Graphic Interface Programming II

Applets…
and
Beans…
and animation in Java
Applets

- Small programs embedded in web pages

```html
<applet
    width=300
    height=300
    code="DrawingLines.class">
</applet>
```
Applets don’t Swing

- Some things have to be done differently
  - No Constructor – init()
  - No main method – the browser starts the applet
  - No Swing
  - paintComponent(Graphics g) – paint(Graphics g)

- Less complex programs – no Frames
  - The web page embeds the Applet

- (For using Swing Components – JApplet)
// Simple Applet class

public class DrawingLine extends Applet {

    // Keeps knowledge of the size of the Applet
    //
    int width, height;

    public void init() {
        width = getSize().width;
        height = getSize().height;
        setBackground( Color.black );
    }

    // In the paint method we just define a simple loop that paints ten
    // green lines over the black background.
    //
    public void paint( Graphics g ) {
        g.setColor( Color.green );
        for ( int i = 0; i < 10; ++i ) {
            g.drawLine( width, height, i * width / 10, 0 );
        }
    }
}
Good or Bad?

- Alternative to Flash
  - Good Java programmer might have advantage

- Not all platforms have Java installed
  - Some browser have Java deactivated by default
  - Security issues

- Need to consider bad and good aspects
Powerful Graphics Display

- Still…
  - Does not work on all platforms…
  - As you well know…
  - Even if Java is implemented
JavaBeans

- GUI Builders use a certain type of class

- A Bean is a class that follows certain rules
  - It has to have a public default constructor (no arguments)
  - It has to have specific getter and setter functions to access its properties
  - It should be serializable (possible to save in any state)

- A Bean can be used as a jsp-object
GUI Builders

- JavaBeans are important for GUI Builders
- Builders use the concept of *Introspection*
  - The Introspector class can examine a bean for design patterns

- Create your own Swing objects as Beans
  - Can be added as Graphic objects in the Builder
import java.awt.Color;
import java.beans.XMLDecoder;
import javax.swing.JLabel;
import java.io.Serializable;

public class SimpleBean extends JLabel implements Serializable {
    public SimpleBean() {
        setText( "Hello world!" );
        setOpaque( true );
        setBackground( Color.RED );
        setForeground( Color.YELLOW );
        setVerticalAlignment( CENTER );
        setHorizontalAlignment( CENTER );
    }
}

Example
SimpleBean

- Extends JLabel – a Visual Component
- Implements Serializable (or Externalizable)

- The Introspection feature allows the SimpleBean to have properties that can be changed

- Can be found in the Beans tab in NetBeans
  • (I am not sure about Eclipse).
Animation

- Good example of a "Design Pattern"

- How to ensure OOP principles?
Purpose of Animation Driver

- One or more Animations
  - Simulation
  - Realtime game (e.g. Battlebots)
  - Other dynamic realtime animation with several animated objects
- One Actor controls the Animation process
  - Singleton (suggested by one group)
  - Controls Animation Progress
Decisions?

- The amount of control in driver?
- Different animation speeds?
- Which objects to animate?
- How is the driver controlled?
OOP Principle

- The driver needs to be free of knowledge about the respective animations

- Each animated object responsible for its own animation, e.g.:
  - Driver controls general animation speed
  - Animated object controls its own relative animation speed
Animator Pattern

- Interface AnimatedObject
  - One primary method: tick()
  - Each tick is one cycle in general Animation

- Animator
  - Tracks (manages) the animated objects
  - Sends ticks to all its Animated Objects
  - Methods:
    - addObject(AnimatedObject ao)
    - removeObject(AnimatedObject ao)
    - ao.tick() // sends ticks to all animated objects
Animator Pattern

- Animated Object could be any class
  - As long as it implements the AnimatedObject interface

- Does not have to be a graphic component!
  - In fact it is easier to draw if it is not…

- A panel can host many animated graphics
  - defines only paintComponent(Graphics g)!
  - the graphic adds its own drawing to the Graphics environment!
Animated Graphics

- Uses the `paintComponent` of (e.g.) Jpanel

- Implements Drawable interface:
  - method: `draw(Graphics g)`
  - the method should specify the graphic commands for the graphic

- Does **not** specify the `paintComponent`
  - is not a Component, but a ”loose” graphic
Jpanel paintComponent

- Each Drawable object implements the draw() method.

```java
@Override
public void paintComponent(Graphics g) {
    Drawable d;
    for (Iterator<?> it = graphicObjects.iterator();
         it.hasNext(); ) {
        d = (Drawable) it.next();
        d.draw(g);
    }
}
```
Complex graphics

- Every graphic uses `draw(Graphics g)` to add itself to the graphics environment
- A single call to `paintComponent()`
Animated Complex Graphics

- If we want to implement graphics that move, the class has to implement both interfaces.

```java
public class MovingStar implements Drawable, AnimatedObject {
    // here the drawing is made in
    // draw(Graphics g)
}
```
Animation Process

- Create graphic Container (JPanel?)
- Collect all graphic objects (not Swing Components)
- For every tick:
  - update the graphic objects (position, size, colour, etc.)
  - In `paintComponent(Graphics g)` draw all graphic objects on the Graphics environment `g`
Trick for animation (snake)

- Use a suitable data structure that can be used as a QUEUE
  - First in – first out actions

- Push new graphic objects onto the QUEUE
- Pop old ones from the QUEUE

- If time is used for popping – varying length of ”tail”