Applets

CS 99 – Summer 2000
Michael Clarkson
Lecture 10

Administration

- Department web server down since Thursday
- Lab 9 due tomorrow
- Lab 10 posted today
- Final project posted tomorrow
- Projected grades posted by tomorrow

Agenda

- What are applets?
- Running applets
- Writing applets
- Using graphics
- Using events and listeners
- Using components

Applets

- Applications
  - What we've been writing
  - Only one type of Java program
- Applets
  - Small applications
  - Accessed via the web
  - Code structured differently than an application

Running Applets

- Applets are embedded in web pages
  - HTML documents
- Running an applet:
  - User accesses web page that includes applet
  - Applet is downloaded automatically
  - Web browser uses JVM to run the applet

Running Applets [2]

- Java is an ideal language for programs distributed over the web
  - Can't redistribute .exe files
    - Macs, UNIX won't run them
  - .class files can be run by any machine
- Java programs become just another media that can be exchanged over the web
  - Text, graphics, and sound

A Simple Applet

```
import java.applet.*;
import java.awt.*;

public class SimpleApplet extends Applet {
  public void paint(Graphics g) {
      g.drawString("Hello, world!", 20, 20);
  }
}
```


Simple Applet [1]

- Note that it doesn't have a `main` method
  - Applet execution does *not* begin with `main`
  - Usually, applets don't have a `main` method
  - Applets begin execution when the name of the class is passed to a browser
- Also doesn't have `System.out` or `Console` method calls
  - Applets don't use console input/output

Simple Applet [2]

- `import java.applet.*;`
  - Imports the `applet` package, which contains the `Applet` class.
- `import java.awt.*;`
  - Imports the Abstract Window Toolkit (AWT).
  - Applets interact with the user using the AWT
  - Large, sophisticated package with support for a windows-based, graphical interface

Simple Applet [3]

- `public class SimpleApplet extends Applet`
  - All applets must extend `Applet`
  - Applet classes must be `public` because they must be visible to another program (the web browser)

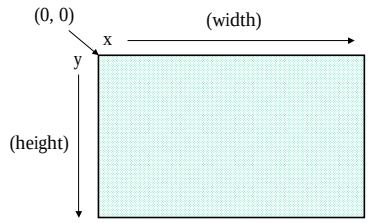
Simple Applet [4]

- `public void paint(Graphics g)`
  - Method defined by AWT
  - Usually overridden by applets
  - Called whenever applet must paint itself
  - `Graphics` parameter represents graphical context in which applet is running
    - Class defined by AWT

Simple Applet [5]

- `g.drawString("Hello, world!", 20, 20);`
  - `drawString` is a member of the `Graphics` class
  - Outputs a string at the specified (x, y) coordinates on the window
    - Upper-left corner is (0, 0)

Applet Coordinates



Compiling/Executing

- Compiling an applet
  - Same as compiling an application
  - Produces .class file
- Executing applet
  - Requires Java-enabled browser
    - Netscape
    - Internet Explorer
  - Or use JDK tool: `appletviewer`

HTML File

- Need an HTML file that contains a tag for loading the applet:

```
<!-- probably in SimpleApplet.html-->
<applet
  code="SimpleApplet"
  width="320"
  height="200"
></applet>
```


Executing Applet

- Either:
  - Open SimpleApplet.html file with browser
  - Run the command (Metrowerks):
    `appletviewer SimpleApplet.html`

Executing Applets [2]

Applet Architecture

- Event driven
  - Applet waits for events to occur
    - Events: mouse movements, key presses, etc.
  - AWT notifies applet of event by calling *event handler*
  - Applet takes appropriate action and *quickly* returns control to AWT
- User initiates interaction with applet
  - In a non-windowed application, the program initiates interaction (e.g., `Console.readString()`)
  - In applet, user interacts as s/he wants, when s/he wants

Initialization/Termination

- When an applet begins running, the AWT calls these methods, in order:
  - `init()`
  - `start()`
  - `paint()`
- When an applet is terminated, AWT calls:
  - `stop()`
  - `destroy()`

Initialization

- `init()`
  - First method in your code to be called
  - Initialize variables here (instead of constructor)
  - Only called once during applet lifetime
- `start()`
  - Restarts an applet after it stops
  - Including first time started
  - Called every time web page is redisplayed by browser (uncovered, reloaded, etc.)

Painting

- `paint()`
  - Called by AWT whenever applet must paint itself
    - Window covered by another window, then uncovered
    - Minimized, then restored
    - Applet first started
- `repaint()`
  - Method that *you* can call to tell applet to repaint itself

Termination

- `stop()`
  - Called whenever web page is left
- `destroy()`
  - Called whenever applet needs to be removed from memory (e.g., web browser closed)

Graphics

- `Graphics` class contains many methods for producing graphical output:
  - `drawString`
  - `drawLine`
  - `drawRect, fillRect`
  - `drawRoundRect, fillRoundRect`
  - `drawOval, fillOval`
  - `drawArc, fillArc`
  - `drawPolygon, fillPolygon`
  - `drawPolyline`
- See LL 2.10, 6.6

Colors

- `Color` class is used to manage colors
- Every graphics context has a foreground and background color
  - `setColor(Color c) // foreground`
  - `setBackground(Color c)`
- `Color` defines several constants for making it easy to use colors
  - `Color.red`, `Color.blue`, etc.
  - See LL 2.10

Fonts

- Represented by `Font` class (see LL Appendix M)
- To change fonts:
  - Save the old font:
    - Font oldFont = g.getFont();
  - Create a new `Font` object:
    - `Font f = new Font(name, style, size);`
    - *name* is a String, e.g., "Courier"
    - *style* is a constant: Font.PLAIN, Font.BOLD, Font.ITALIC
    - *size* is an integer point size, e.g., 12
  - Set the font:
    - `g.setFont(f);`
  - Eventually, restore the original font:
    - `g.setFont(oldFont);`

Events and Listeners

- An *event* is an object that represents some occurrence in which an applet might be interested
  - e.g., user pressing mouse button, typing a key, pushing a button, moving a scroll bar, etc.
  - Has methods to determine information about event
- A *listener* is an object that waits for events to occur and then responds to them
  - Has methods that are called when event occurs

Mouse-related Events

- Mouse events:
  - mouse pressed
  - mouse released
  - mouse clicked
  - mouse entered
  - mouse exited
- Mouse motion events:
  - mouse moved
  - mouse dragged

Listening for Mouse Events

- To listen, an object must implement the `MouseListener` interface
- The object must also register itself with the applet as a listener
- Easy way: make the applet itself the listener

Trivial Listener

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class TrivialMouseListenerApplet extends Applet
   implements MouseListener {
   public void init() {
       addMouseListener(this);
   }
   public void mousePressed(MouseEvent event) { }
   public void mouseReleased(MouseEvent event) { }
   public void mouseClicked(MouseEvent event) { }
   public void mouseEntered(MouseEvent event) { }
   public void mouseExited(MouseEvent event) { }
}
```


Implementing Listeners

- To make listener less trivial, we can react to the event.
- The `MouseEvent` object can tell us where mouse was clicked:
  - `int getX()`
  - `int getY()`
  - `Point getPoint()`
- See LL 5.5

Implementing Listeners

```
private Point clickPoint = null;
public void mouseClicked(MouseEvent event) {
  clickPoint = event.getPoint();
  repaint();
}
public void paint(Graphics g) {
  if (clickPoint != null) {
      g.drawString("Hi", clickPoint.x,
                         clickPoint.y);
  }
}
```


Components

- A *component* is a visual entity that allows the user to interact with a program or displays information
- Examples:
  - buttons
  - text fields
  - labels
  - scroll bars
  - checkboxes
  - lists

Components [2]

- An applet itself is a component
  - Also a *container* – component that contains components
- Components must be added to a container in order to be displayed
  - Added using `add` method of the container
  - Can't specify exactly where to add without advanced usage of `LayoutManager`s

Components [3]

- Components generate events
  - Buttons and text fields generate `ActionEvents`
- The component must have a listener added to it in order to process these events
- Again, the applet itself can be that listener

Component Example

Component Example [2]

```
public class GolfStrokesApplet extends Applet
                  implements ActionListener {

  private int numStrokes = 0;
  private Label strokes;
  private Button inc, dec;

  ...

}
```

Component Example [3]

```
public void init() {
    strokes = new Label("0");
    inc = new Button("+1");
    dec = new Button("-1");

    add(strokes);
    add(inc);
    add(dec);

    inc.addActionListener(this);
    dec.addActionListener(this);
}
```


Component Example [4]

```
public void actionPerformed(ActionEvent event) {
      String button = event.getActionCommand();
      Object source = event.getSource();
      if (source == inc) {
            numStrokes++;
      } else if (source == dec) {
            numStrokes--;
      }
      strokes.setText("" + numStrokes);
  }
```


For Lab Tomorrow

- Read the lab handout in advance
  - First program requires some creativity
  - You'll want to have an idea before you arrive