References

Design Patterns: Elements of Resuable Object-Oriented Software,
Gamma, Helm, Johnson, Vlissides, Addison-Wesley, 1995, pp. 207-218, 305-314
Proxy (Surrogate)

a person authorized to act on behalf of another
class Proxy {
    AbstractSubject realSubject;

    public Foo service(Bar x) {
        return realSubject(x);
    }
}
Why do it?
Remote Proxy

String server = getHelloHostAddress( args);
Hello proxy = (Hello) Naming.lookup( server );
String message = proxy.sayHello();
System.out.println( message );
class Proxy {
   AbstractSubject realSubject;

   public Foo service(Bar x ) {
      some preprocessing
      result = realSubject(x);
      some postprocessing
   }
}
Virtual Proxy

Creates/accesses expensive objects on demand
Java's Synchronized List

ArrayList notSafe = new ArrayList();
List threadSafe = Collections.synchronizedList(notSafe);

static class SynchronizedList {
    List list;
    public Object get(int index) {
        synchronized(mutex) {return list.get(index);}  
    }
}
Java's Unmodifiable List

ArrayList notSafe = new ArrayList();
List noChange = Collections.unmodifiableList(notSafe);

static class UnmodifiableList {
    List list;
    public Object get(int index) { return list.get(index);}

    public Object set(int index, Object element) {
        throw new UnsupportedOperationException();
    }
}
Proxy or Decorator?

ArrayList notSafe = new ArrayList();
List noChange = Collections.unmodifiableList(notSafe);
List threadSafe = Collections.synchronizedList(noChange);

threadSafe -> noChange -> notSafe
Proxy verses Decorator

"Decorators can have similar implementations as proxies"

Proxy controls access to an object

Decorator adds one or more responsibilities to an object
Smalltalk Proxy Trick

Object subclass: #Proxy
  instanceVariableNames: 'target'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Whitney-Examples'

Class Method

on: anObject
  ^super new target: anObject

Instance Methods

doesNotUnderstand: aMessage
  ^target
    perform: aMessage selector
    withArguments: aMessage arguments

target: anObject
  target := anObject

| wrapper |
wrapper := Proxy on: Transcript.
wrapper open.
wrapper show: 'Hi mom'.

| wrapper |
wrapper := Proxy on: 3.
wrapper + 5.

| wrapper |
wrapper := Proxy on: 'Hi '.
wrapper , ' mom'.
Allow an object to alter its behavior when its internal state changes

The object will appear to change its class
Context

state

request()

State

handle()

ConcreteStateA

handle()

ConcreteStateB

handle()
Oracle seer = new Truthful();
seer.willThereBeAFeeIncreaseNextYear();
seer = new Lying();
seer.willThereBeAFeeIncreaseNextYear();
public class Oracle {
    private final TRUTH = "truth";
    private final LIE = "lie";
    private final RANDOM = "random";

    String state = TRUTH;

    public boolean willThereBeAFeeIncreaseNextYear() {
        if (state == TRUTH) {
            blah
        } else if (state == LIE) {
            more blah
        } else if (state == RANDOM) {
            random blah
        }
    }
}
```java
class Oracle {
    private State mode = set mode;

    public boolean willThereBeAFeeIncreaseNextYear() {
        return mode.willThereBeAFeeIncreaseNextYear();
    }
}
```
Example: SDChat Server

Commands

"available"
"login"
"register"
"nickname"
"startconversation"
"quit"
"waitinglist"
"acceptconversation"
"message"
"rejectconnection"
"endconversation"
Server States

- Start
  - nickname
  - register
- authenticated
  - login
- Conversation
  - message
  - endconversation
- Conversation handshake
  - startconversation
  - rejectconversation
  - return
- Exit
  - quit
  - waitinglist
  - available
  - acceptconversation
public class SDChatServer {

    String handleNickname(String data) {
        if (state != START)
            return someErrorMessage();
        handle the main case
    }

    String handleLogin(String data) {
        if (state != START)
            return someErrorMessage();
        handle main case
    }

    String handleWaitinglist(String data) {
        if (state != AUTHENTICATED)
            return someErrorMessage();
        handle main case
    }
}
Who defines state Transitions - Context

class Context {
    private AbstractState state = new StartState();

    public Bar foo(int x) {
        int result = state.foo(x);
        if (someConditionHolds() )
            state = nextState();
        return result;
    }
}
class Context {
    private AbstractState state = new StartState();

    public void foo(int x) {
        state = state.foo(x);
    }

    What if foo returns a value?
Who defines state Transitions - States

class Context {
    private AbstractState state = new StartState();

    public int foo(int x) {
        return state.foo(x, this);
    }

    protected void setState(AbstractState newState) {
        state = newState;
    }
}
Sharing State Objects

Stateless state
  State objects without fields
  Can be shared by multiple contexts

Can store date in context and pass as arguments

Large number of state transitions can be expensive

  Only create state once & reuse same object
class Truthful extends Oracle {

    public boolean foo(int x) {
        int result = state.foo(x);
        this.changeClassTo(Random);
        return result;
    }
}
State Verses Strategy
Rate of Change

**Strategy**
Context usually contains just one strategy object

**State**
Context often changes state objects
State Verses Strategy

Exposure of Change

**Strategy**
Strategies all do the same thing

Client do not see change in behavior of Context

**State**
States act differently

Client see the change in behavior