Building Enterprise Database Applications

using Rational Rose, Java and Mr Architecture

A short course by Kade Hansson

Course Contents

- **UML, object-orientation and design patterns**
- **Java language and essential APIs**
- **Java GUI components and event model**
- **Java I/O and TCP/IP sockets**
- **JDBC, Servlets and JSPs**
- **Mr Architecture**

The Applet Convention

- The applet convention allows Java to run in a web browser
  - Applets are included in HTML pages using the **`object`** or **`applet`** tag:
    - `<`**`applet`** `code="`*ClassName*`.class" width="`*widthInPixels*`" height="`*heightInPixels*`">`
      `<`**`param`** `name="`*name*`" value="`*value*`" />`
      `</`**`applet`**`>`
    - More at http://java.sun.com/docs/books/tutorial/applet/appletsonly/html.html
- An applet implements a GUI (graphical user interface)
- An applet is a subclass of the class *Applet*
- Most AWT applets will override the `Applet.paint` method
- Most applets will need to implement "listener" interfaces

Applet Life Cycle

- An applet has a well-defined lifecycle
  - States
    - **loaded**
    - **initialized**
    - **started**
    - **stopped**
    - **finalized**
    - **unloaded**
  - Transitions
    - It is initialized, when it prepares itself to be run (the `init` method)
    - It is started and stopped, possibly many times (the `start` and `stop` methods)
    - It is finalized, whereupon it performs a final tidy-up (the `destroy` method)

Applet Security

- An applet cannot execute arbitrary code via `System.exec()`
- An applet can't read or write files on the local system
- An applet cannot make arbitrary network connections
  - An applet cannot resolve a name directly
  - An applet is allowed to connect to the host it originated from
- An applet cannot print (but it's GUI can be printed)
- Applet windows (where outside a web page) are clearly identified as such to prevent "spoofing"

- *If you need to do any of this, you probably shouldn't be writing an applet, but check out http://java.sun.com/docs/books/tutorial/security1.2/index.html*

Abstract Window Toolkit

- AWT piggy-backs on platform defined widgets
  - Fast +
  - Economical +
  - Simple to use +
  - Messy results –
  - Inflexible –
  - Restricted set of widgets –

AWT Components and Containers

- *Component*
  - ***Canvas*- something to draw on**
  - ***Button*- fires *ActionEvents***
  - ***Checkbox*- registers choices**
  - ***Choice*- a button with a drop down menu**
  - ***Label*- contains some text**
  - ***List*- an (scrollable) in-line list of options**
  - ***Scrollbar***
  - ***TextComponent*- for user-editable fields**
  - *Container* is a *Component*– **Composite** pattern
    - ***Panel*- generic container**
    - ***ScrollPane*- scrollable container**
    - ***Window*- with resize, close, minimize, maximize gadgets etc.**

Painting

- A typical *Applet* will need to "paint" in its window
  (or at least get the AWT or some other toolkit to paint on its behalf)
  - **Painting is done by overriding the *update()* or *paint()* method**
  - **The AWT calls *update()* to paint the whole of a *Component*, such as an *Applet*'s window**
  - **By default, *Component.update()* clears its area and calls *paint()* (possibly many times)**
  - **Typically, to specify how to draw on an area, you merely override *paint()***
  - ***paint()* may be called multiple times during a single update, once for each visible rectangle of the component**
  - **If you change something visible in a *Component*, and want it to be rerendered, you call *Component.repaint()* (which will by default schedule a call to *update()* ASAP)**
  - **Efficiency in animation applications can be improved by maintaining an *Image* (or two[1]) for the *Component*'s area, and simply painting that at each call to *update()***

- ***You may see something called* PaintEvent *in the forthcoming slide... It's internal to AWT– it is not used with the Event Listener model and is shown only for completeness***

[1] This is called **double-buffering**

Events

- All events are subclasses of *EventObject*[2]
  - Method *getSource()* returns *Object* which caused *Event* (and with which *EventListeners* are registered)
- *ComponentEvent*
  - ***ContainerEvent*[3]*, FocusEvent (FOCUS_GAINED, FOCUS_LOST), PaintEvent*[3]*, WindowEvent (WINDOW_ACTIVATED, WINDOW_CLOSED, WINDOW_CLOSING, WINDOW_DEACTIVATED, WINDOW_DEICONIFIED, WINDOW_ICONIFIED, WINDOW_OPENED)***
  - *InputEvent*
    - ***KeyEvent (KEY_PRESSED, KEY_RELEASED, KEY_TYPED)***
    - ***MouseEvent (MOUSE_PRESSED, MOUSE_RELEASED, MOUSE_CLICKED, MOUSE_ENTERED, MOUSE_EXITED, MOUSE_DRAGGED, MOUSE_MOVED)*** [4]

---

[2] *Event* is defunct – it was used by the Java 1.0 event model
[3] Used internally by AWT – these methods do not follow the Event Listener model
[4] Both *MouseListener* and *MouseMotionListener* use this event

Java Foundation Classes – AKA Swing

- Swing renders its own platform-independent set of widgets using double-buffering
  - Clean look across platforms +
  - Good use of **design patterns** +
  - Rich set of widgets +
  - Accessibility +
  - Slower that AWT –
  - More memory hungry –

Swing Components and Containers

- *JComponent*
  - *JFrame* is a window with a border- it contains a *JPanel* which holds the window contents
  - *JApplet* is similar to *JFrame*, but is used to make Swing applets– it may be embedded in a web browser
  - *JPanel* is the Swing equivalent to *Container*
    - *JPanel* is a *Container* which is used as a "frame" to hold other *JComponent*s

Swing Widgets

- ***JLabel*s are used for static text and graphics**
- ***JButton*s are interactive components which the user may click on– an *ActionEvent* is delivered to all the button's *ActionListener*s when this happens**
- ***JToggleButton*s are similar to *JButton*s, except you usually interrogate their state instead of responding to events**
  - ***JCheckBox*es, which are like boxes you tick on printed forms**
  - ***JRadioIcon*s, which are like boxes in multiple choice quizzes**
- ***JScrollPane*s, which is used to put scroll bars on a *JViewport***
- ***JViewport*s are components which are used to impose a cropped view on a larger group of components.**
- ***JTextComponent*s, which are used to contain editable text. Examples are:**
  - ***JTextField*s, which are for single lines of text (*JPasswordField*s, which obscure their input)**
  - ***JTextArea*s, which allow multiple lines of text**
  - ***JTextPane*s, which provide fully-featured editor windows**
  - ***JEditorPane*s, which provide for marked-up text like HTML**

Still More Swing Widgets

- ***JScrollBar*s, for implementing scroll bars**
- ***JSlider* controls**
- ***JProgressBar* indicators**
- ***JComboBox*es, allowing data entry via a pull-down list**
- ***JBorder*s, for putting pretty borders around other components. Examples are:**
  - ***AbstractBorder*, an abstract class that implements the *Border* interface, but does nothing**
  - ***BevelBorder*, a 3D border that may be raised or lowered**
  - ***CompoundBorder*, a border that can nest multiple borders**
  - ***EmptyBorder*, a border where you specify the reserved space for an undrawn border**
  - ***EtchedBorder*, a border that appears as a groove, instead of raised or lowered**
  - ***LineBorder*, a border for single color borders, with arbitrary thickness**
  - ***MatteBorder*, a border that permits tiling of an icon or color**
  - ***SoftBevelBorder*, a 3D border with softened corners**
  - ***TitledBorder*, a border that permits title strings in arbitrary locations**

The Last of the Swing Widgets

- ***JMenuBar*s, for providing pull down menus– these are constructed from:**
  - ***JMenu*s, which are lists of *JMenuItem*s, *JSeparator*s and possibly further *JMenu*s**
  - ***JMenuItem*s, which represent a single leaf menu option. Special examples are:**
    - ***JCheckBoxMenuItem*s, which are check boxes embedded in menus**
    - ***JRadioIconMenuItem*s, which are radio icons embedded in menus**
  - ***JSeparator*s, which are used to break-up menu options**
- ***JToolBar*s, for providing sets of tool buttons**
- ***JTabbedPane*s, where multiple panes can be called up by clicking on exposed tabs**
- ***JSplitPane*s, which allows you to resize adjoining components together**
- ***JList*s, where the user chooses items from a list**
- ***JTable*s and *JTree*s, for displaying tabular and hierarchial data**
- ***JPopupMenu*s, which are special *JMenu*s (see above) which may be assigned to pop-up over certain *JComponent*s**

But Don’t Forget...

- *Icon* allows graphics to be included as part of a component
  - The most common form of *Icon* is given by the subclass *ImageIcon*
    - Constructor takes the name of an image resource
      e.g. `Icon tinyPicture = new ImageIcon("TinyPicture.gif");`