UI Programming

(part of this content is based on previous classes from Anastasia, S. Huot, M. Beaudouin-Lafon, N. Roussel, O. Chapuis)

Assignment 1 is out!

Design and implement an interactive tool for creating the layout of comic strips


Graphical interfaces

GUIs: input is specified w.r.t. output

Input peripherals specify commands at specific locations on the screen (pointing), where specific objects are drawn by the system. Familiar behavior from physical world

WIMP interfaces

WIMP: Window, Icons, Menus and Pointing

Presentation
  - Windows, icons and other graphical objects

Interaction
  - Menus, dialog boxes, text input fields, etc

Input
  - pointing, selection, ink/path

Perception-action loop
  - feedback
Software layers

Application
Interface Tools & Toolkits
Graphics Library
Windowing System
Input/Output
Operating System

Applications/Communication (MacApp)
Builders, Java Swing, JavaFX, Qt (C++), GTK+, MFC, Cocoa
GDH+, Quartz, GTK+/Xlib, OpenGL
X Windows (+KDE or GNU)

Windows, Mac OS, Unix, Linux, Android, iOS, WindowsCE

Input/output peripherals

Input: where we give commands
Output: where the system shows information & reveals its state

Interactivity vs. computing

Closed systems (computation):
- read input, compute, produce result
- final state (end of computation)

Open systems (interaction):
- events/changes caused by environment
- infinite loop, non-deterministic
Problem

We learn to program algorithms (computational)

Most languages (C/C++, Java, Lisp, Scheme, Pascal, Fortran, ...) designed for algorithmic computations, not interactive systems

Problem

Treating input/output during computation (interrupting computation) ...

- write instructions (print, put, send, ...) to send data to output peripherals
- read instructions (read, get, receive, ...) to read the state or state changes of input peripherals

Problem

To program IS in algorithmic/computational form

two buttons B1 and B2
finish <- false
while not finish do
  button <- waitClick () // interruption, blocked comp.
  if button
    B1 : print « Hello World »
    B2 : finish <- true
  end
end

Managing input

Querying
Query & wait
1 device at a time

Polling
Active wait
Polling in sequence
CPU cost

Events
Wait queue
### Event based (driven) programming

**Source:** Mouse Click

**event (waiting) queue**

```
while active
  if queue is not empty
    event <- queue.dequeue()
    source <- findSource(event)
    source.processEvent(event)
  end if
end while
```

```
queue.enqueue(event)
```

**Example: Swing (and AWT)**

3 threads
- Initial thread: main()
- EDT manages the events queue: sends events to listeners (functions dealing with events) and calls paint methods (drawing functions)
- Worker (or background) threads, where time-consuming tasks are executed

**Software layers**

- Application
- Interface Tools & Toolkits
  - Graphics Library
- Windowing System
- Input/Output
- Operating System
**Interface builders**

Examples: MS Visual Studio (C++, C#, etc.), NetBeans (Java), Interface Builder (ObjectiveC), Android Layout Editor

- Can be used to:
  - create prototypes (but attention it looks real)
  - get the « look » right
  - be part of final product
  - design is fast
  - modest technical training needed
  - can write user manuals from it

But: still need to program (and clean code ...)

---

**Interface toolkits**

- Libraries of interactive objects (« widgets », e.g., buttons) that we use to construct interfaces
- Functions to help programming of GUIs
- ...usually also handle input events (later)

<table>
<thead>
<tr>
<th>Toolkit</th>
<th>Platform</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qt</td>
<td>multiplatform</td>
<td>C++</td>
</tr>
<tr>
<td>GTK+</td>
<td>multiplatform</td>
<td>C</td>
</tr>
<tr>
<td>MFC later WTL</td>
<td>Windows</td>
<td>C++</td>
</tr>
<tr>
<td>WPF (subset of WTL)</td>
<td>Windows</td>
<td>(any .Net language)</td>
</tr>
<tr>
<td>FLTK</td>
<td>multiplatform</td>
<td>C++</td>
</tr>
<tr>
<td>AWT / Swing</td>
<td>multiplatform</td>
<td>Java</td>
</tr>
<tr>
<td>Cocoa</td>
<td>MacOs</td>
<td>Objective C</td>
</tr>
<tr>
<td>Gnustep</td>
<td>Linux, Windows</td>
<td>Objective C</td>
</tr>
<tr>
<td>Motif</td>
<td>Linux</td>
<td>C</td>
</tr>
<tr>
<td>JQuery UI</td>
<td>Web</td>
<td>javascript</td>
</tr>
</tbody>
</table>

Problem with toolkits? ....
Why Java Swing?

Based on Java (any platform, plenty of libraries)
A lot of online resources and examples

Other alternatives for Java?

→ JavaFX: soon becomes the new standard for Java UI programming, supporting a variety of different devices

« widgets » (window gadgets)

Swing widgets
**Swing widgets**

- Buttons, scroll bars, labels, ...

**Widget complexity**

- Simple widgets
  - buttons, scroll bars, labels, ...
- Composite/complex widgets
  - contain other widgets (simple or complex)
  - dialog boxes, menus, color pickers, ...

**Widget tree**

Hierarchical representation of the widget structure

- a widget can belong to only one « container »

**Swing widget classes**

A GUI application has a top-level (container) widget that includes all others

In Swing there are 3 types: JFrame, JDialog and JApplet

They all contain other widgets (simple or complex), that are declared in the field **content pane**
Swing widget classes

Base class for all Swing components (except for top-level containers)

Partial object hierarchy of Swing widgets

http://docs.oracle.com/javase/tutorial/ui/features/components.html

Swing JFrame

A window with a basic bar

```java
public static void main(String[] args) {
    JFrame jf = new JFrame("Ta ta!");
    jf.setVisible(true);
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    System.out.println("finished ? ! ?");
    System.out.println("no, still running …");
}
```

Useful functions

- public JFrame();
- public JFrame(String name);
- public Container getContentPane();
- public void setJMenuBar(JMenuBar menu);
- public void setTitle(String title);
- public void setIconImage(Image image);

This program does not terminate after "no, still running …"

Swing JDialog

A message window (dialog) can be “modal” (blocks interaction)
usually attached to another window (when that closes, so does the dialog)

```java
public static void main(String[] args) {
    JFrame jf = new JFrame("Ta ta!");
    jf.setVisible(true);
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JDialog jd = new JDialog(jf, "A dialog", true);
    jd.setVisible(true);
}
```

Modal attached to

Widget placement

UI toolkits control widget placement:

- should be independent of widget size (menu at least as big as its largest item, change of scrollbar size with document size, adjusting text flow)
- done in layout managers that can be added to container widgets

This program does not terminate after "no, still running …"
import javax.swing.*;
import java.awt.*;

public class SwingDemo2 extends JFrame {
    public void init() {
        this.setTitle("example 2");
        getContentPane().add(new JLabel("Swing Demo 2"));
        Container contentPane = this.getContentPane();
        contentPane.setLayout(new FlowLayout());
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        contentPane.add(new JButton("clique ici"));
        contentPane.add(new JButton("clique là"));
    }
    public static void main(String[] args) {
        SwingDemo2 frame = new SwingDemo2();
        frame.init();
        frame.setSize(200, 200);
        frame.setVisible(true);
    }
}

Bruce Eckel, Thinking in Java, 2nd edition

---

**Widget placement**

**General guides**
- embed geometry of a «child» widget to its parent
- parent controls the placement of its children

**Layout algorithm**
- natural size for each child (to fit content)
- size and position imposed by parent
- constraints: grid, form, etc.

---

**Layout managers (in Swing)**

- BorderLayout
- FlowLayout
- BoxLayout
- GridLayout
- GroupLayout

---

http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html
Layout managers (in Swing)

GridLayout: grid

GridLayout gridLayout = new GridLayout(0, 2);
JPanel gridPanel = new JPanel();
gridPanel.setLayout(gridLayout);
gridPanel.add(new JButton("Button 1"));
gridPanel.add(new JButton("Button 2"));
gridPanel.add(new JButton("Button 3"));
gridPanel.add(new JButton("Long-Named Button 4"));
gridPanel.add(new JButton("5"));

GridBagLayout: sophisticated grid

Placement guides (Mac OS X)

Center balance: visual balance of a container's content between the left and right parts
Placement guides (Mac OS X)

Alignment

Column of labels with right alignment
Column of controls with left alignment

Spacing

Same space before and after separator
Same space on every side
Same space between controls

Placement guides (Mac OS X)

Alignment and consistency

Column with labels with right alignment
Column of controls with left alignment

CRAP
contrast, repetition, alignment, proximity

Consistency between controls of the same type

Slide deck by Saul Greenberg. Permission is granted to use this for non-commercial purposes as long as general credit to Saul Greenberg is clearly maintained.
Warning: Some material in this deck is used from other sources without permission. Credit to the original source is given if it is known.
Good Design Is As Easy as 1-2-3

1. Learn the principles.
   They’re simpler than you might think.

2. Recognize when you’re not using them.
   Put it into words – name the problem.

3. Apply the principles.
   You’ll be amazed.

Good design is as easy as...

1. Learn the principles.
   They’re simpler than you might think.

2. Recognize when you’re not using them.
   Put it into words – name the problem.

3. Apply the principles.
   You’ll be amazed.

A First Lesson in Graphical Design

Contrast

Repetition

Alignment

Proximity

CRAP

Robin Williams Non-Designers' Design Book, Peachpit Press
Contrast
make different things different
brings out dominant elements
creates dynamism

Repetition
repeat design throughout the interface
consistency creates unity

Alignment
creates a visual flow
visually connects el.

Proximity
groups related
separates unrelated
Where does your eye go?

CRAP give you cues about how to read the graphic

Boxes do not create a strong structure
- CRAP fixes it

Where does your eye go?

Some contrast and weak proximity
- ambiguous structure
- interleaved items

Strong proximity (left/right split)
- unambiguous

Where does your eye go?

Where does your eye go?
Where does your eye go?

The strength of proximity

- alignment
- white (negative) space
- explicit structure a poor replacement
Proximity
Alignment
Contrast
Repetition
IBM's Aptiva Communication Center

Example of bad design

Example of bad design

Reparing the layout

Reparing the layout

(Mullet & Sano, 1995)
Facets of a widget

Presentation
appearance

Behavior
reaction to user actions

Interface with the application
notification of state changes

Example: Button
border with text inside
« pressing » or « releasing » animation when clicked
call function when the button is clicked

Variable wrappers (active variables)

two-way link between a state variable of a widget and another application variable
(in Tcl/Tk referred to as tracing)

problems
- limited to simple types
- return link can be costly if automatic
- errors when links are updated by programmers
Event dispatching

widgets act as input peripherals and send events when their state changes
a while loop reads and treats events
associate an object to a widget, and its methods to changes in the widget state

---

Callback functions

Registration at widget creation

Call at widget activation

---

Event dispatching

divide event sending and treatment
better encapsulation (inside widget class)

---

Callback functions

Problem: spaghetti of callbacks
Sharing a state between multiple callbacks by
- global variables that widgets check:
  too many in real applications
- widget trees: callback functions are called with a reference to the widget that called it (visible in the same tree)
  Fragile if we change the structure of the UI, does not deal with other data not associated to widgets (e.g. filename)
- token passing: data passed with the callback function call
Callback functions

```c
/* callback function */
void DoSave (Widget w, void* data) {
    /* retrieve file name */
    filename = (char**) data;
    /* call an application function */
    SaveTo (filename);
    /* close the dialog */
    CloseWindow (getParent(getParent(w)));
}
/* main program */
main () {
    /* variable with file name */
    char* filename = "";
    /* create a widget and associate a callback */
    ok = CreateButton (....);
    RegisterCallback (ok, DoSave, (void*) &filename);
    /* event manager loop */
    MainLoop ();
}
```

Event listeners (Java)

A variation of callbacks in Java:

Methods of type `addListener` that do not specify a callback function but an object (the `listener`)
when a widget changes state, it triggers a predefined method of the `listener` object (e.g. `actionPerformed`)

```java
public class ClickListener implements ActionListener {
    public void actionPerformed(ActionEvent e){
        JButton button = (JButton)e.getSource();
        ...
    }
}
ClickListener listener = new ClickListener();
JButton button = new JButton(''Click me'');
button.addActionListener(listener);
```

Anonymous Inner classes

```java
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e){
        ...
    }
});
panel.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e){
        ...
    }
});
```

Methods and events are predefined
Event listeners (Java)

Anonymous Inner classes

“new <class-name> () { <body> }”

this construction does 2 things:
- creates a new class without name, that is a subclass of <class-name> defined by <body>
- creates a (unique) instance of this new class and returns its value

this (inner) class has access to variables and methods of the class inside which it is defined

Events and listeners (Java)

Each has a source (e.g. JButton, JRadioButton, JCheckBox, JToggleButton, JMenu, JRadioButtonMenuItem, JTextField)

Can get it with the function getSource()

(Listeners) need to implement the interface that corresponds to event e.g. ActionEvent => ActionListener :

```java
public interface ActionListener extends EventListener {
    /** Invoked when an action occurs.*/
    public void actionPerformed(ActionEvent e);
}
```

Events (Java)

all events inherit from the class EventObject

all listeners correspond to an interface that inherits from EventListener

a class receiving notification events of some type needs to implement the corresponding interface:
- ActionEvent
- MouseEvent
- KeyEvent
- ...
Events and listeners (Java)

- Listeners need to be registered (added) to widgets.
- A listener can be added to multiple widgets:
  - E.g., one listener handles events from multiple buttons.
- A widget can have many listeners:
  - E.g., one for "click" events and for "enter" on button events.

« drag-and-drop » to think about

- What are the affected « widgets »?
- What are the events?

How to describe this interaction with a « event listener »?

Interface toolkits

Event-action model:
- Can lead to errors (e.g., forgotten events).
- Difficult to extend (e.g., add hover events).
- Complex code.

Hard to do things the toolkit was not designed for:
- E.g., multi-device input, multi-screen applications,
  advanced interaction techniques (CrossY).