A Small GUI Language

Administrivia

- PA5:
  - due Thu Dec 9
  - if you've ran out of late days, you can still submit late, with a penalty of 10%/day
  - submit not later than Sunday Dec 12
- PA6:
  - due Monday, Dec 13
  - your test cases may be selected as benchmarks to declare the winner
  - winner to be declared at the final exam

Lecture Outline

- Follow-up to Dave's lecture
  - a do-it-yourself language for GUI programming
  - design and implementation
  - from Fall 2003 final exam (design was given)
  - see the exam for more details on this language
- HKN Course Survey
  - with a few curious questions from me

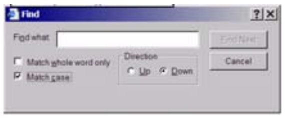
The problem

- Problem:
  - we have a GUI library
  - happy with its functionality
  - but client programs are too tedious to write
  - clients contain repetitive code → opportunity!
- Solution:
  - design a small language
  - a declarative language (state what, not how)
  - a simple, convenient layer over a complicated library
  - client programs will be concise, easy to develop

What is GUI programming?

1. creating windows, menus
2. linking them to actions in client code

- our example language will take care of only the first



Find what

☐ Match whole word only

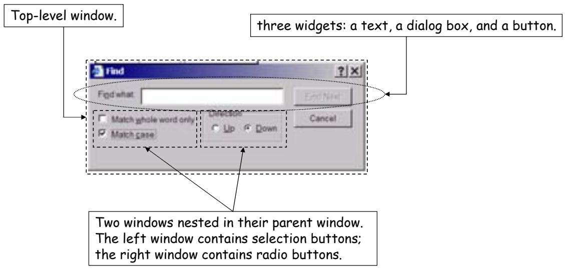
☑ Match case

Direction ○ Up ● Down

A hypothetical GUI library

- Key elements:
  - widgets
    - label (text)
    - dialog box (for entering strings)
    - button (such as Cancel)
    - selection button (select zero or more options)
    - radio buttons (select exactly one of multiple options)
  - windows
    - contain widgets and (nested) windows
    - content organized in rows
    - an optional window title

An example window



Top-level window.

three widgets: a text, a dialog box, and a button.

Two windows nested in their parent window.
The left window contains selection buttons;
the right window contains radio buttons.

Client code for the example window

```
// the constructor's argument is always the parent window;
Window top = new Window(null);    // top-level window is parentless
top.setTitle(``Find");

// The first row of the top-level window

Text t = new Text(top);
t.setPosition(0,0);
// sets position within the parent window, given as x,y coord.
// position is relative to top left corner of parent window
// values are in percent of the parent size
t.setLabel("Find what:");

Dialog d = new Dialog(top);
d.setPosition(20,0);
d.setWidth(18*someConstant);  // there are 18 dashes in <--...-->

Button f = new Button(top);
f.setType(REGULAR_BUTTON);
f.setPosition(80,0);
f.setLabel("Find Next");

// Second row of the top level window
// Left nested window
Window w1 = new Window(top);
w1.setPosition(0,50);

Selection s1 = new Selection(w1);
s1.setPosition(0,0);
s1.setLabel("Match whole word only");

Selection s2 = new Selection(w1);
s2.setPosition(0,50);
s2.setLabel("Match case");
s2.setSelected(true);  // this selection is checked

// Right nested window
Window w2 = new Window(top);
w2.setPosition(45,50);
w2.setTitle("Direction");
w2.setFramed(true);

Button r1 = new Button(w2);  r1.setType(RADIO);
r1.setPosition(0,0);         r1.setLabel("Up");

Button r2 = new Button(w2);  r2.setType(RADIO);
r2.setPosition(50,0);        r2.setLabel("Down");
r2.setSelected(true);  // this button is checked

// The very last element
Button c = new Button(top);
c.setType(REGULAR_BUTTON);
c.setPosition(80,50);
c.setLabel("Cancel");

// Finally, draw the entire window (it draws its subwindows,
// too, of course)
top.draw();
```

The client code in detail (1)

- **Create top-level window**
  - **the constructor's argument is always the parent window**
  - **top-level window is parentless (null argument)**

```
Window top = new Window(null);
```

  - **set title: null argument means window has no title**

```
top.setTitle(“Find”);
```

The client code in detail (2)

- **Now create the first widget (the text label)**

```
Text t = new Text(top);
t.setLabel("Find what:");
```

  - **set position within the parent window, given as x,y coord (in percent of the parent size)**
  - **position is relative to top left corner of parent window**

```
t.setPosition(0,0);
```

- **The second widet (the dialog box))**

```
Dialog d = new Dialog(top);
d.setPosition(20,0);
```

  - **set dialog box width (in percent of the parent width)**

```
d.setWidth(50);
```

The client code in detail (3)

- **Similarly, create the third widget (the Find button)**

```
Button f = new Button(top);

f.setLabel("Find Next");

f.setType(REGULAR_BUTTON);

f.setPosition(80,0);
```

The client code in detail (4)

- **Create the left nested window**

```
Window w1 = new Window(top);
w1.setPosition(0,50);
```

- **Create the selection buttons within the nested window**

```
Selection s1 = new Selection(w1);
s1.setPosition(0,0);
s1.setLabel("Match whole word only");

Selection s2 = new Selection(w1);
s2.setPosition(0,50);
s2.setLabel("Match case");
```

  - **this selection button is initially checked**

```
s2.setSelected(true);
```

The client code in detail (5)

- **Create the right nested window**

```
Window w2 = new Window(top);
w2.setPosition(45,50);
w2.setTitle("Direction");
w2.setFramed(true);
```

- **Create the selection buttons within the nested window**

```
Button r1 = new Button(w2);  r1.setType(RADIO);
r1.setPosition(0,0);  r1.setLabel("Up");

Button r2 = new Button(w2);  r2.setType(RADIO);
r2.setPosition(50,0);  r2.setLabel("Down");
```

  - **this radio button is initially checked**

```
r2.setSelected(true);
```

The client code in detail (6)

- **The last widget**

```
Button c = new Button(top);
c.setType(REGULAR_BUTTON);
c.setPosition(80,50);
c.setLabel("Cancel");
```

- **Finally, draw the top-level window (will draw its sub-windows, too)**

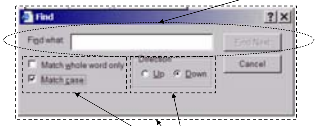
```
top.draw();
```

Designing a higher-level language (1)

- What level of abstraction do we want?
- One painfully low-level detail:
  - the GUI code had to specify coordinates
  - can we avoid specifying these coordinates? idea:
    - organize widgets into rows, specified by the programmer
    - have the compiler for our small language compute coordinates for us
    - less flexibility (cannot fine tune positions) but faster programming
- Another low-level detail we'd alike to avoid
  - specifying parents of windows
  - windows (nearly) always nested, so let's use nested scoping to convey parenthood

Rows: a concept in our new language

first row of the top-level window; contains three elements: a text, a dialog box, and a button.



Two windows nested in their parent window. Each nested window contains two elements; the left window has two rows, the right window has one.

Designing a higher-level language (2)

- Why is client code so verbose?
  - a separate method call to set each attribute
    - good software engineering, but painfully slow coding
  - idea: use compact mnemonic encoding

["Find"]	button with label "Find"
o "Up"	radio button with label "Up"
O "Up"	radio button with label "Up", initially selected
<--->	dialog box with length 3 "units" (there are 3 dashes)

Same window in our small language

```
window "Find" {
    "Find what:"  <------------------> ["Find next"],
    window "" {
          x "Match whole word only",
          X "Match case"
    }
    window framed "Direction" {
          o "Up"
          O "Down"
    }
    ["Cancel"]
}
```

comma starts a new row

Implementation

- We’re done with the language design
  - not really, we only conveyed key idea, with an example
  - in practice, must define language fully (unambiguously document semantics of each language feature)
  - focus of an entire course on programming languages

- Still, let’s proceed to implementation
  - the focus of the final exam’s question

Implementation exam questions (1)

- lexical specification of the small language:
  - identify lexical elements, and their attributes (if any)
- syntactic analysis:
  - write a context-free grammar for the language
- AST
  - what AST nodes do you need? what attributes do they have?
  - draw an AST for the example program
  - syntax directed translation for creating the AST

Implementation exam questions (2)

- Implement an interpreter
  - assume a visitor for your AST
  - can do it in multiple passes
    - compute coordinates
    - invoke the library methods
- Implement a compiler
  - rather trivial once you have an interpreter
  - recall PA2 (interpreter vs. compiler)
    - one created the NFA
    - the other emitted the code that creates the NFA
  - compiler created by emitting parts of interpreter code