CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2010
Doc 16 Memento, Composite & Adapter
March 22, 2010

Copyright ©, All rights reserved. 2010 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.

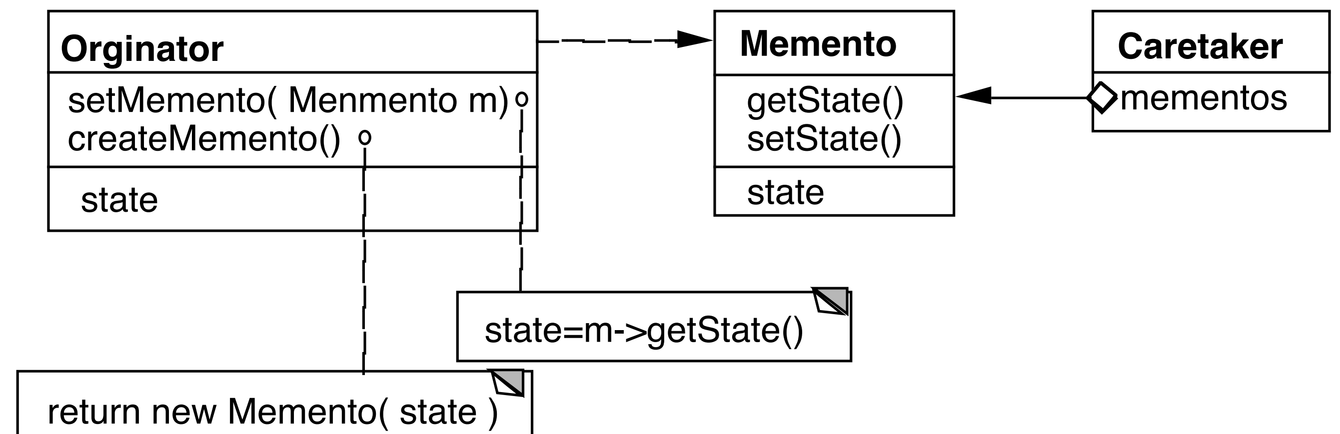
References

Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, 1995, pp. 283-292, 163-174, 139-150

Memento

Store an object's internal state, so the object can be restored to this state later without violating encapsulation

undo, rollbacks



Only originator:

Can access Memento’s get/set state methods

Create Memento

Example

```
package Examples;
class Memento{
	private Hashtable savedState = new Hashtable();

	protected Memento() {}; //Give some protection

	protected void setState( String stateName, Object stateValue ) {
		savedState.put( stateName, stateValue );
	}

	protected Object getState( String stateName) {
		return savedState.get( stateName);
	}

	protected Object getState(String stateName, Object defaultValue ) {
		if ( savedState.containsKey( stateName ) )
			return savedState.get( stateName);
		else
			return defaultValue;
	}
}
```

Sample Originator

```
package Examples;
class ComplexObject {
	private String name;
	private int someData;
	private Vector objectAsState = new Vector();

	public Memento createMemento() {
		Memento currentState = new Memento();
		currentState.setState( "name", name );
		currentState.setState( "someData", new Integer(someData) );
		currentState.setState( "objectAsState", objectAsState.clone() );
		return currentState;
	}

	public void restoreState( Memento oldState) {
		name = (String) oldState.getState( "name", name );
		objectAsState = (Vector) oldState.getState( "objectAsState" );
		Integer data = (Integer) oldState.getState( "someData");
		someData = data.intValue();
	}
```

Why not let the Originator save its old state?

```
class ComplexObject {
        private String name;
        private int someData;
        private Vector objectAsState = new Vector();
        private Stack history;

        public createMemento() {
                Memento currentState = new Memento();
                currentState.setState( "name", name );
                currentState.setState( "someData", new Integer(someData) );
                currentState.setState( "objectAsState", objectAsState.clone() );
                history.push(currentState);
        }

        public void restoreState() {
                Memento oldState = history.pop();
                name = (String) oldState.getState( "name", name );
                objectAsState = (Vector) oldState.getState( "objectAsState" );
                Integer data = (Integer) oldState.getState( "someData");
                someData = data.intValue();
        }
```

Some Consequences

Expensive

Narrow & Wide interfaces - Keep data hidden

```
Class Memento {
public:
    virtual ~Memento();
private:
    friend class Originator;
    Memento();
    void setState(State*);
    State* GetState();
```

```
class Originator {
    private String state;

    private class Memento {
        private String state;
        public Memento(String stateToSave)
            { state = stateToSave; }
   public String getState() { return state; }
        }

        public Object memento()
            { return new Memento(state);}
```

Using Clone to Save State

```
interface Memento extends Cloneable { }

class ComplexObject implements Memento {
	private String name;
	private int someData;

	public Memento createMemento() {
		Memento myState = null;
		try {
			myState =  (Memento) this.clone();
		}
		catch (CloneNotSupportedException notReachable) {
		}
		return myState;
	}

	public void  restoreState( Memento savedState) {
		ComplexObject myNewState = (ComplexObject)savedState;
		name = myNewState.name;
		someData = myNewState.someData;
	}
}
```

What if Protocol

When there are complex validations or
performing operations that make it difficult to restore later

Make a copy of the Originator

Perform operations on the copy

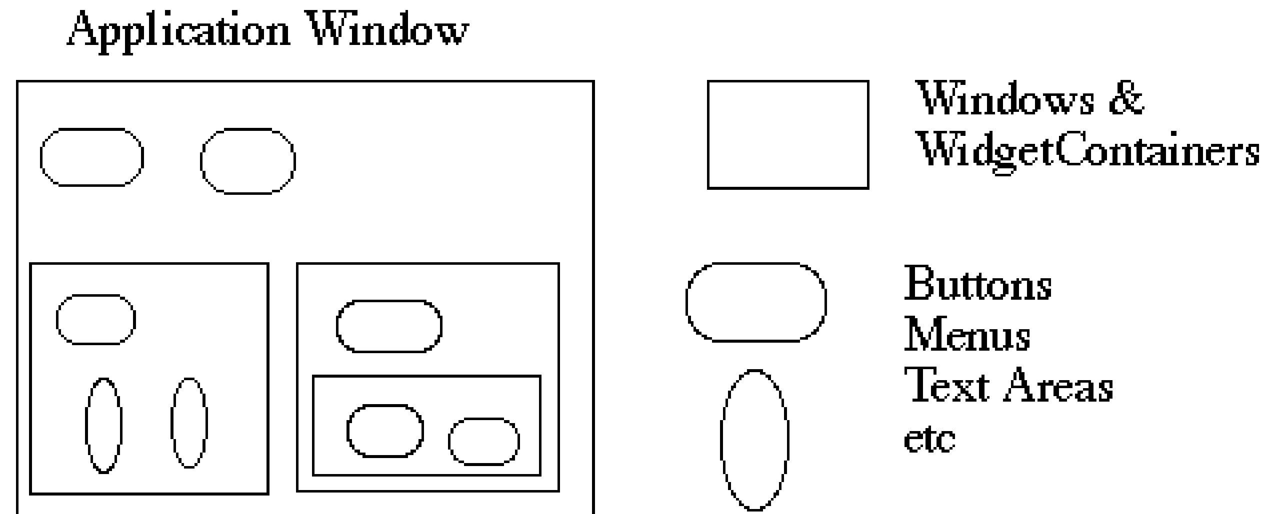
Check if operations invalidate the internal state of copy

If so discard the copy & raise an exception

Else perform the operations on the Originator

Composite

Composite Motivation



How does the window hold and deal with the different items it has to manage?

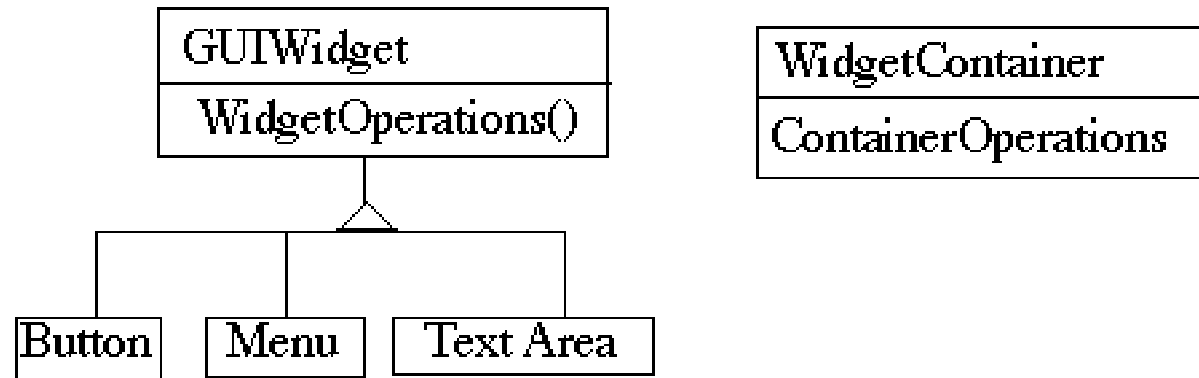
Widgets are different that WidgetContainers

Bad News

```
class Window {
	Buttons[] myButtons;
	Menus[] myMenus;
	TextAreas[] myTextAreas;
	WidgetContainer[] myContainers;

	public void update() {
		if ( myButtons != null )
			for ( int k = 0; k < myButtons.length(); k++ )
				myButtons[k].refresh();
		if ( myMenus != null )
			for ( int k = 0; k < myMenus.length(); k++ )
				myMenus[k].display();
		if ( myTextAreas != null )
			for ( int k = 0; k < myButtons.length(); k++ )
				myTextAreas[k].refresh();
		if ( myContainers != null )
			for ( int k = 0; k < myContainers.length(); k++ )
				myContainers[k].updateElements();
		etc.
	}
	public void fooOperation(){
		if (myButtons != null)
		etc.
```

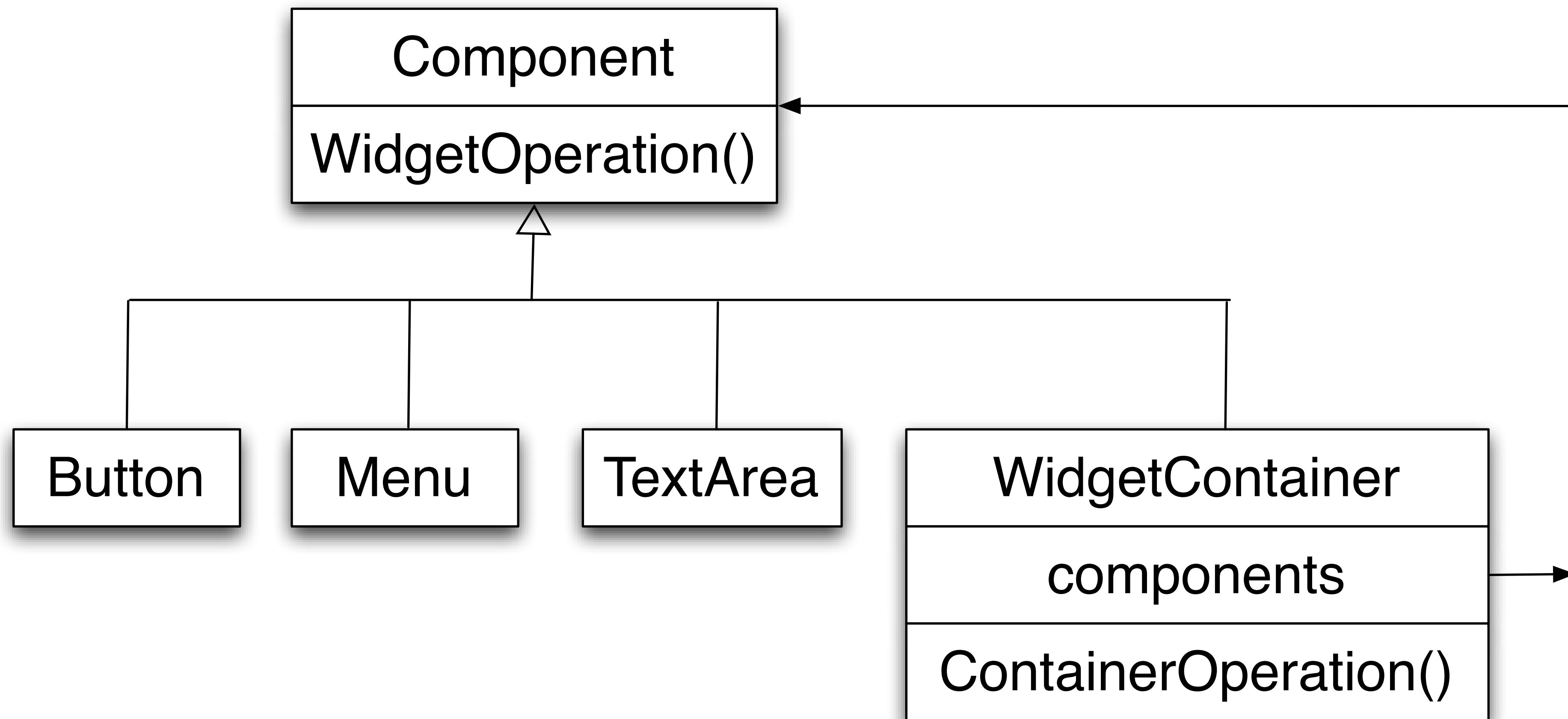
An Improvement




```
class Window {
    GUIWidgets[] myWidgets;
    WidgetContainer[] myContainers;

    public void update(){
        if ( myWidgets != null )
            for ( int k = 0; k < myWidgets.length(); k++ )
                myWidgets[k].update();
        if ( myContainers != null )
            for ( int k = 0; k < myContainers.length(); k++ )
                myContainers[k].updateElements();
        etc.
    }
}
```

Composite Pattern



Composite Pattern

Component implements default behavior for widgets when possible

Button, Menu, etc overrides Component methods when needed

WidgetContainer will have to overrides all widgetOperations

```
class WidgetContainer {
	Component[] myComponents;

	public void update() {
		if ( myComponents != null )
			for ( int k = 0; k < myComponents.length(); k++ )
				myComponents[k].update();
	}
}
```

Issue - WidgetContainer Operations

Should the WidgetContainer operations be declared in Component?

Pro - Transparency

Declaring them in the Component gives all subclasses the same interface

All subclasses can be treated alike. (?)

Con - Safety

Declaring them in WidgetContainer is safer

Adding or removing widgets to non-WidgetContainers is an error

One out is to check the type of the object before using a WidgetContainer operation

Issue - Parent References

```
class WidgetContainer
	{
	Component[] myComponents;

	public void update() {
		if ( myComponents != null )
			for ( int k = 0; k < myComponents.length(); k++ )
				myComponents[k].update();
	}

	public add( Component aComponent ) {
		myComponents.append( aComponent );
		aComponent.setParent( this );
	}
}

class Button extends Component {
	private Component parent;
	public void setParent( Component myParent) {
		parent = myParent;
	}

	etc.
```

More Issues

Should Component implement a list of Components?

- The button etc. will have a useless data member

Child ordering is important in some cases

Who should delete components?

Applicability

Use Composite pattern when you want

To represent part-whole hierarchies of objects

Clients to be able to ignore the difference between compositions of objects and individual objects

Adapter

Address Book & JTable

Display an AddressBook object in a JTable

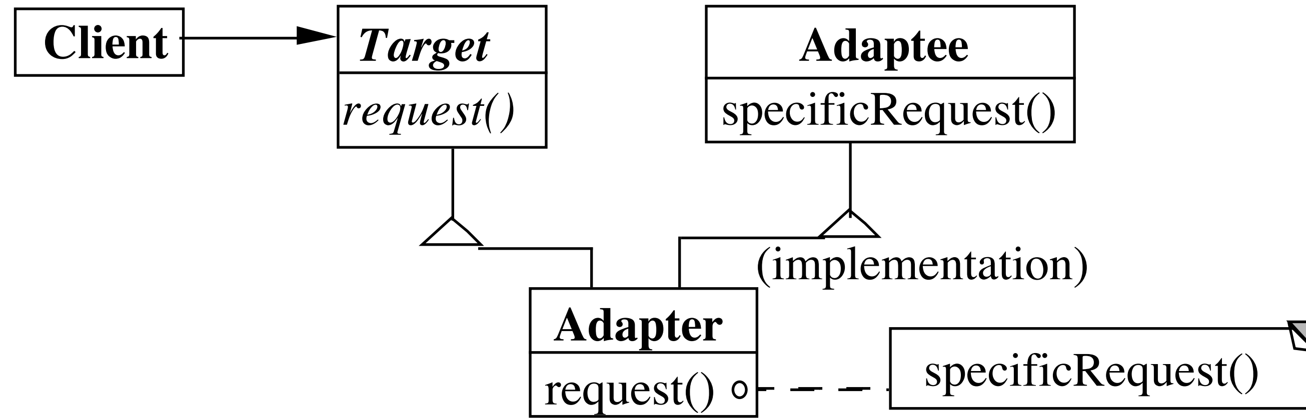
JTables require objects of type TableModel

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

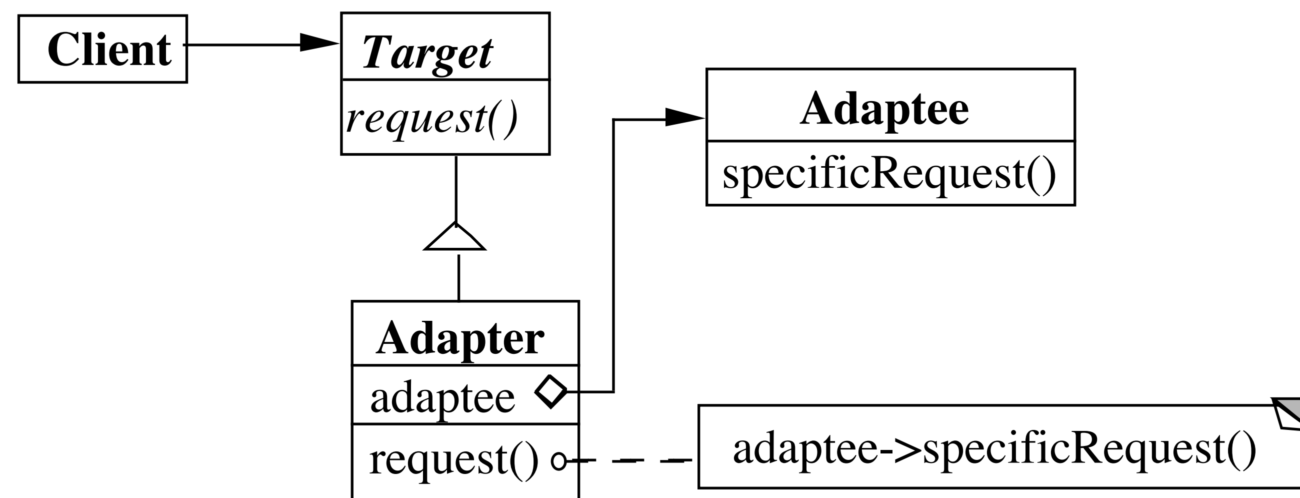
```
public class AddressBookTableAdapter implements TableModel
{
    AddressBook ab;
    public AddressBookTableAdapter( AddressBook ab ){
        this.ab = ab;
    }
    //TableModel impl
    public getRowCount(){
        ab.getSize();

    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
```

Actions we want

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
```