CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2007
Doc 15 Observer, Adapter & Builder
Apr 11, 2007

References

Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, 1995, pp. 293-303, 139-150, 97-106

The Design Patterns Smalltalk Companion, Alpert, Brown, Woolf, Addision-Wesley, 1998, pp. 305-326, 105-120, 47-62

Java API

VisualWorks Smalltalk API

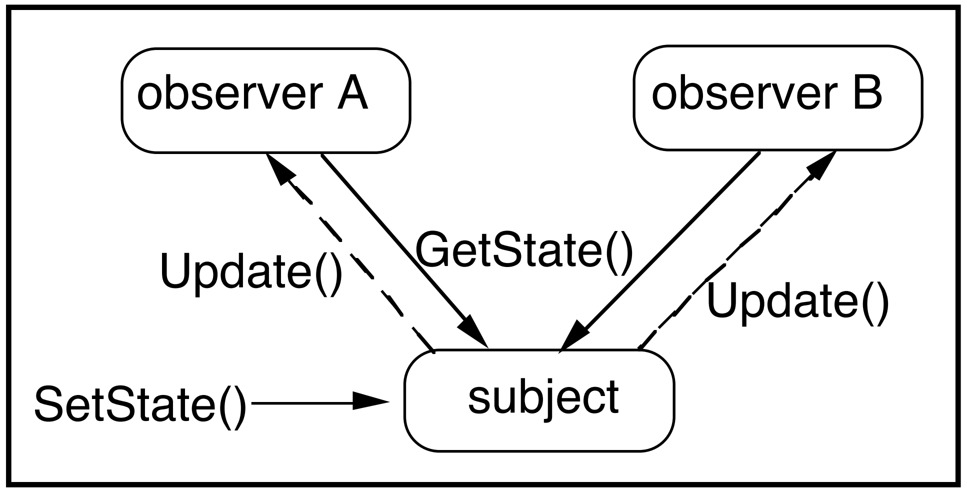
Address Book Example http://developerlife.com/lessons/adapter/default.htm

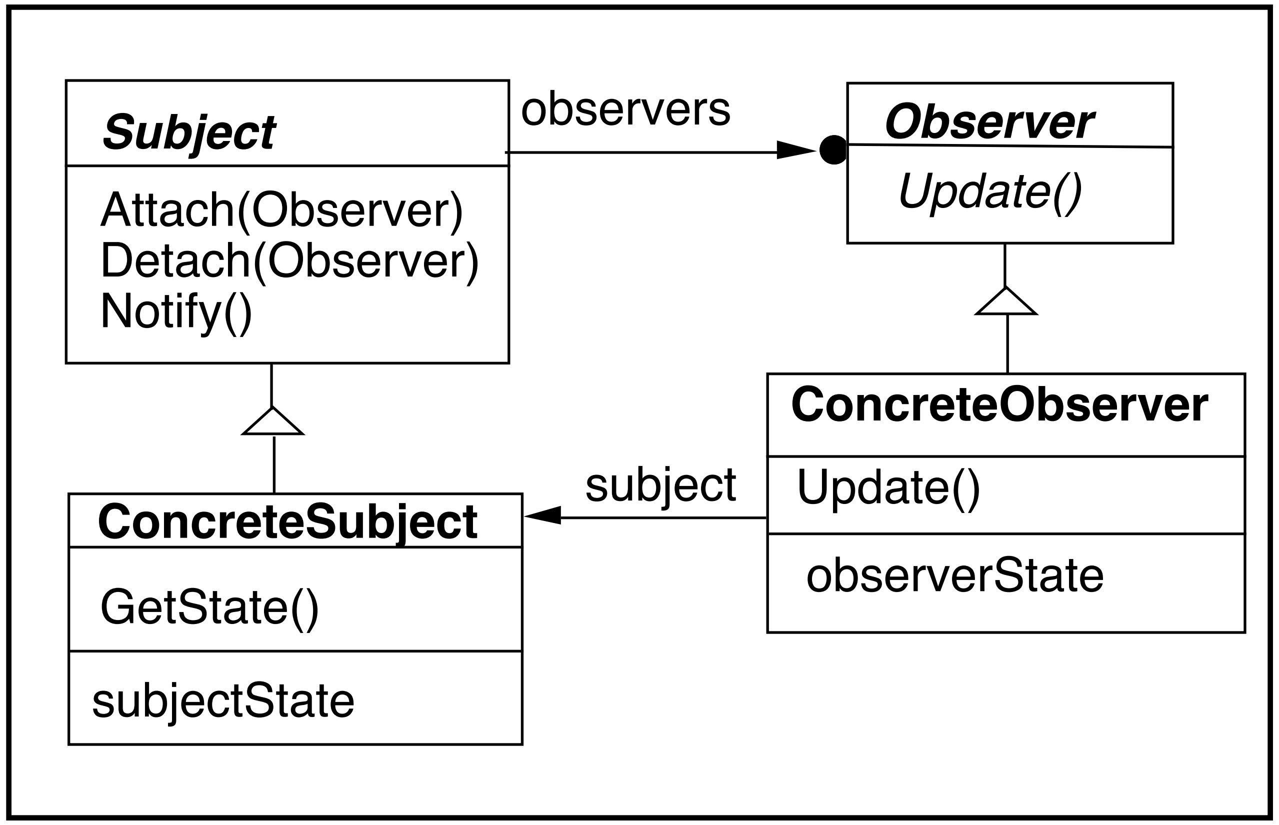
Observer

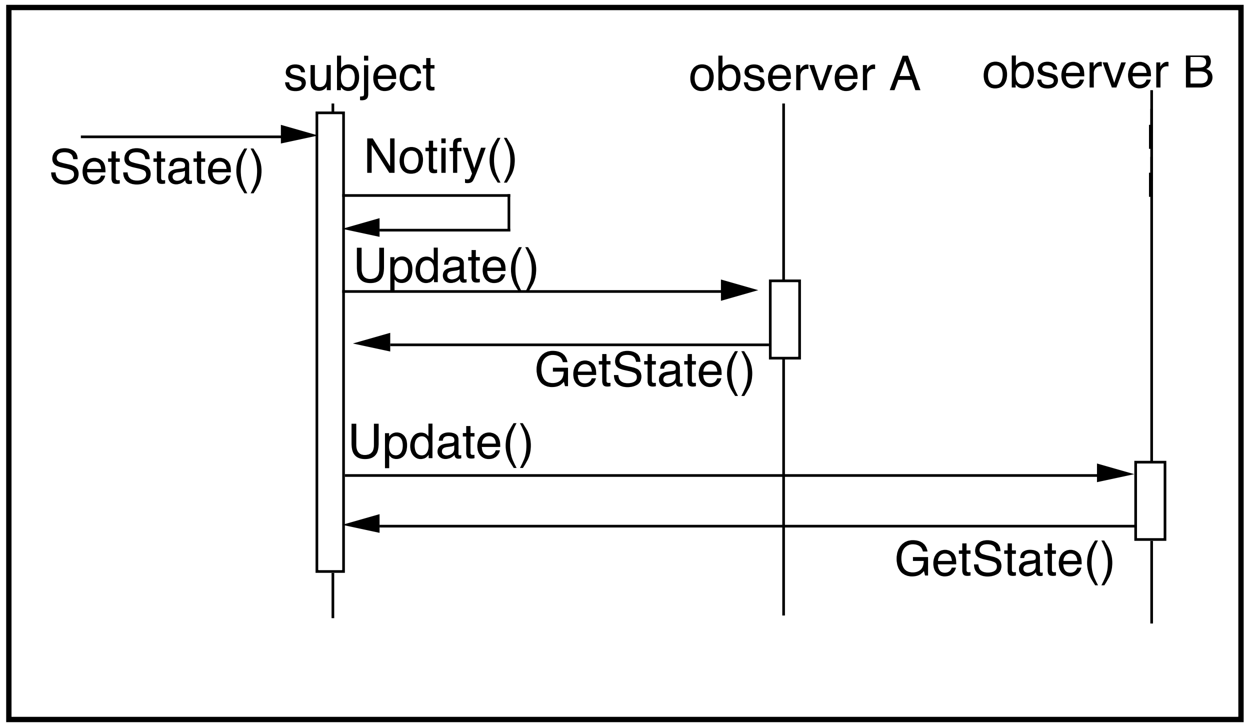
One-to-many dependency between objects

When one object changes state,
  all its dependents are notified and updated automatically

Structure


observer A
observer B
Update()
GetState()
Update()
SetState()
subject


Subject
Attach(Observer)
Detach(Observer)
Notify()
observers
Observer
Update()
ConcreteObserver
Update()
observerState
subject
ConcreteSubject
GetState()
subjectState


subject
observer A
observer B
SetState()
Notify()
Update()
GetState()
Update()
GetState()

Pseudo Java Example

```
public class Subject {
   Window display;
   public void someMethod() {
      this.modifyMyStateSomeHow();
      display.addText( this.text() );
   }
}
```

Abstract coupling - Subject and Observer

Broadcast communication

Updates can take too long

```
public class Subject {
   ArrayList observers = new ArrayList();

   public void someMethod() {
      this.modifyMyStateSomeHow();
      changed();
   }

   private void changed() {
      Iterator needsUpdate = observers.iterator();
      while (needsUpdate.hasNext() )
         needsUpdate.next().update( this );
   }
}

public class SampleWindow {
   public void update(Object subject) {
      text = ((Subject) subject).getText();
      Thread.sleep(10000).
   }
}
```

Some Language Support

Smalltalk	Java	Ruby	Observer Pattern
Object	Observer		Abstract Observer class
Object & Model	Observable	Observable	Subject class

Smalltalk Implementation

Object implements methods for both Observer and Subject.

Actual Subjects should subclass Model

Java's Observer

Class java.util.Observable

```
void    addObserver(Observer o)
void    clearChanged()
int     countObservers()
void    deleteObserver(Observer o)
void    deleteObservers()
boolean hasChanged()
void    notifyObservers()
void    notifyObservers(Object arg)
void    setChanged()
```

Observable object may have any number of Observers

Whenever the Observable instance changes,
it notifies all of its observers

Notification is done by calling the update() method on all observers.

Interface java.util.Observer

Allows all classes to be observable by instances of class Observer

Java Example

```
class Counter extends Observable  {
        public static final String INCREASE = "increase";
        public static final String DECREASE = "decrease";
        private int count = 0;
        private String label;

        public Counter( String label )    {       this.label = label; }

        public String label()              {  return label; }
        public int value()                 {  return count; }
        public String toString()           {  return String.valueOf( count );}

        public void increase() {
                count++;
                setChanged();
                notifyObservers( INCREASE );
        }

        public void decrease()  {
                count--;
                setChanged();
                notifyObservers( DECREASE );
                }
        }
```

Java Observer

```
class IncreaseDetector implements Observer {
	public void update( java.util.Observable whatChanged,
						java.lang.Object message) {
		if ( message.equals( Counter.INCREASE) )  {
			Counter increased = (Counter) whatChanged;
			System.out.println( increased.label() + " changed to " +
							increased.value());
		}
	}

	public static void main(String[] args) {
		Counter test = new Counter();
		IncreaseDetector adding = new IncreaseDetector();
		test.addObserver(adding);
		test.increase();
}
```

Ruby Example

```
require'observer'

class Counter
  include Observable

  attr_reader :count

  def initialize
    @count = 0
  end
  def increase
    @count += 1
    changed
    notify_observers(:INCREASE)
  end

  def decrease
    @count -= 1
    changed
    notify_observers(:DECREASE)
  end
end
```

```
class IncreaseDetector

  def update(type)
    if type == :INCREASE
      puts('Increase')
    end
  end
end

count = Counter.new()
puts count.count
count.add_observer(IncreaseDetector.new)
count.increase
count.increase
puts count.count
```

Implementation Issues

Mapping subjects(Observables) to observers

Use list in subject
Use hash table

```
public class Observable {
    private boolean changed = false;
    private Vector obs;

public Observable() {
        obs = new Vector();
    }

public synchronized void addObserver(Observer o) {
        if (!obs.contains(o)) {
            obs.addElement(o);
        }
    }
```

Observing more than one subject

If an observer has more than one subject how does it know which one changed?

Pass information in the update method

Deleting Subjects

In C++ the subject may no longer exist

Java/Smalltalk observer may prevent subject from garbage collection

Who Triggers the update?

Have methods that change the state trigger update

```
class Counter extends Observable  {                // some code removed
        public void increase()  {
                count++;
                setChanged();
                notifyObservers( INCREASE );
        }
}
```

Have clients call Notify at the right time

```
class Counter extends Observable  {        // some code removed
                public void increase()  {    count++;   }
}

Counter pageHits = new Counter();
pageHits.increase();
pageHits.increase();
pageHits.increase();
pageHits.notifyObservers();
```

Subject is self-consistent before Notification

```
class ComplexObservable extends Observable {
        Widget frontPart = new Widget();
        Gadget internalPart = new Gadget();

        public void trickyChange() {
                frontPart.widgetChange();
                internalpart.anotherChange();
                setChanged();
                notifyObservers( );
        }
}

class MySubclass extends ComplexObservable {
        Gear backEnd = new Gear();

        public void trickyChange() {
                super.trickyChange();
                backEnd.yetAnotherChange();
                setChanged();
                notifyObservers( );
        }
}
```

Adding information about the change

push models - add parameters in the update method

```
class IncreaseDetector extends Counter implements Observer { // stuff not shown

        public void update( Observable whatChanged, Object message) {
                if ( message.equals( INCREASE) )
                        increase();
        }
}

class Counter extends Observable {          // some code removed
        public void increase()  {
                count++;
                setChanged();
                notifyObservers( INCREASE );
        }
}
```

Adding information about the change

pull model - observer asks Subject what happened

```
class IncreaseDetector extends Counter implements Observer {
        public void update( Observable whatChanged ) {
                if ( whatChanged.didYouIncrease() )
                        increase();
        }
}

class Counter extends Observable {          // some code removed
        public void increase() {
                count++;
                setChanged();
                notifyObservers( );
        }
}
```

Scaling the Pattern

Java Event Model

AWT/Swing components broadcast events to Listeners

JDK1.0 AWT components broadcast an event to all its listeners

A listener normally not interested all events

Broadcasting to all listeners was too slow with many listeners

Java 1.1+ Event Model

Each component supports different types of events:

Component supports

ComponentEvent	FocusEvent
KeyEvent	MouseEvent

Each event type supports one or more listener types:

MouseEvent

MouseListener	MouseMotionListener

Each listener interface replaces update with multiple methods

MouseListener

mouseClicked()	mouseEntered()
mousePressed()	mouseReleased()

Listeners

- Only register for events of interest
- Don't need case statements to determine what happened

Small Models

Often an object has a number of fields(aspects) of interest to observers

Rather than make the object a subject make the individual fields subjects
- Simplifies the main object
- Observers can register for only the data they are interested in

VisualWorks ValueHolder

Subject for one value

ValueHolder allows you to:

- Set/get the value
  - Setting the value notifies the observers of the change
- Add/Remove dependents

Adapter

Address Book & JTable

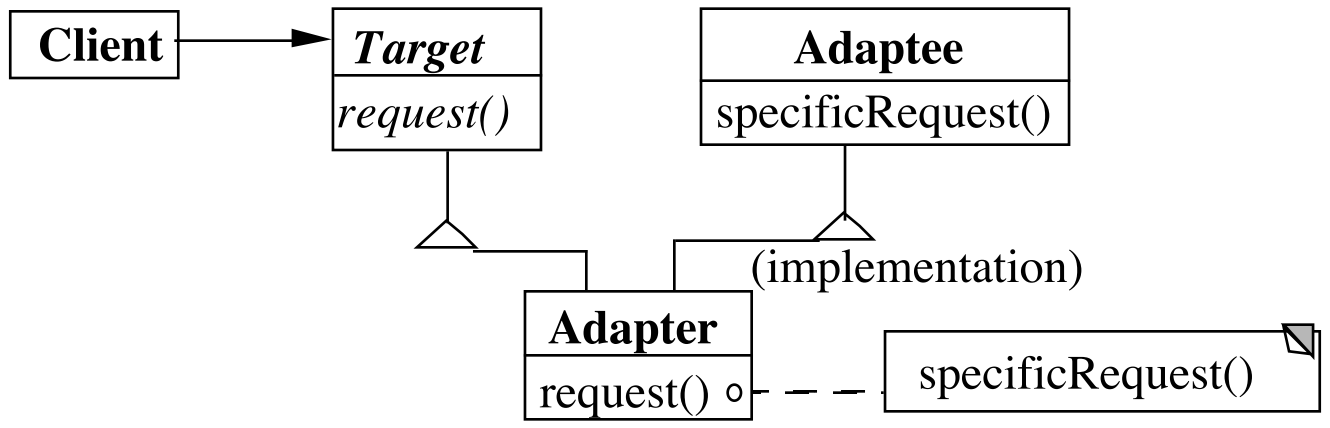
Display an AddressBook object in a JTable

JTables require objects of type TableModel

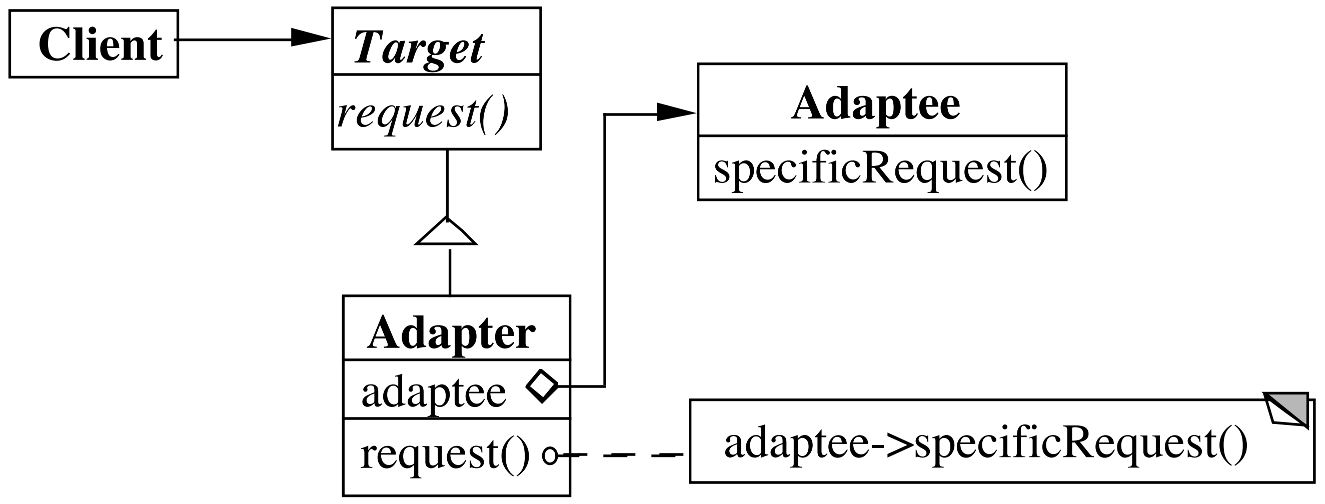
```
public class AddressBook{
    List personList;
    public int getSize(){...}
    public int addPerson(...){...}
    public Person getPerson(...){...}
    ...
}
```

```
public class AddressBookTableAdapter implements TableModel
{
    AddressBook ab;
    public AddressBookTableAdapter( AddressBook ab ){
        this.ab = ab;
    }
    //TableModel impl
    public getRowCount(){
        ab.getSize();

    public Object getValueAt(int rowIndex, int columnIndex) {
        Person requested =
            ad.getPerson(convertRowToName(rowIndex);
        return requested.get(convert(columnIndex));
    }
```

Class Adapter



Object Adapter



Class Adapter Example

```
class OldSquarePeg {
        public: void squarePegOperation()  { do something }
}

class RoundPeg {
        public:  void virtual roundPegOperation = 0;
}

class PegAdapter: private OldSquarePeg,  public RoundPeg  {
        public:
                void virtual roundPegOperation()  {
                        add some corners;
                        squarePegOperation();
                }
}

void clientMethod() {
        RoundPeg* aPeg = new PegAdapter();
        aPeg->roundPegOperation();
}
```

Object Adapter

```
class OldSquarePeg{
        public: void squarePegOperation() { do something }
        }

class RoundPeg      {
        public: void virtual roundPegOperation = 0;
        }

class PegAdapter:  public RoundPeg       {
        private:
                OldSquarePeg* square;

        public:
                PegAdapter()  { square = new OldSquarePeg; }

                void virtual roundPegOperation()     {
                        add some corners;
                        square->squarePegOperation();
                        }
        }
```

How Much Adapting does the Adapter do?

Two-way Adapters

```
class OldSquarePeg {
        public:
                void virtual squarePegOperation() { blah }
}

class RoundPeg {
        public:
                void virtual roundPegOperation() { blah }
}

class PegAdapter: public OldSquarePeg, RoundPeg {
        public:
                void virtual roundPegOperation() {
                        add some corners;
                        squarePegOperation();
                }
                void virtual squarePegOperation() {
                        add some corners;
                        roundPegOperation();
                }
}
```

Flasher and MouseListener

```
class Flasher
    def toggle()
        @flashing = !@flashing
    end

    def pause()
        #etc
    end

    def resume()
        #etc
    end
end
```

```
class MouseListener
    def mouseClicked(event)
    end

    def mouseEntered(event)
    end

    def mouseExited(event)
    end
end
```

mouse click toggles flasher
mouse enter pauses
mouse exits resumes

Flasher as MouseListener

```
class Flasher
   def toggle()
      @flashing = !@flashing
   end

   def pause()
      #etc
   end

   def resume()
      #etc
   end

   def mouseClicked(event)
      toggle()
   end

   def mouseEntered(event)
      pause()
   end

   def mouseExited(event)
      resume()
   end
end
```

Simple Adapter

```
class Flasher
    def toggle()
        @flashing = !@flashing
    end

    def pause()
        #etc
    end

    def resume()
        #etc
    end
end
```

```
class FlasherAdaptor
    def initialize(aFlasher)
        @flasher = aFlasher
    end

    def mouseClicked(event)
        @flasher.toggle()
    end

    def mouseEntered(event)
        @flasher.pause()
    end

    def mouseExited(event)
        @flasher.resume()
    end
end
```

```
yellowFlasher = Flasher.new(yellow, fast)
FlasherAdapter.new(yellowFlasher)
```

A Ruby Adapter - Fowardable

```
class Flasher
   def toggle()
      @flashing = !@flashing
   end

   def pause()
      #etc
   end

   def resume()
      #etc
   end
end
```

```
require 'forwardable'

class FlasherMouseListener
   extend Forwardable

   def initialize()
       @flasher = Flasher.new()
    end
   def_delegator(:@flasher, :toggle, :mouseClick)
   def_delegator(:@flasher, :pause, :mouseEnter)
   def_delegator(:@flasher, :resume, :mouseExit)

end


adaptor = FlasherMouseListener.new()
adaptor.mouseClick()
```

Parameterized Adapter

```
class MouseListenerAdapter
   def initialize(adaptee, clickMethod, enterMethod, exitMethod)
      @adaptee = adaptee
      @clickMethod = clickMethod
      @enterMethod = enterMethod
      @exitMethod = exitMethod
   end

   def mouseClicked(event)
      @adaptee.send(clickMethod)
   end

   def mouseEntered(event)
      @adaptee.send(clickMethod)
   end

   def mouseExited(event)
      @adaptee.send(clickMethod)
   end
end
```

```
yellowFlasher = Flasher.new(yellow, fast)
MouseListenerAdapter.new(
                         yellowFlasher,
                         :toggle,
                         :pause,
                         :resume)
```

Better Parameterized Adapter

```
class MouseListenerAdapter
   def initialize(adaptee, clickLambda, enterLambda, exitLambda)
      @adaptee = adaptee
      @clickLambda = clickLambda
      @enterLambda = enterLambda
      @exitLambda = exitLambda
   end

   def mouseClicked(event)
      @clickLambda.call(adaptee)
   end

   def mouseEntered(event)
       @enterLambda.call(adaptee)
   end

   def mouseExited(event)
       @exitLambda.call(adaptee)
   end
end
```

```
yellowFlasher = Flasher.new(yellow, fast)
MouseListenerAdapter.new(
        yellowFlasher,
        lambda {|flasher | flasher.toggle()},
        lambda {|flasher | flasher.pause()},
        lambda {|flasher | flasher.resume()})
```

What is this lambda?

no name function that remembers it environment

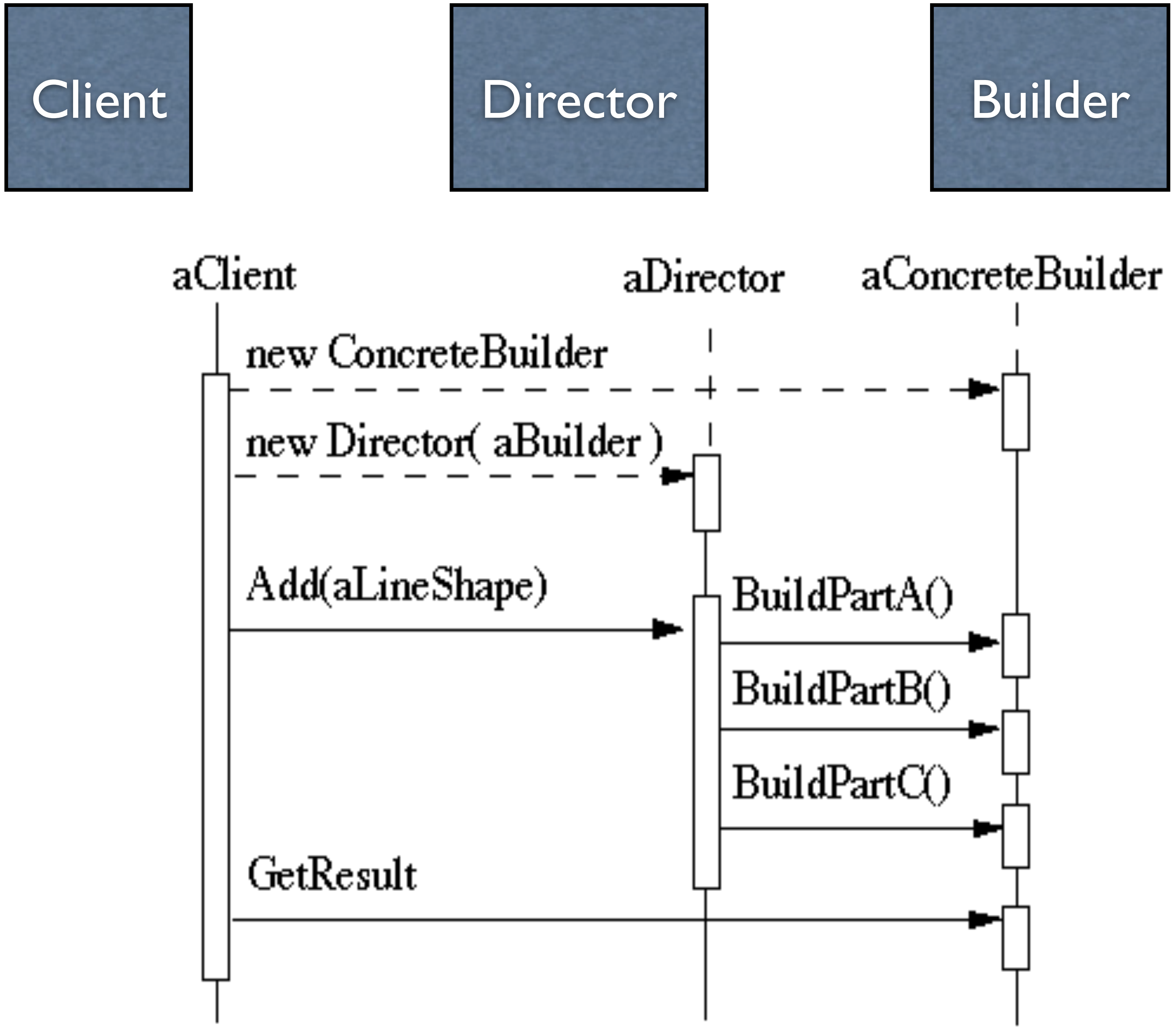
```
a = lambda {|param| puts(param)}
a.call(4)            #4

b = 5
c = lambda {|param| puts(param + b)}
c.call(4)            #9

def hideB(aLambda)
   b = 10
   aLambda.call(4)
end

hideB(c)             #9
```

Builder

Separate construction of a complex object from its representation

So same construction process can create different representations

Builder


Client
Director
Builder
aClient
aDirector
aConcreteBuilder
new ConcreteBuilder
new Director(aBuilder)
Add(aLineShape)
BuildPartA()
BuildPartB()
BuildPartC()
GetResult

RTF Converter

A word processing document has complex structure

How to convert Rich Text Format (RTF) to

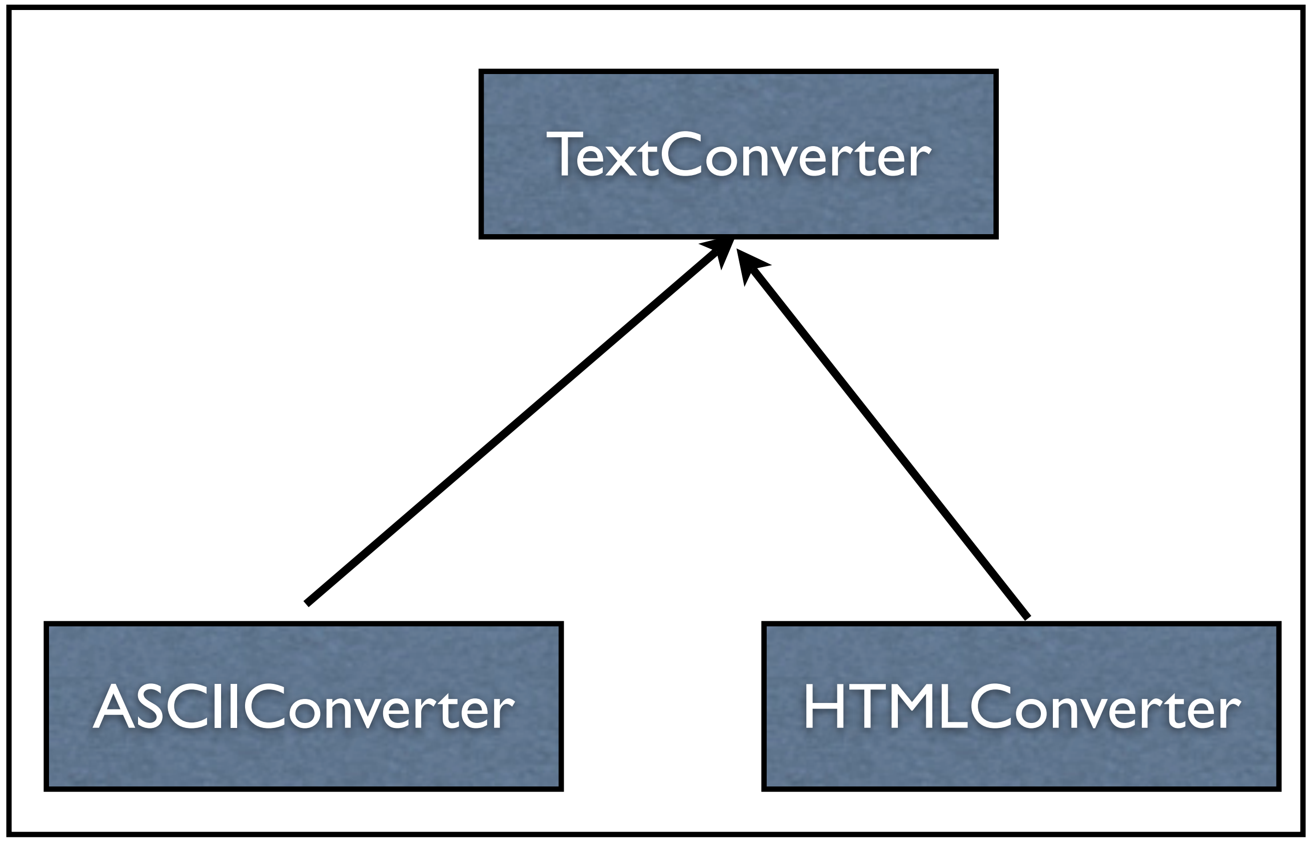
TeX
html
PDF
etc.

Pseudo Solution

```
class RTF_Reader {
   TextConverter builder;
   String RTF_Text;

   public RTF_Reader( TextConverter aBuilder, String RTFtoConvert ){
      builder = aBuilder;
      RTF_Text = RTFtoConvert;
      }

   public void parseRTF(){
      RTFTokenizer rtf = new RTFTokenizer( RTF_Text );

      while ( rtf.hasMoreTokens() ){
         RTFToken next = rtf.nextToken();

         switch ( next.type() ){
            case CHAR:    builder.character( next.char() ); break;
            case FONT:    builder.font( next.font() ); break;
            case PARA:    builder.newParagraph( ); break;
            etc.
         }
      }
   }
}
```

Builder Classes

```
abstract class TextConverter {
   public void character( char nextChar ) {  }
   public void font( Font newFont )   { }
   public void newParagraph() {}
}
```



Sample Program

```
main(){
  ASCII_Converter simplerText = new ASCII_Converter();
  String rtfText;

  // read a file of rtf into rtfText

  RTF_Reader myReader =
    new RTF_Reader( simplerText, rtfText );

  myReader.parseRTF();

  String myProduct = simplerText.getText();
}
```

The Hard Part

The builder interface

Example - Simple API XML (SAX)

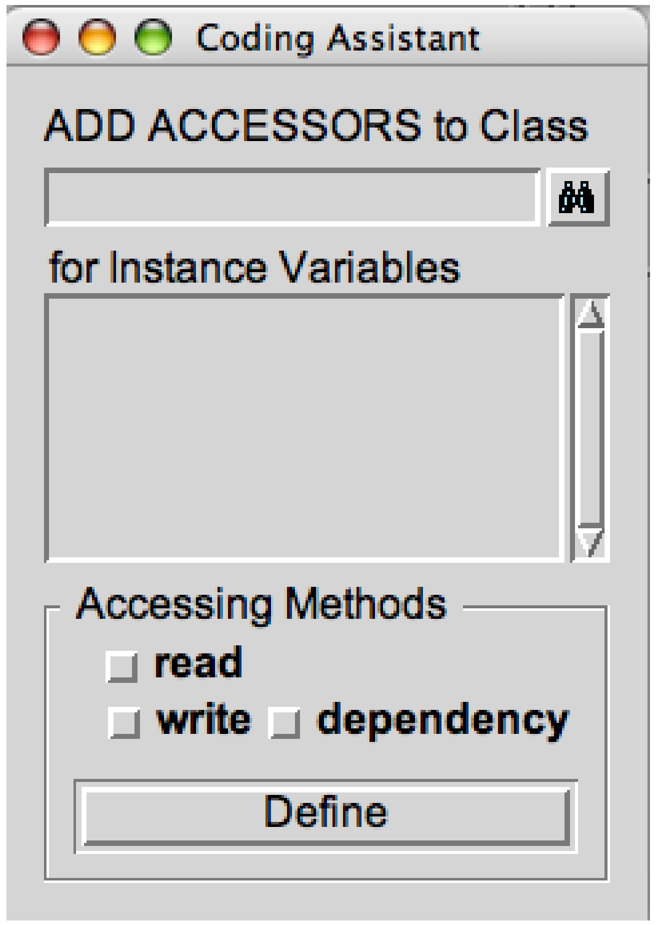
```
public static void main (String args[]) throws Exception {
  XMLReader xr = XMLReaderFactory.createXMLReader();
  MySAXApp handler = new MySAXApp();
  xr.setContentHandler(handler);
  xr.setErrorHandler(handler);

  FileReader r = new FileReader("Foo.xml");
  xr.parse(new InputSource(r));
  handler.getResult();
   }
```

Examples - VW Smalltalk

ClassBuilder
MenuBuilder
UIBuilder

UIBuilder

```
#(#{UI.FullSpec}
    #window:
    #(#{UI.WindowSpec}
      #label: #(#{Kernel.UserMessage} #key: #CodingAssistant
            #defaultString: 'Coding Assistant' #catalogID: #UIPainter)
      #min: #(#{Core.Point} 242 320 )
      #max: #(#{Core.Point} 242 320 )
      #bounds: #(#{Graphics.Rectangle} 279 140 521 460 ) )
    #component:
    #(#{UI.SpecCollection}
      #collection: #(
        #(#{UI.LabelSpec}
          #layout: #(#{Graphics.LayoutOrigin} 14 0 12 0 )
          #label: #(#{Kernel.UserMessage} #key: #ADDACCESSORSToClass
            #defaultString: 'ADD ACCESSORS to Class' #catalogID: #UIPainter) )
        #(#{UI.LabelSpec}
          #layout: #(#{Graphics.LayoutOrigin} 16 0 65 0 )
          #label: #(#{Kernel.UserMessage} #key: #forInstanceVariables
             #defaultString: 'for Instance Variables' #catalogID: #UIPainter) )
```



Strategy
vs
Builder