

7.1 Introduction

- Applets are relatively small Java programs whose execution is triggered by a browser (**client-side!**)
- The purpose of an applet is to provide processing capability and interactivity for HTML documents through widgets
- The 'standard' operations of applets are provided by the parent class, `JApplet`

```
public class class_name extends JApplet { ... }
```

- Use of applets is still widespread, and there is heavy use in intranets, where all browsers can be required to support the latest JVM
- Applets are an alternative to CGI and embedded client-side scripts

7.1 Introduction (continued)

- Comparisons (between JavaScript & applets):

- CGI is faster than applets and JavaScript, but it is run on the server
 - JavaScript is easier to learn and use than Java, but less expressive
 - Java is faster than JavaScript
 - Java graphics are powerful, but JavaScript has none
 - JavaScript does not require the additional download from the server that is required for applets
 - Java may become more of a server-side tool, in the form of servlets, than a client-side tool
- Other tools: Shockwave/Director/Lingo, Flash/ActionScript(ECMA-262), Scalable Vector Graphics (SVG), LiveMotion, etc.

7.2 Primary Applet Activities

- Browser actions:
 - a. Download and instantiate the applet class
 - b. Call the applet's `init` method
 - c. Call the applet's `start` method
 - This starts the execution of the applet
 - When the user takes a link from the document that has the applet, the browser calls the applet's `stop` method
 - When the browser is stopped by the user, the browser calls the applet's `destroy` method
- An applet's display is actually a **multi-layered frame**
 - We're only interested in one layer, the **content pane**
 - We don't write directly to the content pane

7.2 Primary Applet Activities

-Two categories of graphics operations in applets:

1. Custom drawing – use a set of primitives, using overridden versions of `paintComponent`
 - Custom drawing is done outside the applet, usually in a `Jpanel` **panel**
 - The applet instantiates the panel and adds it to the applet's content **pane**
2. Use predefined graphics objects
 - Do not use `paintComponent`
 - Put graphics objects directly into a **panel** created in the applet

7.3 The `paintComponent` Method (continued)

- Always called by the browser (not the applet itself)
- Takes one parameter, an object of class `Graphics`, which is defined in `java.awt`
 - The parameter object is created by the browser
- The protocol of `paintComponent` is:

```
public void paintComponent(  
    Graphics grafObj) { ... }
```

- The simplest use of `paintComponent` is to display text, using the `drawString` method
 - Three parameters: a `String` literal, the `x` coordinate of the left end of the string, and the `y` coordinate of the base of the string
 - Before calling `drawString` (or any other primitive), the parent class' `paintComponent` method is called to paint the background
`super.paintComponent (grafObj) ;`

→ SHOW [Wel.java](#) [wel.html](#)

7.3 The `paintComponent` Method (continued)

- Font Control

- The `Wel` applet draws strings using default values for the font, the font size, and the font style
- The `Font` class, defined in `java.awt.Font`, has three variables that specify the font name, style, and size of the font used by `drawString`

The `size` parameter is in points

- The styles are `PLAIN`, `BOLD`, and `ITALIC`

- To change the font, create a `Font` object with the desired parameters and set it with the `setFont` method of `Graphics`, which takes a `Font` parameter

→ SHOW [Wel2.java](#) [Wel2.html](#)

7.4 The <object> Tag


- Used to specify an applet in an HTML document
- Creates a space in the document display for applet output (like does)

```
<object codetype = "application/java"  
      classid = "java:applet_class_file"  
      width = "applet display width"  
      height = "applet display height">  
</object>
```

- The display width and height are in pixels
- The applet_class_file is the compiled version
- To test the Wel2 applet, we could use

```
<object codetype = "application/java"  
      classid = "java:Wel2.class"  
      width = "500"  
      height = "100">  
</object>
```

7.4 The <object> Tag (continued)



- Portability problem with the <object> tag:
 - <object> is part of the HTML 4.0 standard, but
 - IE6 recognizes <object>, but not the classid attribute (it likes the code attribute, instead)
 - If code is used, the java: part must be omitted
 - Likewise for appletviewer
 - NS6 does not recognize the code attribute
- Also, <applet> is in HTML4.01 but deprecated

7.5 Applet Parameters

- Applets can be sent parameters through HTML, using the `<param>` tag and its two attributes, `name` and `value`

- Parameter values are strings

- e.g., `<param name = "fruit" value = "apple">`

- The applet gets the parameter values with `getParameter`, which takes a string parameter, which is the name of the parameter

```
String myFruit = getParameter("fruit");
```

- If no parameter with the given name has been specified in the HTML document, `getParameter` returns `null`

- By checking the return value against `null`, a default value can be set

- If the parameter value is not really a string (although parameters are all sent as strings), the value returned from `getParameter` must be converted, as on the next page...

7.5 Applet Parameters (continued)

```
String pString = getParameter("size");  
if (pString == null)  
    mySize = 24;  
else  
    mySize = Integer.parseInt(pString);
```

- The best place to put the code to get parameter values is in `init`

- Parameters are stored in instance variables

→ SHOW [Wel3.java](#) and [Wel3.html](#)

7.6 Simple Graphics

- *Coordinate system:* (0, 0) is at the upper left corner

- The methods that draw graphic figures are called through the `Graphics` object (the parameter to `paintComponent`)

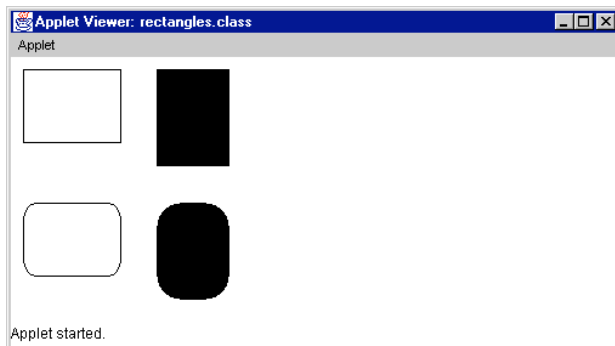
- Lines are drawn with `drawLine(x1, y1, x2, y2)`

- Draws a line from (x1, y1) to (x2, y2)

7.6 Simple Graphics (continued)

- Rectangles are drawn with `drawRect` and `fillRect`
 - Both take four parameters, the coordinates of the upper left corner of the rectangle and the width and height of the rectangle (width and height are in pixels)
- Rectangles with rounded corners can be drawn with `drawRoundRect` and `fillRoundRect`
 - These two take two more parameters, which specify the numbers of horizontal pixels and vertical pixels in the rounding

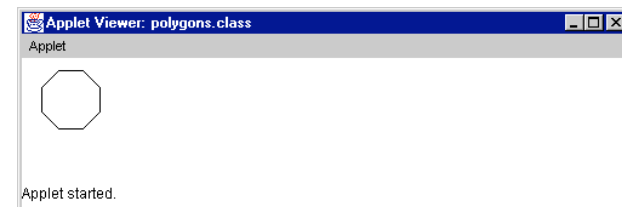
→ SHOW [Rectangles.java](#)



7.6 Simple Graphics (AWT) (continued)

- 3D rectangles can be created with a 5th parameter, `true` (not pushed) or `false` (pushed)
- Polygons are drawn with `drawPolygon`, which takes three parameters, two arrays of coordinates of edge endpoints, and the number of edges

→ SHOW [Polygons.java](#)



- `drawPolygon` can also take a single parameter, which is a `Polygon` object, whose constructor takes the same three parameters as `drawPolygon`
- Ovals are like rectangles (same parameters)

7.7 Color

- The `Color` class has predefined objects for common colors

```
Color.white, Color.black, Color.gray,  
Color.red, Color.green, Color.blue,  
Color.yellow, Color.magenta, Color.cyan,  
Color.pink, Color.orange
```

- An object for any color can be created with the `Color` constructor, as in

```
Color myColor = new Color(x, y, z);
```

- The color of the `Graphics` object can be set with `setColor`, as in

```
grafObj.setColor(Color.cyan);
```

- The foreground and background colors of the applet display are set with methods from `JPanel`

7.8 Interactive Applets

- Java Swing GUI Components (widgets)

1. Labels

- `JLabel` objects are static strings

```
final JLabel lab11 = new JLabel(  
    "Click this button");
```

2. Plain buttons

```
JButton myButton = new JButton(  
    "Click here for fun");
```

3. Checkboxes

```
JCheckbox box1 = new JCheckbox("Beer");  
JCheckbox box2 = new JCheckbox("Pretzels");
```

- `JCheckbox` can take a second parameter, a **Boolean**, that specifies the the initial checkness of the box

7.8 Interactive Applets (continued)

4. Radio buttons – JRadioButton objects in a ButtonGroup

```
ButtonGroup drink = new ButtonGroup();
JRadioButton box1 =
    new JRadioButton("Coke", true);
JRadioButton box2 =
    new JRadioButton("Pepsi", false);
drink.add(box1);
drink.add(box2);
```

5. Text Boxes – JTextField objects

```
JTextField age = new JTextField(3);
```

- Could take a different first parameter, a string literal, which appears in the box when the box is initially displayed

7.8 Interactive Applets (continued)

- A panel object is needed to contain components
- In this case, the panel can be created in the applet

```
JPanel myPanel = new JPanel();

myPanel.setBackground(Color.yellow);
myPanel.setForeground(Color.blue);

myPanel.add(box1);
```

- Layout Managers

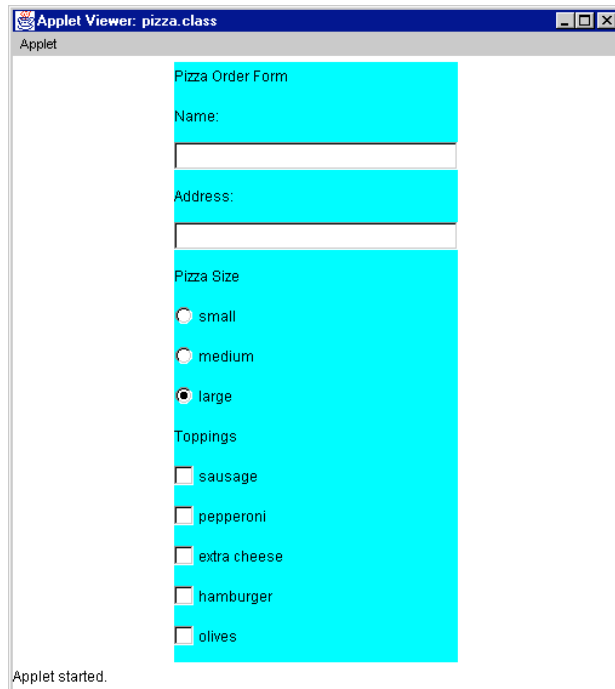
- Default for Swing is BorderLayout – places components on the borders of the panel
- GridLayout is similar to HTML document panels

```
JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(
    new GridLayout(3, 2, 10, 10));
```

- Three rows of two components each, with 10 pixels between the components

→ SHOW [Pizza.java](#)

7.8 Interactive Applets (continued)



7.8 Interactive Applets (continued)

- *The Java Event Model*

- Related to the JavaScript event model
- Event handlers are called *event listeners*
- Connection of an event to a listener is established through event listener registration
 - Done with a method of the class that implements the listener interface
 - The panel object that holds the components can be the event listener for those components
- Event generators send messages (call methods) to registered event listeners when events occur
- Event handling methods must conform to a standard protocol, which comes from an interface
- We only consider the “semantic” events
(there are also “low-level” events)

7.8 Interactive Applets (continued)

- Semantic Event Classes

ActionEvent	click a button, select from a menu or list, or type the enter button in a text field
ItemEvent	select a checkbox or list item
TextEvent	change the contents of a text field or text area

- For the two most commonly used events, `ActionEvent` and `ItemEvent`, there are the following interfaces and handler methods:

Interface	Handler method
ActionListener	actionPerformed
ItemListener	itemStateChanged

- The methods to register the listener is the interface name with “add” prepended

- e.g.,

```
button1.addActionListener(this);
```

7.8 Interactive Applets (continued)

- Event handlers get an event object as a parameter, through which information about the event can be gotten with methods, such as `getState`

- e.g., `button1.getState()` returns `true` if the button is on, `false` otherwise

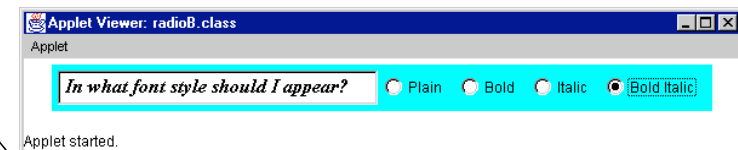
- When an event handler has just a few lines, it can be implemented as an instance of an anonymous nested class

- Example: a button that sets a font

```
button.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        text.setFont(newFont);  
    }  
});
```

→SHOW [RadioB.java](#) Run [RadioB](#)

- Note: It does not use an inner class for the handler



7.9 Concurrency in Java

- Our only interest in concurrency here is to illustrate how threads can be used to create animation in an applet
- A *thread of control* is a sequence of program points reached as execution flows through the program
- A nonconcurrent program has a single thread of control; a concurrent program has more than one
- Java supports lightweight concurrency through its threads
- The concurrent program units in Java are methods named `run`, whose code can be in concurrent execution with other `run` methods and with `main`
- There are two ways to implement threads, as a subclass of `Thread` and by implementing the interface `Runnable`
- The `Thread` class
 - Two essential methods, `run` and `start`
 - `run` is the concurrent method
 - `start` tells the `run` method to begin execution

7.9 Concurrency in Java (continued)

- All Java programs run in threads
 - For applications, when execution is to begin, a thread is created for `main` and its `start` method is called
 - For applets, when the browser finds one, it creates a thread and calls the applet
- SHOW [Names.java](#), output, [NamesDelayed.java](#), →and output
- *Thread States*
 - New - created, but `start` hasn't been called
 - Runnable or ready - ready to run, but is not currently running
 - In the ready queue
 - Running - actually has the processor
 - Blocked - was running, but is not now, because it was interrupted (i/o, end of time slot, gave up its time slot, etc.)
 - Dead - either its `stop` was called or its `run` method completed its execution

7.9 Concurrency in Java (continued)

- Thread methods
- `yield` is a request from the running thread to give up the processor; a static method
- `sleep(time)` - blocks the thread for at least as many milliseconds as the parameter specifies; also a static method
 - `sleep` can throw `InterruptedException`, which must be caught
- `stop` - now deprecated, because of safety problems
 - Now we override it and just set the thread reference to `null` (destroys the thread)
- *An example* - an animated digital clock
 - An applet must implement `Runnable`, its `start` and `stop` methods, and the `repaint` method of `Graphics`
 - `repaint` is called after the applet display has changed

7.9 Concurrency in Java (continued)

- Our applet is named `Clock`
- Its `start` method creates a new `Thread` object, sending `this` to the constructor. This sets the new `Thread` object's target to the `Clock` object, which forces the thread to get its `run` method from the `Clock` object
- After creating the `Thread` object, `start` is called to start its execution
- The `run` method sets a variable to the currently executing thread, and then loops as long as the thread is `clockThread`
- The loop gets the new time with a new `Date` object and repaints the display every second
- Look at [Clock.java](#)
- Run [Clock](#)

