Graphic Interface Programming II

Using Swing and OOP – more advanced issues
Good interface programming

- Swing is an EXTENDIBLE toolkit

- Build your OWN components

- Many times you need the same (similar) solution several times

- Make a general (reusable) solution
Example: ImagePanel

- Working with images
- Want to use a JPanel with an image drawn on it
- Extend the JPanel!
public class ImagePanel extends JPanel {

    private Image img;

    public ImagePanel(String img) {
        this(new ImageIcon(img).getImage());
    }

    public ImagePanel(Image img) {
        this.img = img;
        Dimension size = new Dimension(img.getWidth(null), img.getHeight(null));
        setPreferredSize(size);
        setMinimumSize(size);
        setMaximumSize(size);
        setSize(size);
        setLayout(null);
    }

    public void paintComponent(Graphics g) {
        g.drawImage(img, 0, 0, null);
    }
}
public class ImageLabel extends JLabel {

    public ImageLabel(String img) {
        this(new ImageIcon(img));
    }

    public ImageLabel(ImageIcon icon) {
        setIcon(icon);
        setIconTextGap(0);
        setBorder(null);
        setText(null);
        setSize(icon.getImage().getWidth(null),
                icon.getImage().getHeight(null));
    }
}
import java.awt.*;
import javax.swing.*;

public class ImageButton extends JButton {

    public ImageButton(String img) {
        this(new ImageIcon(img));
    }

    public ImageButton(ImageIcon icon) {
        setIcon(icon);
        setMargin(new Insets(0,0,0,0));
        setIconTextGap(0);
        setBorderPainted(false);
        setBorder(null);
        setText(null);
        setSize(icon.getImage().getWidth(null),
                icon.getImage().getHeight(null));
    }
}
Reusable!

- All these Components can be extended in turn

- They are (extended) JPanels, JLabels and JButtons

- … but with a twist – Your twist
ReUsable Methods?

- Why not just make a method library?

```java
//Create and set up a colored label.
private JLabel createColoredLabel(String text, Color color, Point origin) {
    JLabel label = new JLabel(text);
    label.setVerticalAlignment(JLabel.TOP);
    label.setHorizontalAlignment(JLabel.CENTER);
    label.setOpaque(true);
    label.setBackground(color);
    label.setForeground(Color.black);
    label.setBorder(BorderFactory.createLineBorder(Color.black));
    label.setBounds(origin.x, origin.y, 140, 140); return label;
}
```
ReUsable Methods?

- Flexibility
  - Multiple Constructors
- No Copying between classes – less errors

```java
//A Colored Label Class with Black Border
private class ColoredLabel extends JLabel {
    ColoredLabel(String text, Color color, Point origin) {
        super(text); // The JLabel Constructor.
        this.setVerticalAlignment(JLabel.TOP);
        this.setHorizontalAlignment(JLabel.CENTER);
        this.setOpaque(true);
        this.setBackground(color);
        this.setForeground(Color.black);
        this.setBorder(BorderFactory.createLineBorder(Color.black));
        this.setBounds(origin.x, origin.y, 140, 140);
    }
}
```
RichLabel

- A Label class that can be configured to display a message in different styles

- Provides consistency

- Simple to adapt

- Reusable (if made right – this one isn’t)
// drop shadow w/ highlight
    label.setLeftShadow(1,1,Color.white);
    label.setRightShadow(2,3,Color.black);
    label.setForeground(Color.gray);
    label.setFont(label.getFont().deriveFont(140f));

// 3d letters
    label.setLeftShadow(5,5,Color.white);
    label.setRightShadow(-3,-3, new Color(0xccccff));
    label.setForeground(new Color(0x8888ff));
    label.setFont(label.getFont().deriveFont(140f));
Custom Borders
Custom Borders

Hack #59: Image Border

Image Border Test
Custom Borders

Provide the puzzle pieces

Let the class do the puzzling
Useful Principle

- Border images are simple to do in Photoshop

- Cut in ”slices”

- Use standardized class to draw the borders
  - Do the programming once, change images
Useful Principle

- Border images are simple to do in Photoshop
- Cut in "slices"
- Use standardized class to draw the borders
  - Do the programming once, change images

This one won’t work with this method
Not only for Java

- This method is also used for theme building in Web sites
  - Content management systems

- Often driven by PHP, but same idea

- Automated graphics generation
Always think in terms of reuse!

- What might be useful?
  - TextLabels that have a different Font?
  - TextLabels that change the textSize when scaled?
  - A Shaped Window?

- Your issue is to create it once, so you can reuse it many times
paintComponent(Graphics g)

- A key to advanced interface design in Java
- Necessary to understand how paintComponent does the painting
- We can use it to do interesting effects
Standard case

- In extended classes:
  - Call super.paintComponent(g)
  - Causes the basic drawing of the component to be performed
    - E.g. In a button, draws the button outline, the text, etc.
  - Then describe what you want to draw on top of that

- Good for most standard purposes
Painting

- **paintComponent** — The main method for painting.
  - By default, it first paints the background if the component is opaque. Then it performs any custom painting.

- **paintBorder** — Tells the component's border (if any) to paint.
  - *Do not invoke or override this method.*

- **paintChildren** — Tells any components contained by this component to paint themselves.
  - *Do not invoke or override this method.*
Stages

This is why:
• a Swing button will be drawn over any painted graphic
• you may want to override the border or children methods
"Do not invoke or override…"

- Yes, you should!
- But you have to know **how** to do this…
- Always call "super…" to be on the safe side
Overriding paintChildren

```java
public void paintChildren (Graphics g) {
    super.paintChildren(g);
    … // Initialization of image and other stuff
    g.drawImage( … some image …);
}
```

- The image will be drawn over the swing components
WaterMarking

- E.g., adding a fixed Corporate Logotype "under" some text

- Can be implemented through clever use of paintComponent()

- Requires some understanding of the painting process
ViewPort

- For large information views

- A sized "hole" through which a part of the info is visible

- ScrollBars are used to move the ViewPort over the content
Example
JScrollPane

- An implementation of the ViewPort paradigm

- Takes a JPanel, JTextArea etc. as client and adds a view port to it

- We can modify it!

(http://java.sun.com/docs/books/tutorial/uiswing/components/scrollpane.html)
Layer order

- JScrollPane contains...

- JPanel, which contains... // or JTextArea

- all components added to it...
  - Buttons, areas etc.

```java
TextArea = new JTextArea(5, 30);
...
JScrollPane scrollPane = new JScrollPane(textArea);
...```
public void setBackgroundTexture(URL url) throws IOException {
    bgimage = ImageIO.read(url);
    Rectangle rect = new Rectangle(0, 0,
        bgimage.getWidth(null), bgimage.getHeight(null));
    texture = new TexturePaint(bgimage, rect);
}

Create the Background texture from an image (URL)

public void setView(JComponent view) {
    view.setOpaque(false);
    super.setView(view);
}

Set Opacity BEFORE calling the super method
public void paintComponent(Graphics g) {
    // do the superclass behavior first
    super.paintComponent(g);

    // paint the texture
    if(texture != null) {
        Graphics2D g2 = (Graphics2D)g;
        g2.setPaint(texture);
        g.fillRect(0,0,getWidth(),getHeight());
    }
}

Call superclass behaviour and then paint the Picture with the paint.
Watermarking

- This makes the watermark stay put and the text moves over it.

- If the watermark is added to the Textarea then it will move with the scroll.
Knowledge about painting

- Essential for creating graphics
  - Outside of the "box"

- Requires practice

- Create a testing class:
  - A simple frame including the selected paintComponent definition.
  - Just clone the class för a new version
JComponent Layers

- JFrame
- ContentPane
- JPanel
- JButtons, JTextAreas, etc
- Other SubComponents

All this as managed by the paintComponent definitions (initiated by the JFrame or by a call to repaint())
Display Layers
On my request

- Interfaces
- Packages
- Structuring
When to use?

- Interface

- Simplest definition:
  
  • Use an interface when you need to be able to handle several different classes as one type
  
  • Interface is (sort of) a replacement for multiple inheritance
Other use

- Use an interface when you want to publish a support class and make sure that all methods needed will be defined by the programmer.
- Do not overuse interfaces!
  - You might consider implementing default methods instead (that can be overridden).
- Gives a safer implementation (if there is a default behaviour)
  - Compare AbstractXxxx classes.
Packages

- When should we split up into packages?

- When a project grows big (or we know that it will grow big).

- To structure different functionalities

- To avoid name clashes
Structuring tools

- Support for overview and clarity
- Support for programming projects
- It is just as bad to have too many as having too few
  - A package should be a conceptual unit
Structuring

- Use file and project structures (packages, interfaces, extensions) that are logical
  - Put similar things together with good names

- Try also to structure the files in the project in a logical way

- Use visual structuring tools
  - Straight lines in indentations, lines to divide in sections. Meta Comments that describe how classes connect