Software Engineering

Lecture 07 – Design Patterns: Part 2

© 2015-20 Dr. Florian Echtler
Bauhaus-Universität Weimar
<florian.echtler@uni-weimar.de>

Today's topics

- Design Patterns part 2:
  - Structural patterns
  - Behavioural patterns
- UI patterns

Structural patterns

- Used to *compose/hide/modify* objects
- Rules-of-thumb: http://www.vincehuston.org/dp/structural_rules.html
- Examples:
  - Adapter/Facade/Proxy
  - Composite
  - Decorator

Adapter

Image source (PD): https://en.wikipedia.org/wiki/AC_power_plugs_and_sockets

- *Adapt* existing class interface to alternative/ new interface
- Real-world example: see below

Adapter: Code example

Source (FU): http://www.vincehuston.org/dp/adapter.html

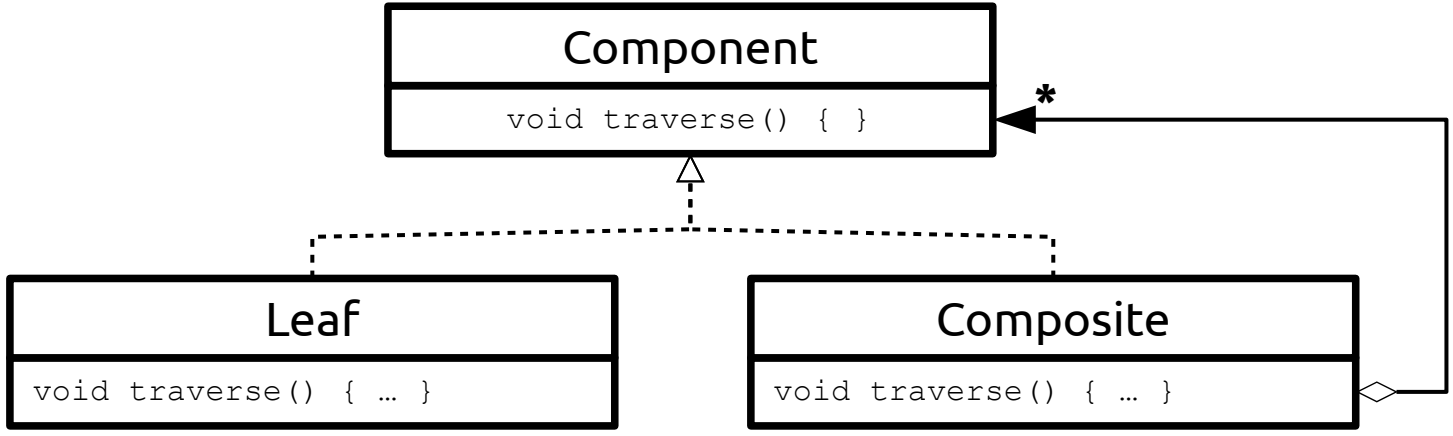
```
class LegacyRectangle {
  public void draw( int x, int y, int w, int h ) {
        System.out.println("rectangle at (" + x + ',' + y
       + ") with width " + w + " and height " + h);
    }
}

// adapter to new Shape interface
class Rectangle extends Shape {
  public Rectangle() {
    adaptee = new LegacyRectangle();
  }
  public void draw( int x1, int y1, int x2, int y2 ) {
    adaptee.draw( Math.min(x1,x2), Math.min(y1,y2),
                  Math.abs(x2-x1), Math.abs(y2-y1) );
  }
  private LegacyRectangle adaptee;
}
```

Adapter: similar patterns

- *Facade* – provide simplified external interface for complex, interrelated set of objects
- *Proxy* – provide same interface with extra functionality:
  - Placeholder for “expensive” objects
  - Local representation for remote object
  - Protective proxy: access control
  - Caching/reference counting (e.g. smart pointer)

Composite

- Useful for representation of tree structures
- E.g. scene graph, GUI widgets, directories, …
- Composite contains components which can be other composites



Composite: Code example

Source (FU): http://www.vincehuston.org/dp/composite.html

```
class DirEntry {
  public void traverse() { }
  protected String m_name;
}
class File extends DirEntry {
  public File( String name ) { m_name = name; }
  public void traverse() { System.out.println(m_name); }
}
class Directory extends DirEntry {
  public Directory( String name ) { m_name = name; }
  public void add( DirEntry obj ) { m_entries.add( obj ); }
  public void traverse() {
    System.out.println(m_name + ":");
    for (DirEntry entry: m_entries) {
      entry.traverse();
    }
  }
  private ArrayList<DirEntry> m_entries;
}
```

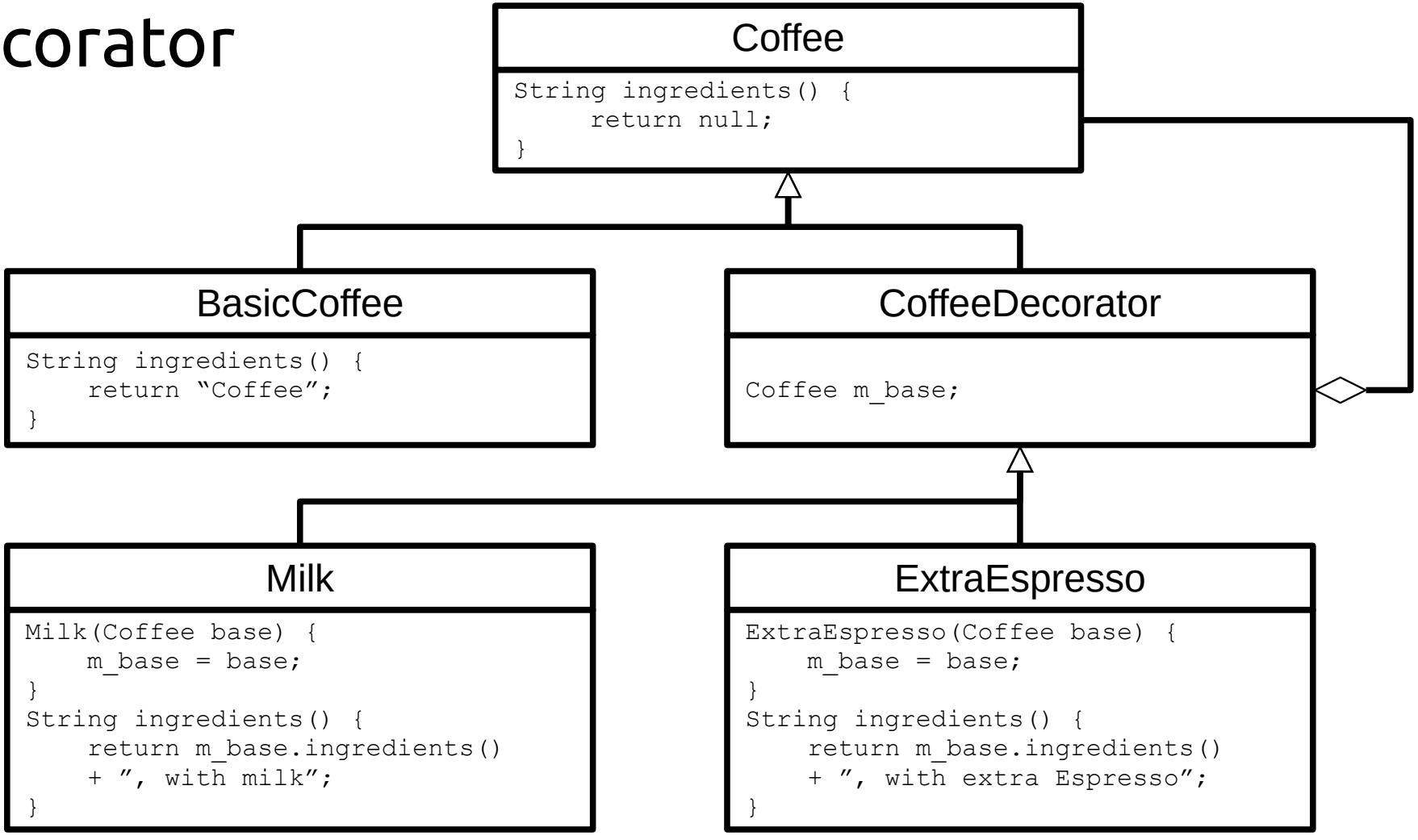
Decorator

- Enables adding extra functionality on the fly
- Important – decorators can be stacked:

```
Stream decoratedStream = new GZIPStream( new
ASCII7Stream( new FileStream("fileName.gz")));
```

- Can't be used to modify the interface
- Structure similar to Composite

Decorator



Decorator

```
Coffee coffeeWithMilk = new Milk( new BasicCoffee() );

System.out.println(coffeeWithMilk->ingredients());
```

→ Coffee, with milk

```
Coffee doubleShotLatte = new ExtraEspresso(
  new ExtraEspresso( coffeeWithMilk ));

System.out.println(doubleShotLatte->ingredients());
```

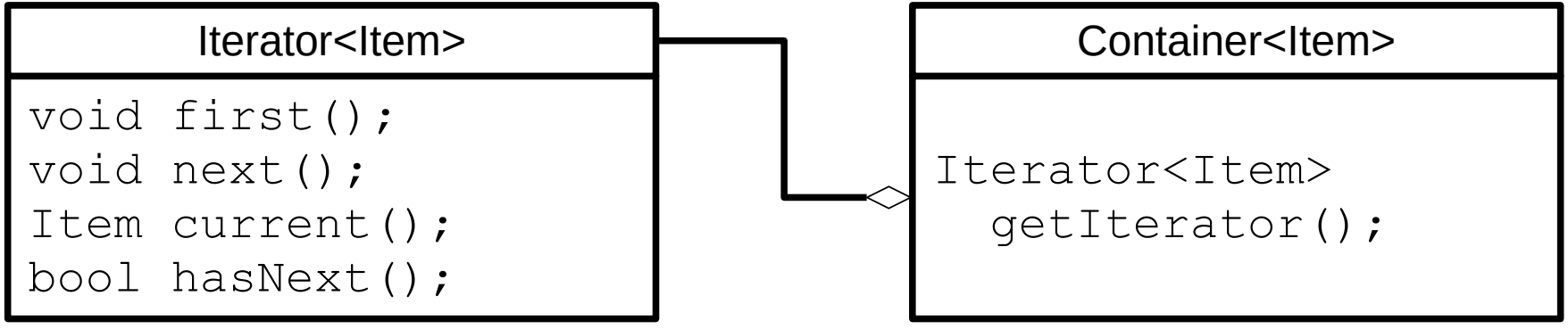
→ Coffee, with milk, with extra Espresso, with extra Espresso

Behavioural patterns

- Used to *control/interact with* objects
- Rules-of-thumb: http://www.vincehuston.org/dp/behavioral_rules.html
- Examples:
  - Iterator
  - Command
  - Visitor
  - Observer

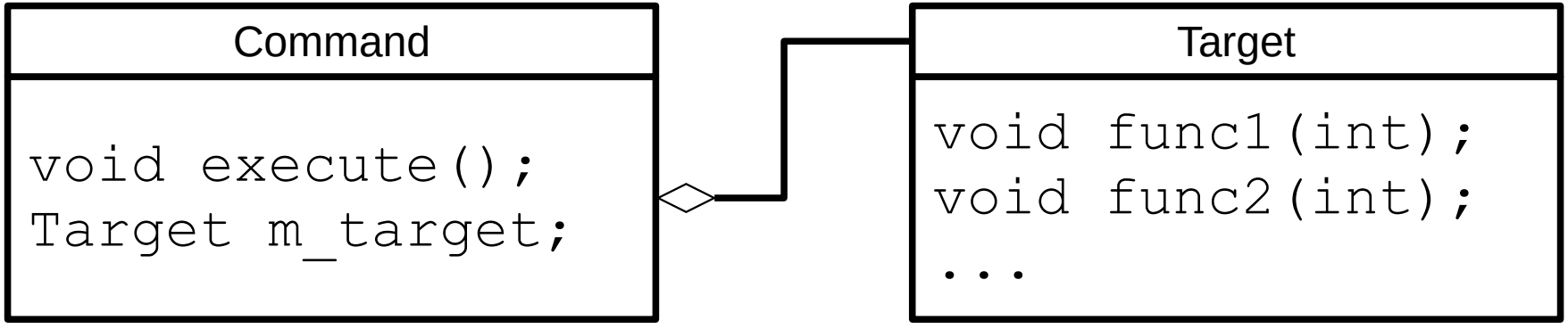
Iterator

- Goal: traverse a collection
- Prominent implementation: Java containers
- Decouples data structures from algorithms
- Iterator object is returned from data structure



Command

- Promotes *method call* to *object*
- Command objects can be passed to other methods and invoked later



Command

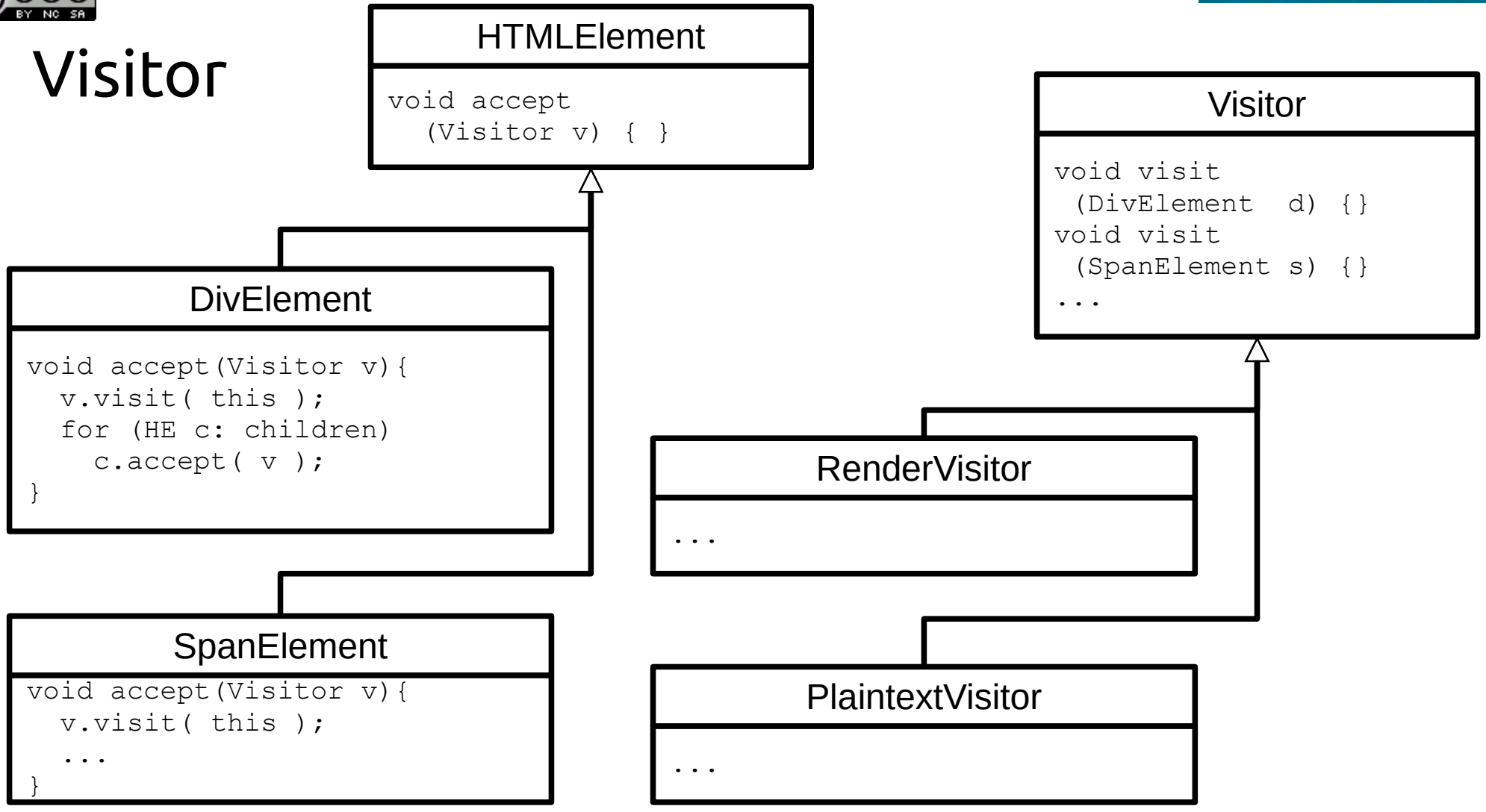
Source (FU): http://www.vincehuston.org/dp/command.html

```
class Target {
  public void func1( int param );
  public void func2( int param );
}

class Command {
  public Command( Consumer<Int> method, int param ) {
    m_method = method; m_param = param;
  }
  public void execute() { m_method.accept(m_param); }

  private Consumer<Int> m_method;
  private int m_param;
}

Target my_target = new Target();
Command cmd = new Command( my_target::func2, 123 );
[...]
cmd.execute();
```

Visitor

- Visitor applies an operation to all objects of an element hierarchy (often recursively)
- Complements the Composite pattern
- Allows to add functionality to elements without changing elements themselves

Visitor

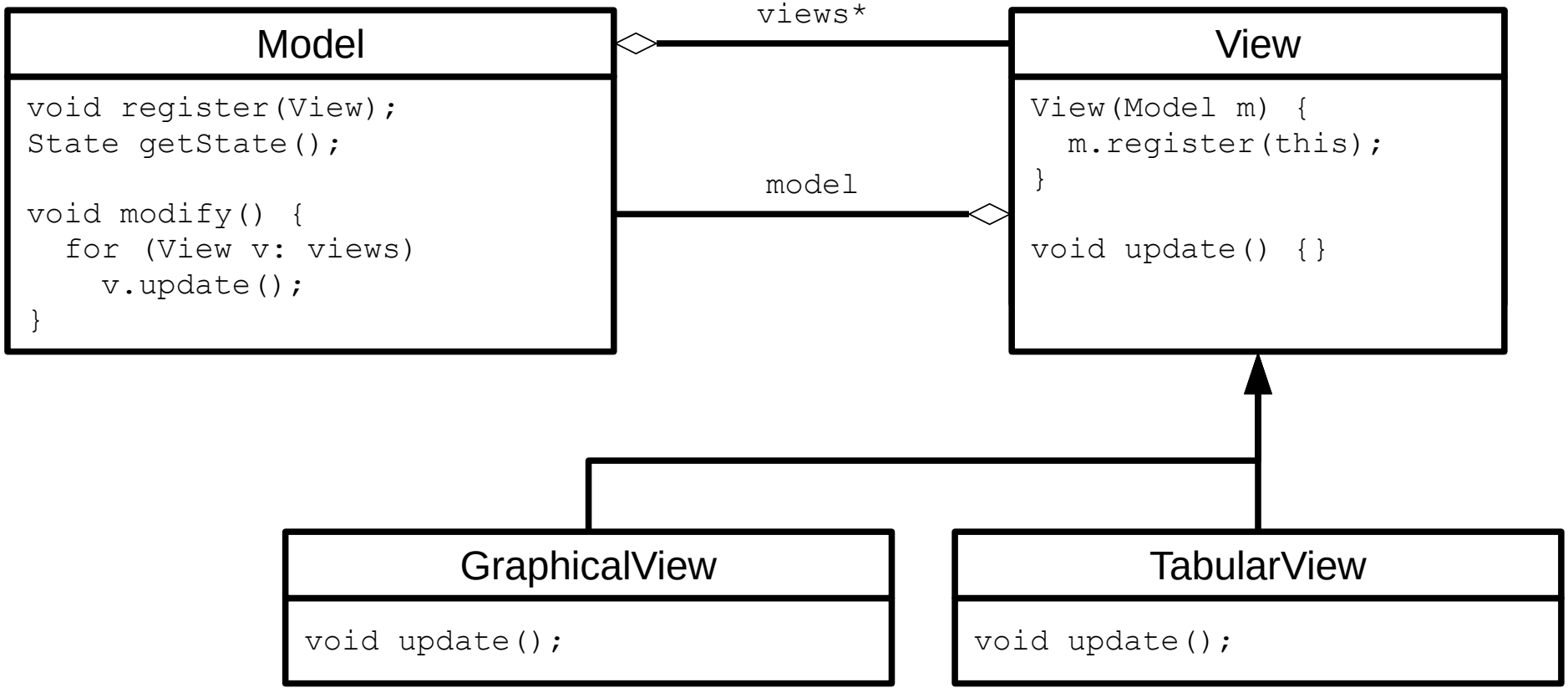


Observer

- Provides a way to notify dependent components if a central state changes
- Often used to implement Model-View-Controller architecture

Observer

Source: http://www.vincehuston.org/dp/observer.html



“Mainloop” Pattern

- *Not* object-oriented, common in many GUI/ graphics libraries
- Library has internal *mainloop*

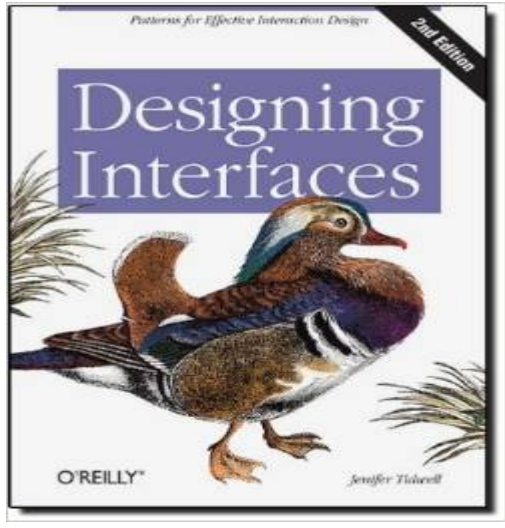
```
void mainloop() {
   while (event = getNextEvent() && !quit)
      process(event);
}
```

- Problem: using multiple libraries?
- Possible solution: expose `getNextEvent() + process()`, write own `mainloop()`

UI Patterns

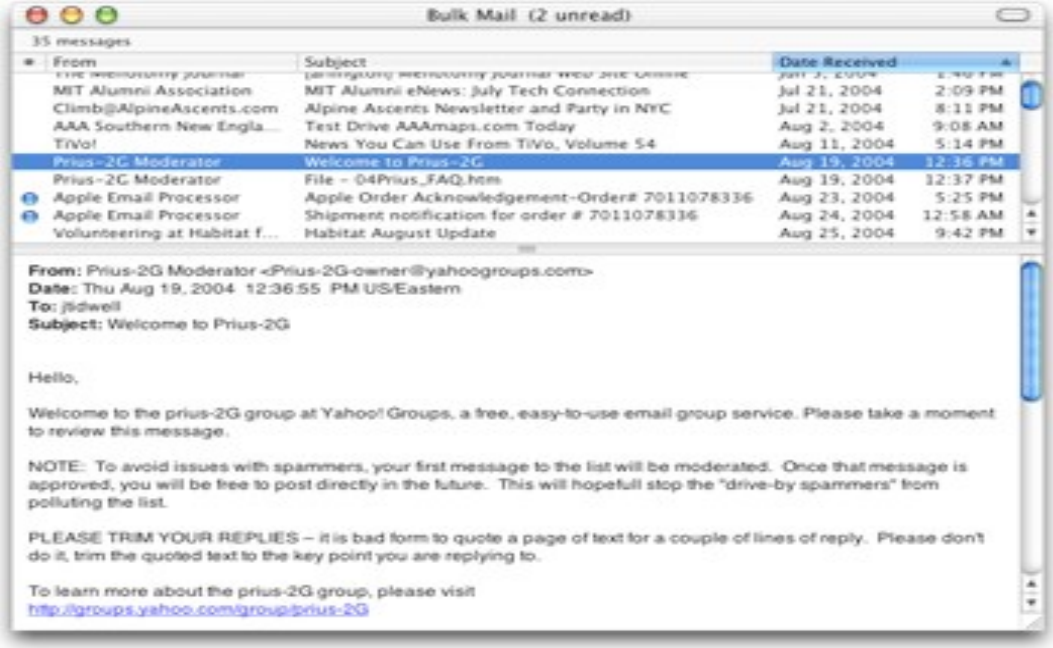
Image source (FU): http://designinginterfaces.com/

- Patterns can also be applied to GUIs
- Don't involve code, but rather layout, colors, widgets, …
- Book reference: “Designing Interfaces”
  by Jenifer Tidwell, O'Reilly Books



UI Patterns: Two-Panel Selector

Image source (FU): “Designing Interfaces”, Jennifer Tidwell



Mac Mail

What: Put two side-by-side panels on the interface. In the first, show a set of items that the user can select at will; in the other, show the content of the selected item.

UI Patterns: One-Window Drilldown

Image source (FU): “Designing Interfaces”, Jennifer Tidwell

Two iPod menus

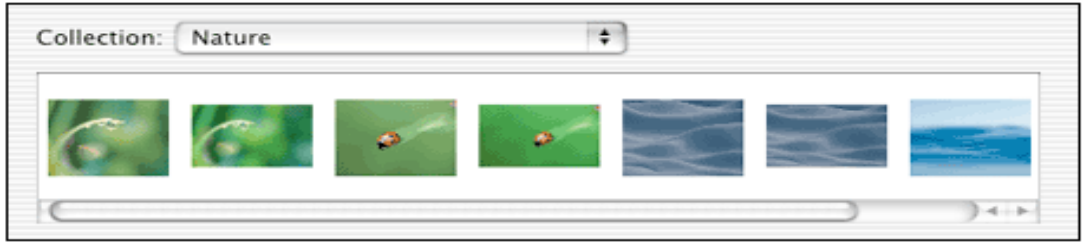
What: Show each of the application's pages within a single window. As a user drills down through a menu of options, or into an object's details, replace the window contents completely with the new page.

Alternative to Two-Panel Selector, useful for:

- Limited display space
- Infrequent usage (e.g. Ubuntu/MacOS settings panel)

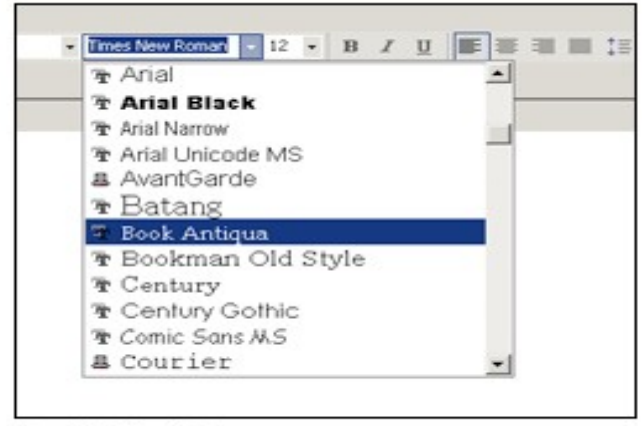
UI Patterns: Illustrated Choices

Image source (FU): “Designing Interfaces”, Jennifer Tidwell



Mac OS X System Properties

What: use pictures instead of words (or in addition to them) to show available choices.



From Word for Windows

UI Patterns: Extras on Demand

Image source (FU): “Designing Interfaces”, Jennifer Tidwell

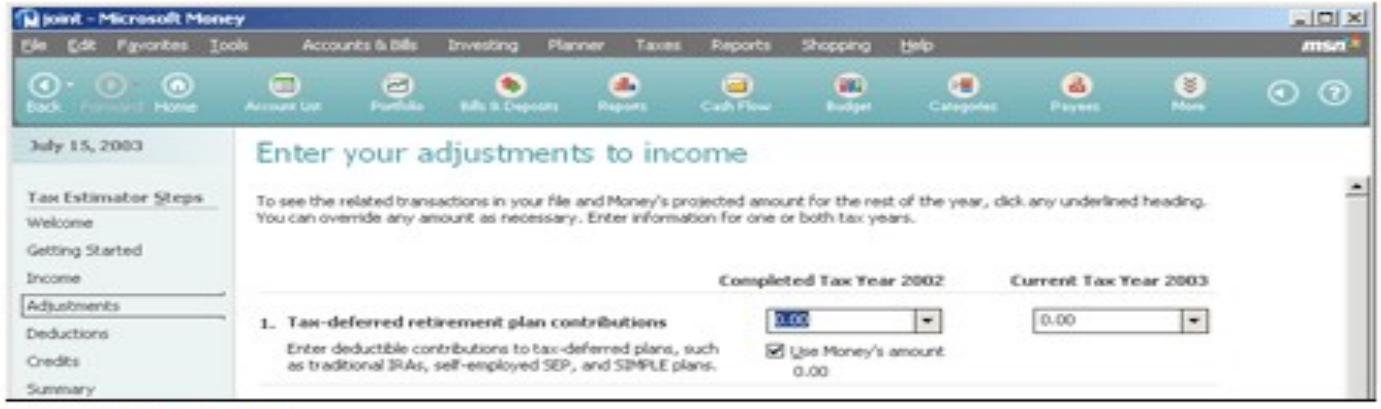
The color dialog box in Windows 2000

What: Show the most important content up front, but hide the rest. Let the user reach it via a single, simple gesture.

Use when: There's too much stuff to be shown on the page, but some of it isn't very important. You'd rather have a simpler UI, but you have to put all this content somewhere.

UI Patterns: Global Navigation

Image source (FU): “Designing Interfaces”, Jennifer Tidwell



From Microsoft Money

What: Using a small section of every page, show a consistent set of links or buttons that take the user to key sections of the site or application.

UI Patterns: Few Hues, Many Values

Image source (FU): “Designing Interfaces”, Jennifer Tidwell

From http://thebanmappingproject.org

What: Choose one, two, or at most three major color hues to use in the interface. Create a color palette by selecting assorted values (brightnesses) from within those few hues.

Questions/Comments?

Image source (FU): http://www.vincehuston.org/dp/

The Periodic Table of Patterns