Adapter Design Pattern
State Design Pattern

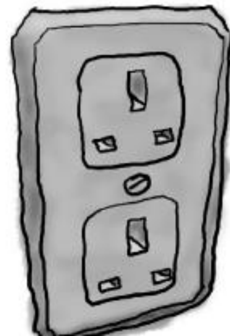
C Sc 335

Rick Mercer

Adapter Design Pattern

- Gang of Four state the intent of Adapter is to
  - *Convert the interface of a class into another interface that the clients expect. Adapter lets classes work together that could not otherwise because of incompatible interfaces.*
- Use it when you need a way to *create a new interface for an object that does the right stuff but has the wrong interface* *Alan Shalloway*

European Wall Outlet



AC Power Adapter

Standard AC Plug

The US laptop expects another interface.

Object Adapters

- Before Java 5.0, we often adapted an ArrayList or HashMap to have an easier to use collection
  - Use a Containment Relationship:
    - A collection with ArrayList or HashMap instance variable
  - Put the cast in the method once instead of everywhere
    - http://www.refactoring.com/catalog/encapsulateDowncast.html
  - Add Employees rather than Objects (*type safe)*
  - Method names then mean more to the clients
    - `Employee getEmployeeWithID(String)` good
    - Object `get(int)` bad
- Not a compelling example with Java generics
  - However, you might see some legacy code with

Object Adapters

- Object Adapters rely on one object (the adapting object) containing another (the adapted object)
- A Stack class should have a Vector and use only Vectors add, get, and size methods (aka Wrapper)
  - Stack should not extend Vector like Sun Oracle does

```
java.lang.Object
    java.util.AbstractCollection<E>
        java.util.AbstractList<E>
            java.util.Vector<E>
                java.util.Stack<E>
```

Class Adapters

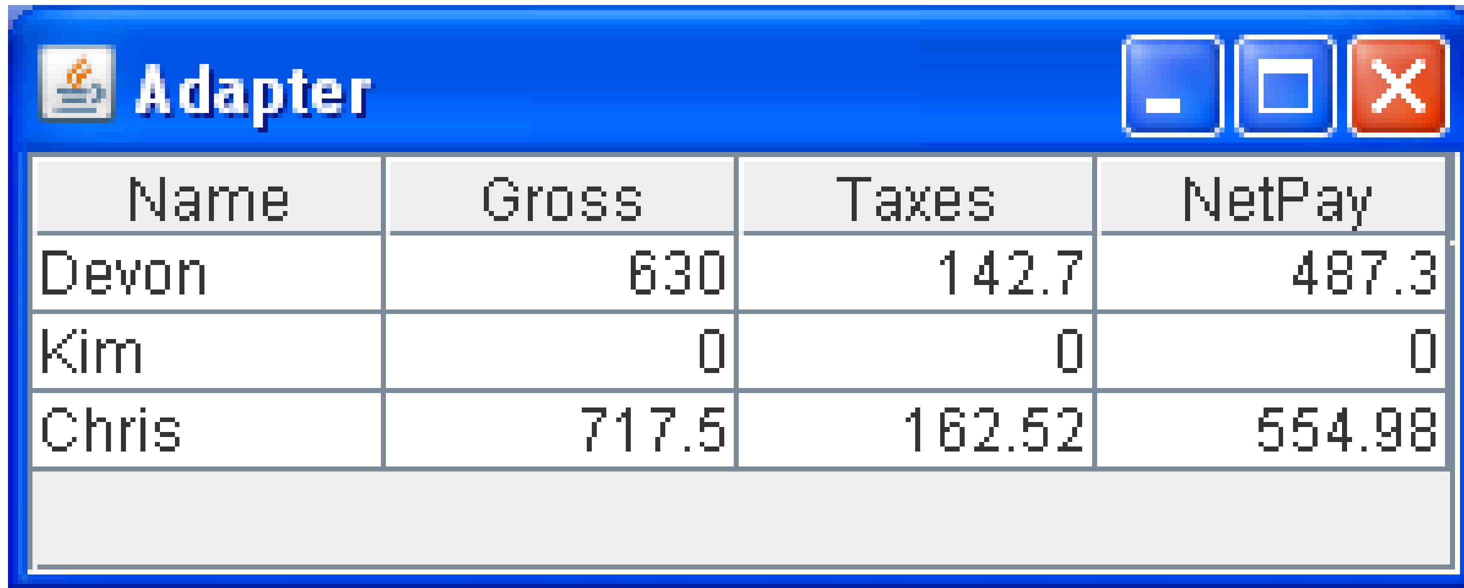
- Class Adapters also come about by extending a class or implementing an interface used by the client code
- You have used class adapters at least twice!
  - Adapted your song collection so it could be stored in a `ListModel` object, which in turn was used by a `JList` to show a graphical view of the list elements
  - `JList` needs the methods defined in the `ListModel` interface: `getSize()` and `getElementAt(int)`

TableModel adapts your model class

- A **`JTable`** requires a **`TableModel`** object that represents a class in model (the data to show)
- Your model class must have methods such as
  - `getColumnCount`, `getRowCount`, `getValueAt`
- Why? **`JTable`** uses these methods to display view
  - Need to adapt our model class to what JTable expects
- Adapt your model class to the interface expected by **`JTable`** by implementing all 10 methods

Adapt my collection to look like TableModel

- JTable shows a list of Employees like this



Name	Gross	Taxes	NetPay
Devon	630	142.7	487.3
Kim	0	0	0
Chris	717.5	162.52	554.98

EmployeeList adapted to TableModel

```
public class EmployeeList implements TableModel {

private ArrayList<Employee> data =
             new ArrayList<Employee>();

  public EmployeeList() {
    data.add(new Employee("Devon", 40, 15.75, 3, "M"));
    data.add(new Employee("Kim", 0, 12.50, 1, "S"));
    data.add(new Employee("Chris", 35, 20.50, 2, "M"));
  }

  public void add(Employee employee) {
    data.add(employee);
  }

  public Iterator<Employee> iterator() {
    return data.iterator();
  }
}
```

Class Adapter

- Code demo: Adapt EmployeeList to the interface the JTable needs by implementing TableModel
  - Or we could have extended DefaultTableModel and overridden the methods (let's choose containment over inheritance)

```
public class EmployeeList implements TableModel {
```

…. Implement TableModel methods ….

okay, to save time, see next slide for getValueAt

One TabelModel method

```
// Adapt tax and pay methods to getValueAt(int column)
public Object getValueAt(int rowIndex, int columnIndex) {
  Employee currentEmployee = data.get(rowIndex);
  double totalTaxes =   currentEmployee.incomeTax()
                      + currentEmployee.medicareTax()
                      + currentEmployee.socialSecurityTax();
  switch (columnIndex) {
  case 0:
    return currentEmployee.getName();
  case 1:
    return currentEmployee.grossPay();
  case 2:
    return totalTaxes;
  case 3:
    return data.get(rowIndex).grossPay() - totalTaxes;
  default:
    return null;
  }
}
```

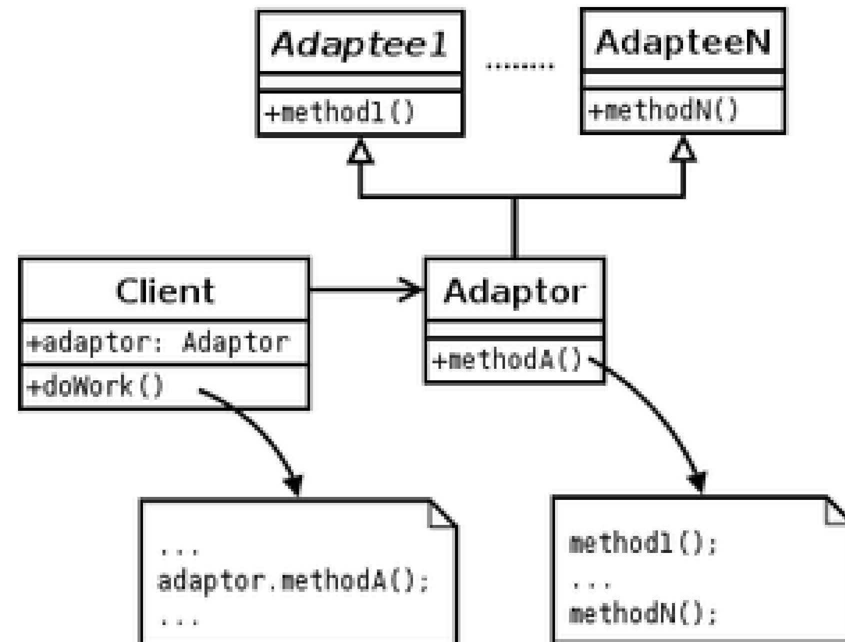
A View: to demonstrate

```
class EmployeeFrame extends JFrame {
  public static void main(String[] args) {
    new EmployeeFrame().setVisible(true);
  }

  private EmployeeList threeEmps;

  public EmployeeFrame() {
    threeEmps = new EmployeeList();
    EmployeeList threeEmps = new EmployeeList();
    setSize(300, 120);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JTable view = new JTable(threeEmps);
    this.add(view, BorderLayout.CENTER);
  }
}
```

- Client: EmployFrame
- Adaptor: JTable
- Adaptee: EmployList




```
class EmployeeFrame extends JFrame {
  public static void main(String[] args) {
    new EmployeeFrame().setVisible(true);
  }

  private EmployeeList threeEmps;

  public EmployeeFrame() {
    threeEmps = new EmployeeList();
    EmployeeList threeEmps = new EmployeeList();
    setSize(300, 120);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JTable view = new JTable(threeEmps);
    this.add(view, BorderLayout.CENTER);
  }
}
```

Adapter Classes

- The WindowListener interface has seven methods that you must implement
- If you only need to respond to one window event, you can extend WindowAdapter (sic)
  - and override whatever methods you need to

```
private class Terminator extends WindowAdapter {
  // This is a WindowAdapter, methods do nothing
  public void WindowClosing(WindowEvent e)   {
    System.exit(0);
  }
  // the other 6 methods are in WindowAdaptor
  // and they are set to do nothing
}
```

Besides WindowListener/WindowAdapter, Java has lots of Listener/Adapter pairs

package java.awt.event

ComponentListener/ComponentAdapter
ContainerListener/ContainerAdapter
FocusListener/FocusAdapter
HierarchyBoundsListener/HierarchyBoundsAdapter
KeyListener/KeyAdapter
MouseListener/MouseAdapter
MouseMotionListener/MouseMotionAdapter
WindowListener/WindowAdapter

package java.awt.dnd

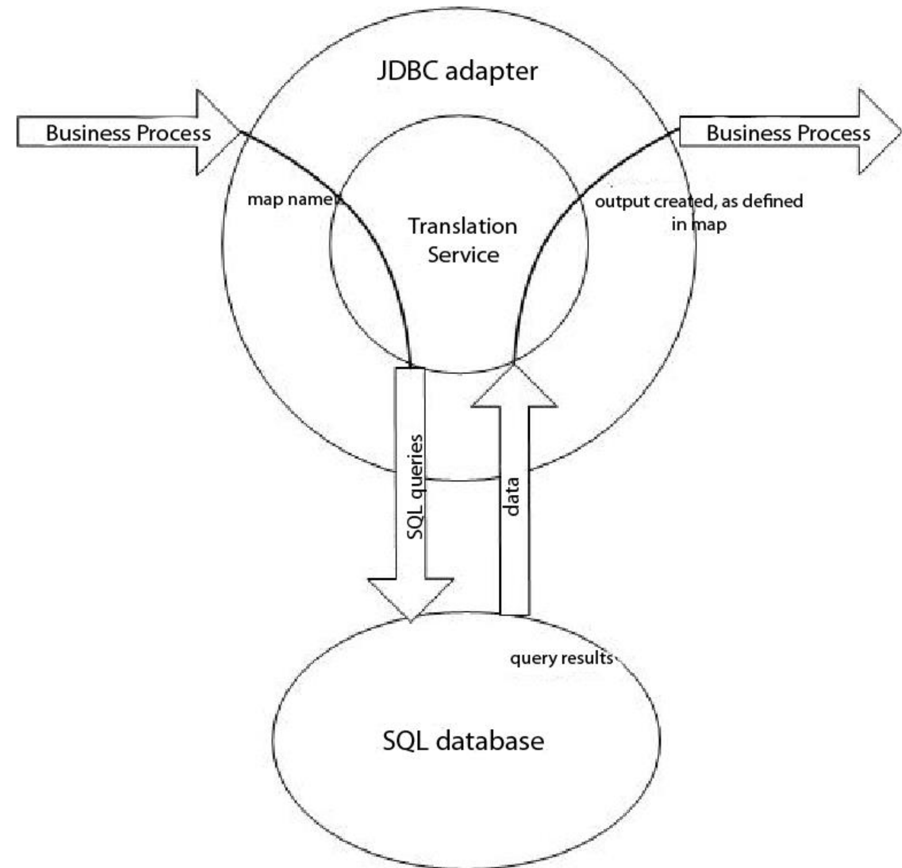
DragSourceListener/DragSourceAdapter
DragTargetListener/DragTargetAdapter

package javax.swing.event

InternalFrameListener/InternalFrameAdapter
MouseInputListener/MouseInputAdapter

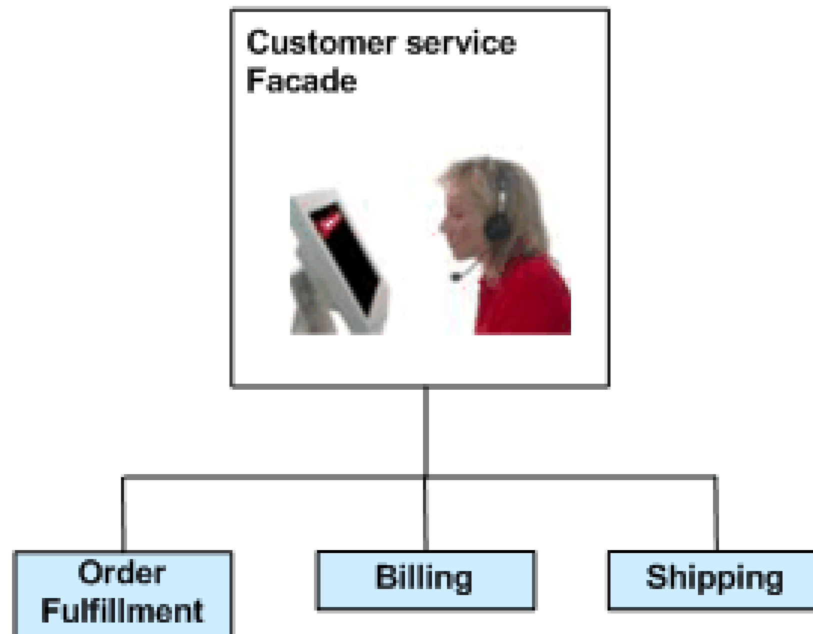
Java Data Base Connectivity (JDBC) Adaptor

- Write code in Java using the methods of the JDBC Adaptor
- The Adaptor creates SQL commands for you



Picture from IBM

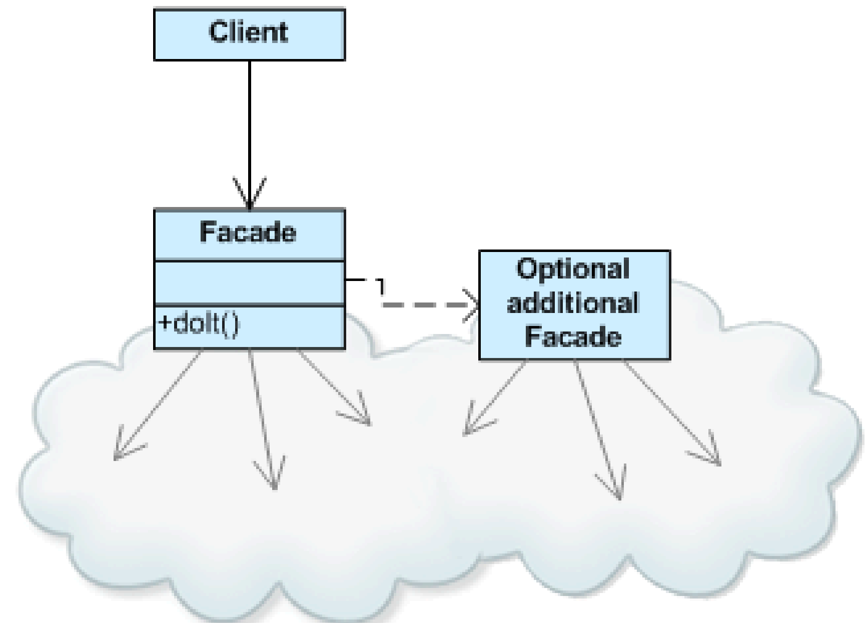
The Façade Design Pattern



Façade is closely related to Adapter

- Provide a unified interface to a set of interfaces in a System. Façade defines a higher level interface that makes the subsystem easier to use *GangOf4*

Facade takes a "riddle wrapped in an enigma shrouded in mystery", and interjects a wrapper that tames the amorphous and inscrutable mass of software. *SourceMaking*



Façade

- Façade is used to
  - Create a simpler interface
  - Reduce the number of objects that a client deals with
  - Hide or encapsulate a large system
- CSc 436 student wants to build a Façade
  - …creating an open source library to introduce people to the power of the OpenCL API. Why?
  - Many people complain about the various intricacies of the "boiler plate" code just to get things working. This library will handle all this for the user so they can focus on learning the techniques of OpenCL.

OpenCL™ is the first open, royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and handheld/embedded devices.

The State Design Pattern

Sate

- Most objects have state that changes
- State can become a prominent aspect of its behavior
- An object that can be in one of several states, with different behavior in each state

Use State when . . .

- Complex if statements determine what to do

```
if (thisSprite == running)
      doRunAnimation();
else if (thisSpite == shooting)
      doShootingAnimination();
else if (thisSpite == noLongerAlive)
      doRollOverAnimation();
      ...
```

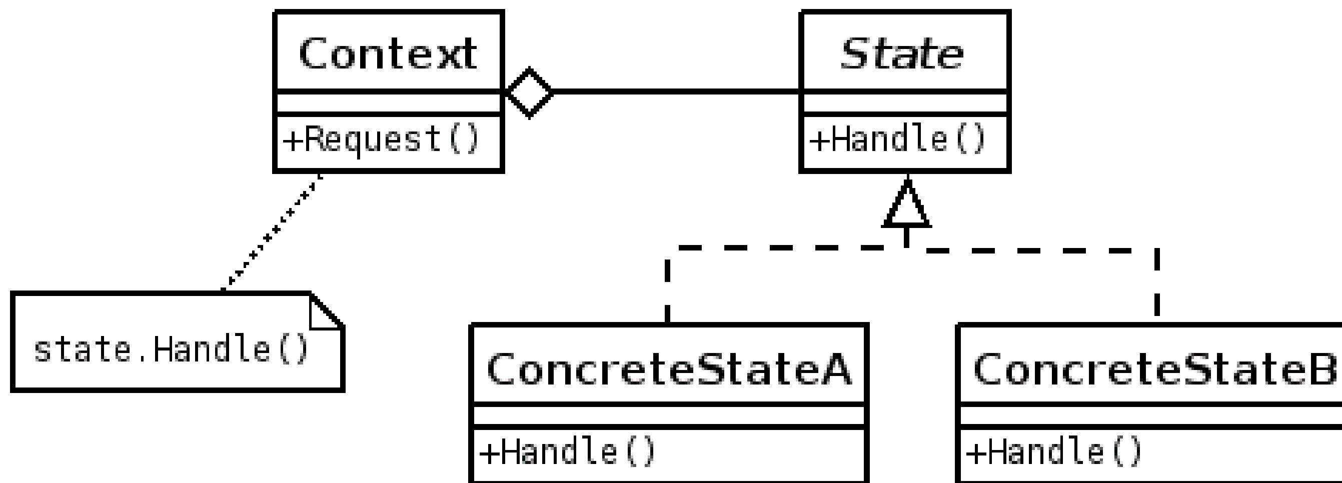
- An object can be in one of several states, with different behavior in each state

State Design Pattern

- State is one of the Behavioral patterns
  - It is similar to Strategy
- Allows an object to alter its behavior when its internal state changes
  - The object will appear to change its class

General Form

from Wikipedia, copied from Gof4

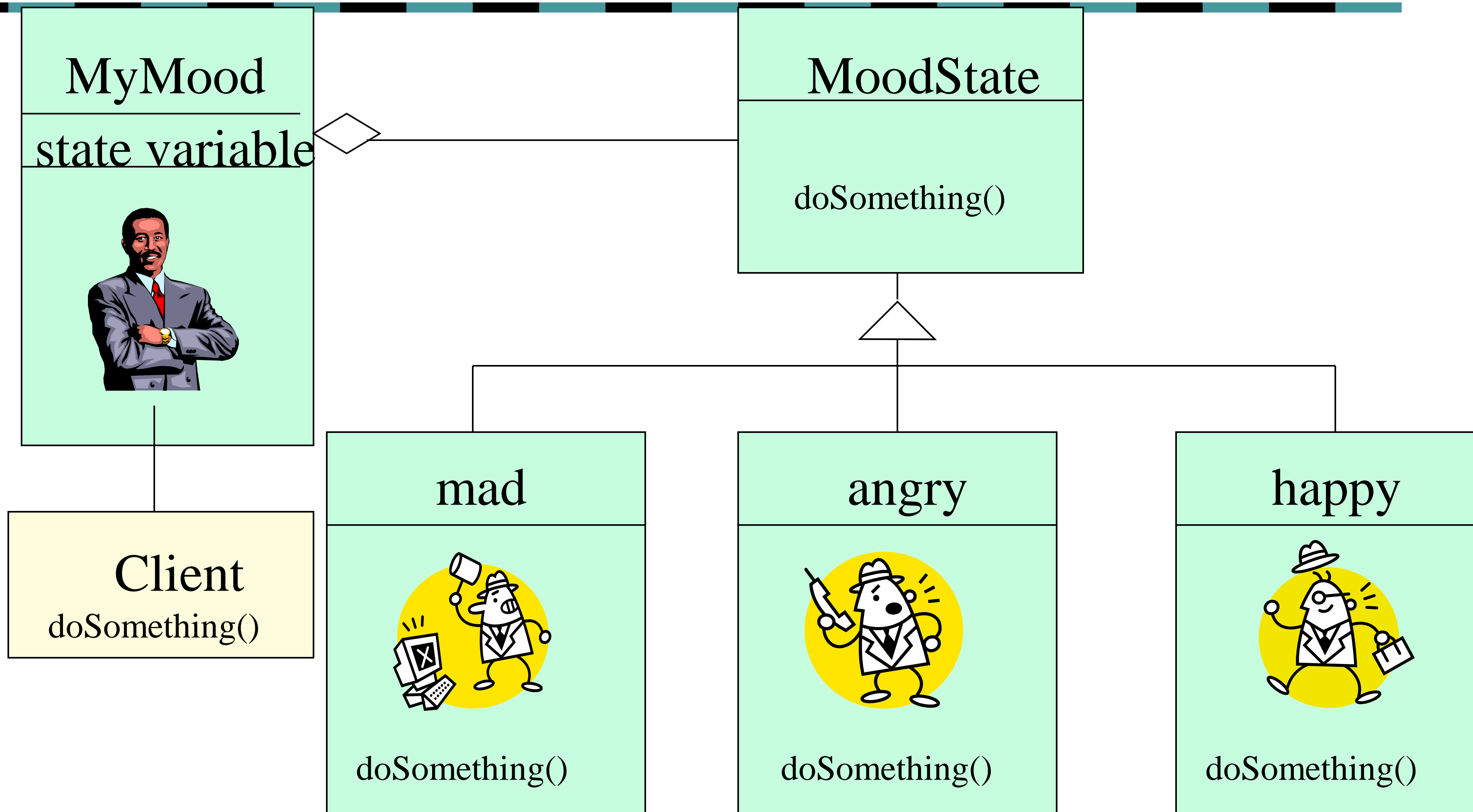


"Context" class: Represents the interface to the outside world

"State" abstract class: Base class which defines the different states of the "state machine"

"Derived" classes from the State class: Defines the true nature of the state that the state machine can be in

Example from Atri Jaterjee

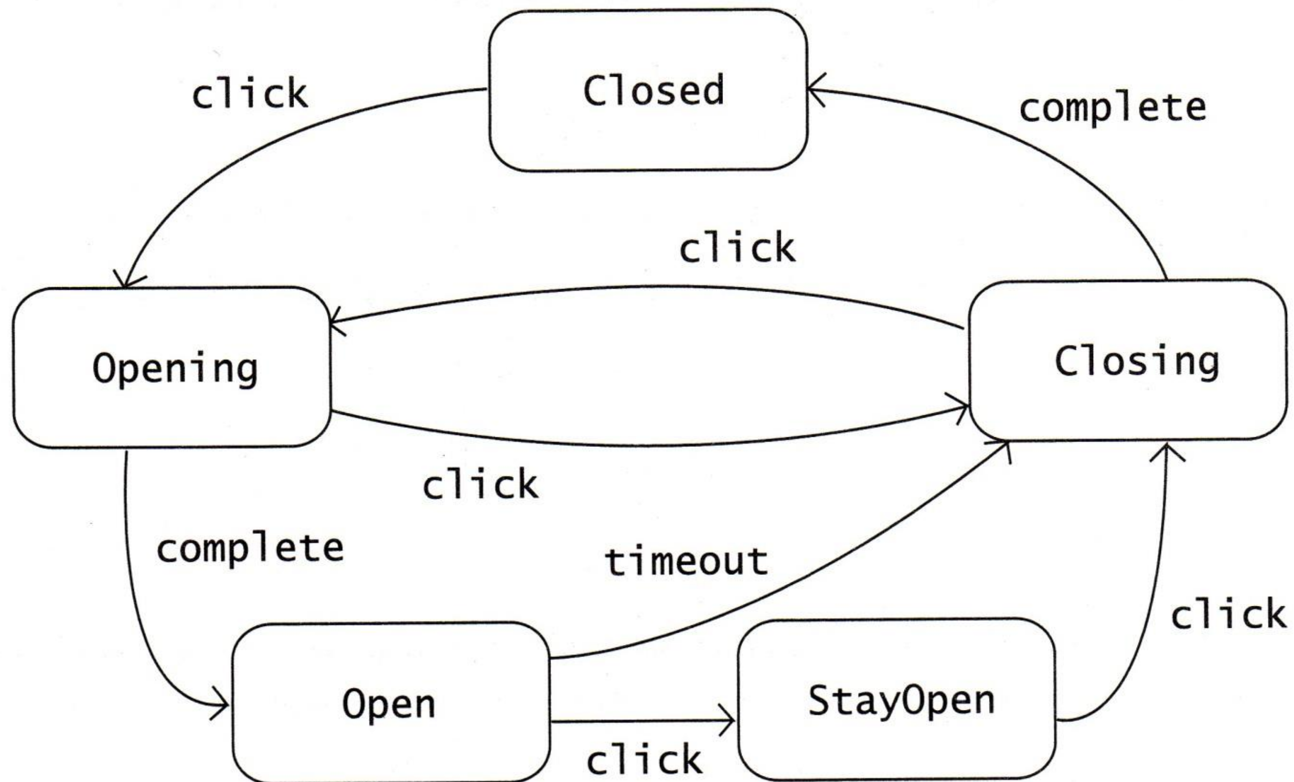


Another Example

from Steve Metsker's Design Patterns Java Workbook, Addison Wesley

- Consider the state of a carousal door in a factory
  - large smart rack that accepts material through a doorway and stores material according to a bar code
  - there is a single button to operate this door
    - if closed, door begins opening
    - if opening, another click begins closing
    - once open, 2 seconds later (timeout), the door begins closing
      - can be prevented by clicking after open state and before timeout begins
- These state changes can be represented by a state machine (next slide)

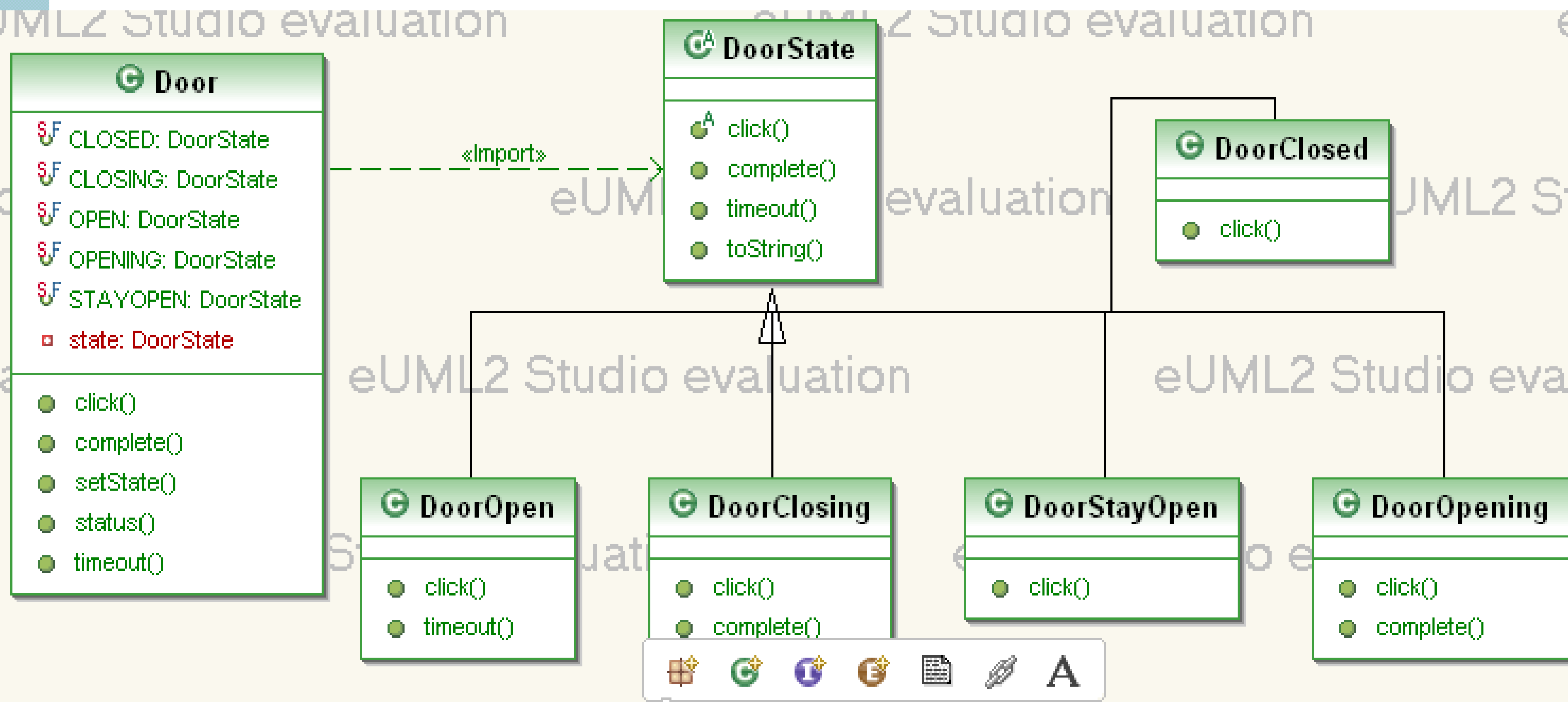
A UML State Diagram



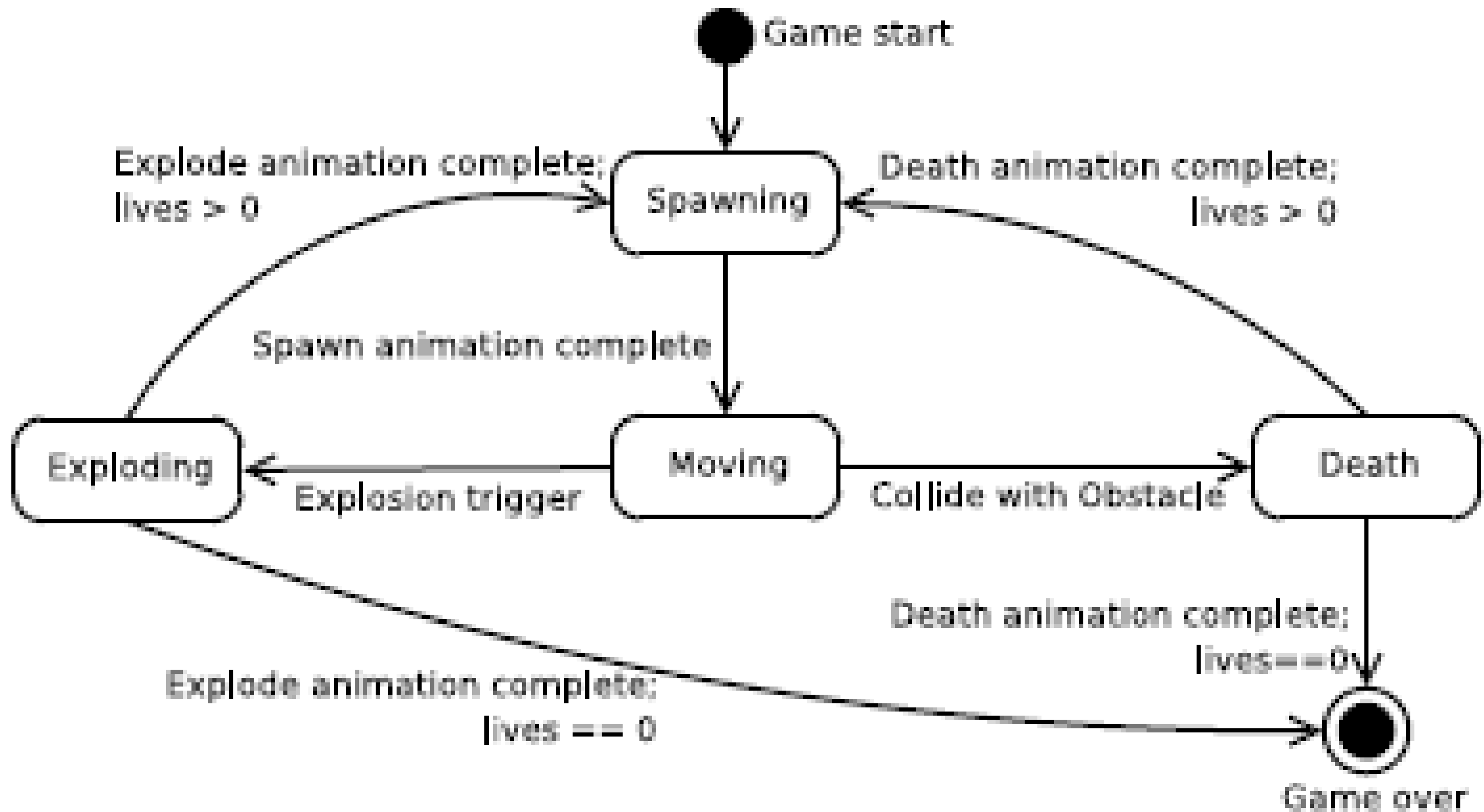
Things to do

- Define a “context” class to present a single interface
- Define a State abstract base class.
- Represent different “states” of the state machine as derived classes of the State base class
- Define state-specific behavior in the appropriate State derived classes (see code demo that changes state, from Opening to Closing or Closing to Opening for example)
- Maintain a reference to the current “state” in the “context” class
- To change the state of the state machine, change the current “state” reference

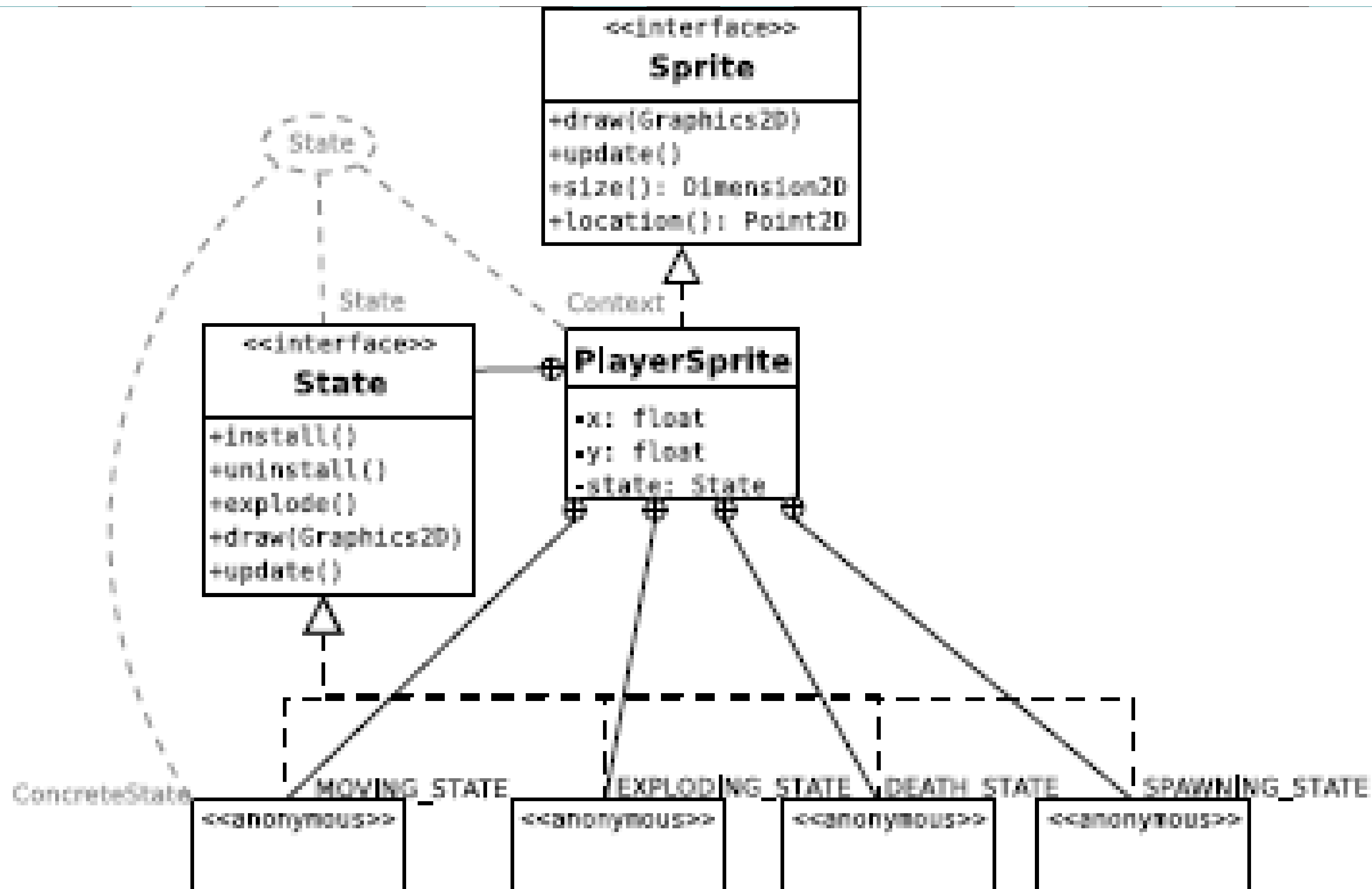
Code reverse engineered (demo)



Another Example
A game



UML diagram of state



Play a game

- See EEClone for ideas about animations and using Strategy