

CSE 341, Spring 2005

Lecture 27

More OO design

“Classic” C

```
struct Widget {width, height, type, ...type-specific fields...}  
Widget[1000] data;  
x = 0; y = 0;  
for(i = 0; i < n; i++) {  
    if(x+data[i].width > linewidth) {x = 0; y += maxh; maxh = 0}  
    switch ( data[i].type ) {  
        1: drawstring(x,y,data[i].string,...); break;  
        2: drawradio(x,y,data[i].radiolabels,...); break;  
        3: ...  
    }  
    x += data[i].width; maxh = max(maxh,data[i].height)  
}
```

Better C

```
struct Widget {width, height, type, ...type-specific fields...}
Widget[1000] data;
x = 0; y = 0;
for(i = 0; i < n; i++) {
    if(x+data[i].width > linewidth) {x = 0; y += maxh; maxh = 0}
    draw_widget(x,y,data[i]);
    x += data[i].width; maxh = max(maxh,data[i].height)
}
void draw_widget(x,y,theWidget) {
    switch ( theWidget.type ) {
        1: drawstring(x,y,theWidget.string,...); break;
        2: drawradio(x,y,theWidget.radiolabels,...); break;
        3: ...
    }
}
```

Java

```
abstract class Widget {
    int: width, height
    abstract void draw(x,y)
}
class strWidget extends Widget {
    String: string...
    void draw(x,y) {
        drawstr(x,y,string,...)
    }
}
class radioWidget extends Widget {
    String: radiolabels...
    void draw(x,y) {
        drawradio(x,y,radiolabels,...)
    }
}
...
```

```
Widget[] data; // initialized somehow
x = 0; y = 0; maxh = 0;
for(i = 0; i < data.length; i++) {
    if(x+data[i].width > linewidth) {
        x = 0;
        y += maxh;
        maxh = 0
    }
    data[i].draw(x,y)
    x += data[i].width;
    maxh =
        max(maxh,data[i].height)
}
```

Scheme

- A widget is stored as a list, with type, width, height, ... in fixed positions
- A screen is a list of widgets
- Main algorithm is pretty similar to above, except recursion (or mapcar) used to iterate over list
- Code seems somewhat opaque since widget fields often accessed as “(caddr widgetlist)”, e.g.

ML

- A lot like Scheme, but use of ML data struct makes field access more transparent
- Iteration via “foldl”, using another data struct to “accumulate” info about current x, y to decide whether next widget fits on a line

Critique

Pro/Con of OO design (here)

- *Algorithm* recognizably the same in all four languages, despite, e.g., loops vs recursion vs fold.
- + OO Localizes/groups/encapsulates info
 - + Main does layout alg, largely widget-independent
 - + Widget holds generic widget-essence
 - + Subclasses hold widget-specific stuff
- + OO probably better for code reuse/extension
- OO somewhat verbose
- Re Scheme: typlessness is a 2-edged sword, & lack of named data struct fields probably hurts (the “caddr” problem)

Change Orders

- Format control
 - Fonts, sizes, colors, ...
- Layout control
 - Justification, recursive subregions, tables...
- New widgets
 - Sliders, dials, pull-downs, .png, .jpg, ...
- Windows/Mac/Linux ports...