Swing is COOL, but what else is there?

On other GUI toolkits
GUI Toolkits

- AWT, Swing - Java Based (Jython)
  - Java3d, JavaFx (example)
- Tk, TkInter
  - Old but very cross-platform
- GTK+/#, Qt
  - Python, C, Java, C#
- Windows Forms – C#, Visual Basic
- Other…
  - e.g., wxPython, specialized toolkits
(NRGUiS)

- (Not Really Graphic User Interfaces)
- A few examples:
  - Java3D - example
  - OpenGL
    - 3D-drawing environment
    - Used for animations, 3D-modelling
    - User interface module: glut
  - MatLab
    - Modelling with mathematical formulas
    - Data display
So, what’s the difference?

- Small differences
  - in the principles
  - larger in the details

- Big similarities
  - If you know one, you can learn the others

- But still...
  - (e.g. Terminology problems)
General Idea - Containment

- Top Level
- Frame
  - Frame
    - Text Entry
  - Frame
    - Button
  - Frame
    - Button

Containment

Event handling
Other similarities

- LayoutManagers
- Widgets
  - Standard sets
- Component Hierarchy
- Interactivity (Event handling)
- Predefined standard components
- OO (sort of)
Differences?

- Some theory to start with

- `paintComponent(Graphics g)`
  - an interesting conceptual view into GUI technique

- Illustrates an important distinction
Three perspectives

- Conceptual – What are things?
- Presentation – How are they (re)presented?
- Graphical – How are they implemented?
Conceptual

- A window is ”on the screen”
- Some graphic is
  - ”on the canvas” which is
  - ”in the window”
- A button is placed on the window
- An animation is ”moving over the window”
  - could be ”over the component” (or under)
The Conceptual perspective

- is common to most GUI Toolkits
  - (in principle)

- the number and types of components may vary

- But a window is a window is a window.
The Conceptual perspective

- Implements the containment principle
- Is the key to our “understanding” of the GUI
- Is not ”real” but an abstraction
The Presentation perspective

- Describes the components
- and their relations on the graphic interface

- A GUI window is composed of… etc.
  - If you click on point(x,y) that will affect Button B.
The Presentation perspective

- Translates the conceptual understanding (perspective) of the GUI

- into a flat representation

- A 2D abstraction of a (mental) 3D world
  - Things are "on top" of each other
The Graphic perspective

- Describes the composed picture of the GUI

- Everything is lines and dots
  - (pixels on a canvas that is painted)

- No conceptual difference between parts

- The paintComponent "view"
  - (i.e. the resulting Graphics environment)
The Graphic perspective

- Translates the presentation of the conceptual perspective
- into a flat graphic picture (a pixel map)
  - Everything is pixels and points
- No objects, just graphics
- The lowest level of representation
  - You can (have to) manage everything
Interactive graphics

- Have to handle Mouse clicks and other Actions

- `public boolean within(Point p)`

- Flexible, but tedious to manage
Conceptual Perspective

- Overall the same between toolkits
- Widgets may vary
- Standard Interactive Widget operations
- Standard look-and feel
  - (but consider e.g. the Mac menus)
(Re)presentational perspective

- The general principle is the same
  - (almost better than in conceptual perspective)

- Difference in functionality (sometimes)
- Difference in terminology (sometimes)
Graphic perspective

- Graphics tend to differ the most
  - Canvas painting is implemented similarly

- Widget/Component paint differs (often)

- Still pixel/point
Interaction

- Conceptual level
  - Swing thinking
  - Interaction with Components

- Presentation level
  - Interaction with Component representation

- Graphic level
  - Interaction with pixels
    - Is this pixel within "boundaries"?
AWT vs. Swing

- **AWT**
  - less advanced
  - closer resemblance to platform LookandFeel
  - some features not implemented
    - E.g. icons, tooltips

- **Swing**
  - advanced features
  - platform independent
    - (cross-platform LookandFeel)
  - greater control
Swing and Windows Forms

- Similar mechanisms
- Allows for rich redesign possibilities (of components)
- Overriding of painting mechanism
  - (custom controls)
Tkinter

- Older technology (historically)

- Robust and well tested system
  - Stable implementations

- More difficult to make fine tuning
GTK+

- Graphic toolkit
- Coded in C
  - links to many different programming languages
  - cross-platform implementations
- Advanced features
Qt

- Written in C++ (efficient)
- Connects with many programming languages
  - Via APIs
  - Python, Java, C, C++ etc.
- Competent toolkit

- Is not completely integrated with the programming languages
  - (compared to Swing/Java, Windows forms)
Hello World, PyQt

#!/usr/bin/python
# HelloWorld.py
import sys
from PyQt4 import QtGui, QtCore
class QuitButton(QtGui.QWidget):
    def __init__(self, parent=None):
        QtGui.QWidget.__init__(self, parent)
        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Hello, World!')
        quit = QtGui.QPushButton('Close', self)
        quit.setGeometry(10, 10, 60, 35)
        self.connect(quit, QtCore.SIGNAL('clicked()'),
                     QtGui.qApp, QtCore.SLOT('quit()'))
app = QtGui.QApplication(sys.argv)
qb = QuitButton()
qb.show()
sys.exit(app.exec_())
Event handling

- **ActionListeners (Swing)**
- **EventListeners (General)**
  - We click on something and something happens

- **Signals and Slots (Qt)**
  - Signals are sent on Events
  - Slots react to the signals
The final choice?

- There is no winner

- Important to know the principles

- Swing provides one of the richer STANDARD GUI environments

- The principles are applicable to many GUI-toolkits
Phones and Tablets?

- Different environments:
  - iPad, iPod and iPhone: Objective-C
  - Android: Java

- Requires special SDKs for the specific platforms

- Programming is similar to Swing programming
Major differences?

- Interaction means
  - No keyboard (bluetooth)
  - Gestures
  - Multitouch
  - Feedback

- Interaction media
  - Small screen space
  - Interaction without mouse and keyboard
Programming?

- Specially developed SDKs
- Provides support for most technical features
  - Through APIs
- Two different ways to work
Native development

- Software is developed specifically for the intended device (IOS, Android, Windows Phone, etc.)

- Special IDEs (Xcode, Eclipse + extensions, NetBeans Adapted)
  - Simulation environments
Native Development

- Specialised apps
- More Control
- Higher entry level
  - Need to handle low-level things properly
  - Still – API support!
General Development

- Application is developed in JavaScript + HTML5

- IDE converts to App for selected platform
  - Applies specific APIs plus SDK

- One implementation – several platforms
  - Even Standalone Web applikation
General Development

- Lower entry-level

- Less powerful applications
  - But still pretty advanced

- Difficult making special solutions
  - (for interface features, e.g.)
Seminar Tuesday 22/5

- Present your application (in its current state)

- Also present for discussion one difficulty/problem you have encountered,
  - and if possible your solution to it.

- Presentation time: 15 minutes
  - Allow five minutes for discussion