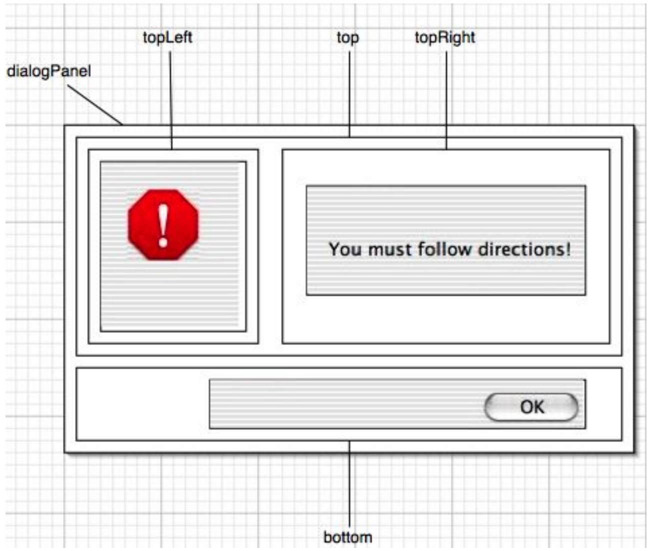
Panel Attributes

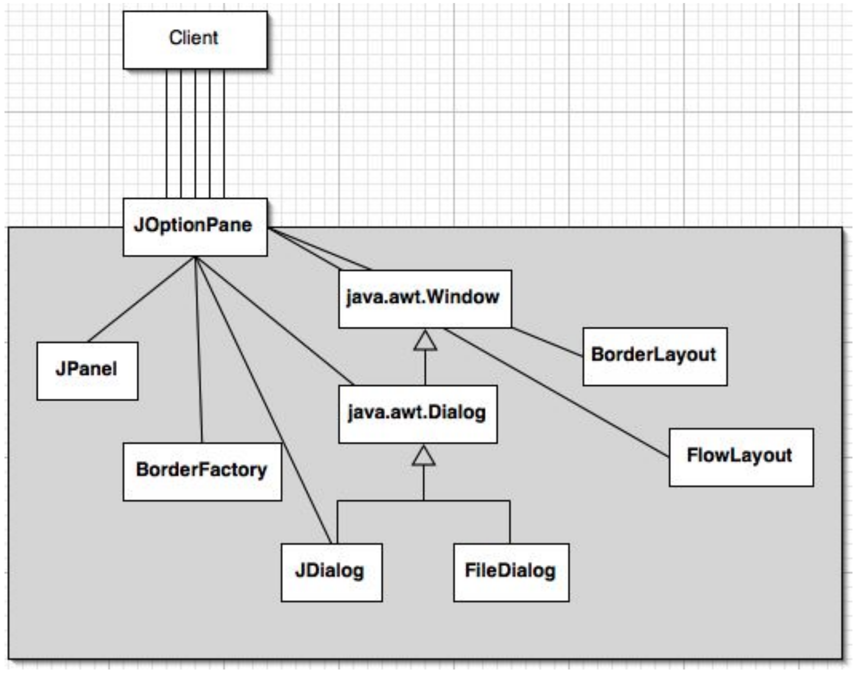


```
      ...
      cp.add(dialogPanel);
      dialog.setResizable(false);
      dialog.pack();
      dialog.setLocationRelativeTo(TLDViewer.this);
      dialog.show();
   }
});
...
```

JOptionPane Facade

```
...
openButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    JOptionPane.showMessageDialog(TLDViewer.this,
      "You must follow directions!",
      "Error!", JOptionPane.ERROR_MESSAGE);
  }
});
...
```





Consequences

+ Shields clients from subsystem components
    - Number of objects that clients deals is reduced
+ Promotes weak coupling between subsystem and clients
    - Subsystem components may be strongly coupled
    - Weak coupling lets you vary the components of the subsystem without affecting its clients
+ Eliminates complex or circular dependencies
+ Can reduce compilation dependencies in large systems

× Doesn't prevent applications from using subsystem classes if they need to