

Programming graphics

- Need a window – `javax.swing.JFrame`
 - Several essential steps to use (necessary “plumbing”):
 - Set the size – width and height in `pixels`
 - Set a title (optional), and a close operation
 - Make it visible
 - See [EmptyFrameViewer.java](#) (p. 59)
- Add `javax.swing.JComponents` to window
 - Draw shapes, colors, ... on these components
- That’s all there is to it!
 - Except for the painstaking labor, of course

Drawing rectangles for example

- Define class that `extends JComponent`
 - Or a subclass like `JPanel` for additional features
- Implement `paintComponent` method
 - Use `Graphics` object passed to this method
 - Actually a `Graphics2D` object since Java 1.2
 - Then let that object draw `Rectangle` objects
 - See [RectangleComponent.java](#) (p. 61)
- Add the component to a frame for viewing
 - e.g., [RectangleViewer.java](#)

java.awt.Graphics2D

- Is a subclass of `java.awt.Graphics`
 - So cast is allowed; and `Graphics` methods inherited
 - If don't cast, must use primitive drawing methods:
 - e.g., `drawRect(int, int, int, int)`,
`fillOval(int, int, int, int), ...`
 - i.e., not object-oriented – so lots of work to use/reuse
- But `Graphics2D` can do a *lot* more stuff
 - e.g., `draw(java.awt.Shape)` draws any `Shape`, including `Rectangle`, `Ellipse2D`, `Polygon`, ...
 - `fill(Shape)` draws and fills `Shape` with current color

Drawing more complex shapes

- Text example (p. 114-116) – Car.java
 - Acts like a `Car` that can draw itself
 - `Car` constructor sets `x` and `y` locations
 - Includes `draw(Graphics2D g2)` method
 - Lets `Graphics2D` object draw lines, ellipses, rectangles
- A class like CarComponent.java just uses it:

```
Car myCar = new Car(x, y);  
myCar.draw(g2); // passes reference to graphics object
```
- Still need a view window, like CarViewer.java

Color

- *Current* color applies to text, lines, and fills:

```
g2.setColor(Color.RED);  
g2.draw(...); // draws ... in red  
g2.setColor(Color.BLUE);  
g2.fill(...); // fills ... with blue
```

- Custom colors available:

- Can set by `float` values in range 0.0F to 1.0F:

```
Color gb = new Color(0.0F, 0.7F, 1.0F);  
g2.setColor.gb);
```

- Or by `int` values in range 0 to 255:

```
Color bg = new Color(0, 255, 175);
```

Rendering text with Graphics2D

- Actually necessary to “draw” the text at a specified location on the Graphics object
 - `g.drawString(aString, x, y)` – uses current rendering context (e.g., color), and current text attributes (e.g., font)
- Font: a *face* name, a *style*, and a *point size*

```
Font f =  
new Font("Serif", Font.BOLD, 24);  
g2.setFont(f); // sets font for g2
```

Applets – an alternate approach

- A way to run a program – but *not an application*
 - No `main` method necessary
- Need a **subclass** of `Applet` (or `JApplet`)
 - So: `class __ extends Applet` (or `extends JApplet`)
- Most web browsers know how to create a new applet, and how to use certain `Applet` methods
 - So, applets must be embedded in an **html** page
 - And, to be useful, they must include at least one of the methods the browser invokes (e.g., `paint`)

“Running” an Applet

- The applet is started by the web browser as soon as the web page (html file) is visited
- The html file (stands for hypertext markup language) — must have an applet tag in it:

```
<applet code=AppletClassName.class  
        width=### height=###>  
</applet> <!-- needs a closing tag -->
```

- The browser works in a certain order:
 - Creates a new applet object – includes a window, a Graphics object, lots of class Applet methods
 - Invokes (1) `init` – once, (2) `start` – first & return visits, (3) `paint` – first & every need to paint (also `stop`, `destroy`)

Implementing a “simple” applet

- `import javax.swing.JApplet; // mandatory`
 - Also usually `Graphics` and `Graphics2D` and others
- Declare a class that extends `JApplet`:
`public class RectangleApplet extends JApplet`
- Implement `paint` method (at least)
 - Same procedures as `paintComponent` for components
- Create an html file to load the applet in a web browser or the `appletviewer` (provided with JDK)
- See [RectangleApplet.java](#) (p. 63) and related html files (p. 64)

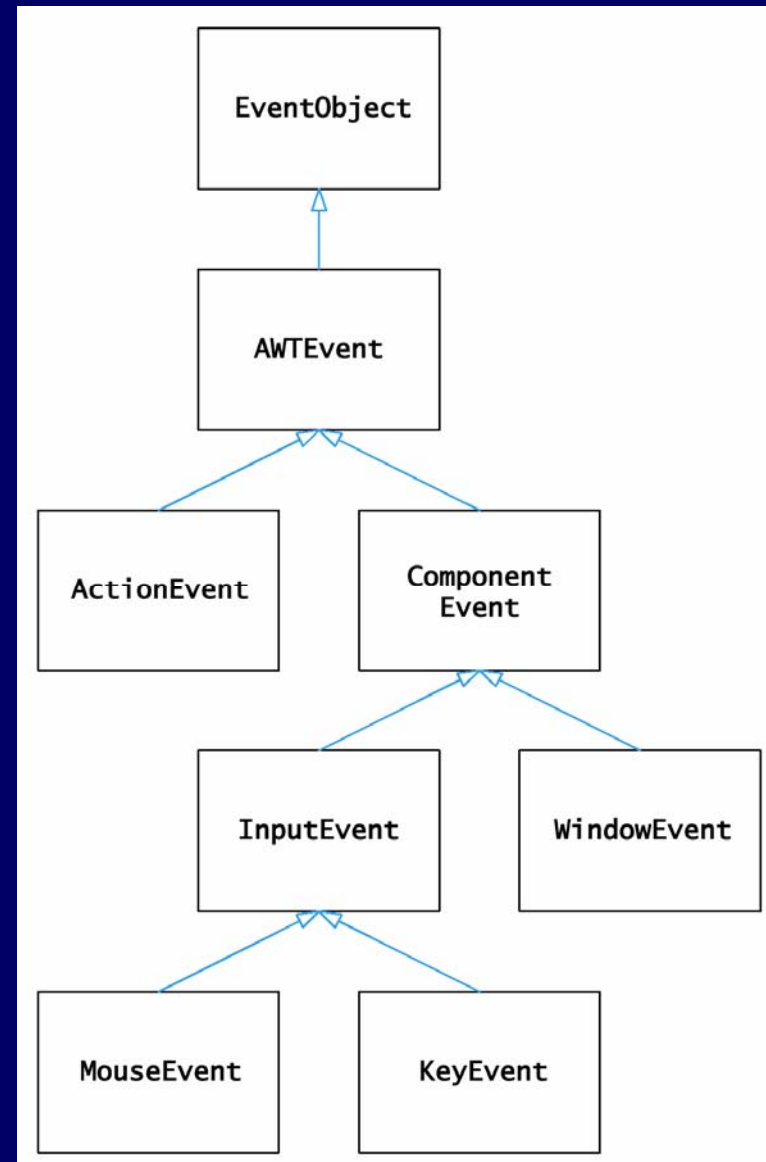
Images

- Images are not drawings
 - You don't draw them, you show them
 - But `Graphics` method is called `drawImage` anyway
- Image is an abstract class
 - Generally, create instance by loading from file or URL
 - Can also create/edit by classes in `java.awt.image`
- Applets know how to get images ([ImageApplet](#))
 - Applications use `Toolkit` to get (demo)

```
Toolkit.getDefaultToolkit().getImage("javacup.gif");
```

Events

- Signals from the outside
 - Usually user initiates:
 - Mouse click, button push, menu-select, enter text, ...
 - Not always: [TimerTester.java](#)
- Each event has a type – a class in `java.awt.event`
 - So each event is an **object**
 - Created *almost constantly* (by Java window manager)



Event sources

- Technically: can be any external signal
 - Including from a different program or system
 - e.g., a “client” applet contacting a web “server”
 - In a GUI, focus is on user signals
- Java `Components` generate the events
 - Each component generates different event types, e.g.:
 - Panels (inc. Applets) → `MouseEvents`
 - Textfields and buttons → `ActionEvents`
 - Windows → `WindowEvents`
 - Keyboard → `KeyEvents`
 - Components *inform* a list of `Listeners` about events

Listeners

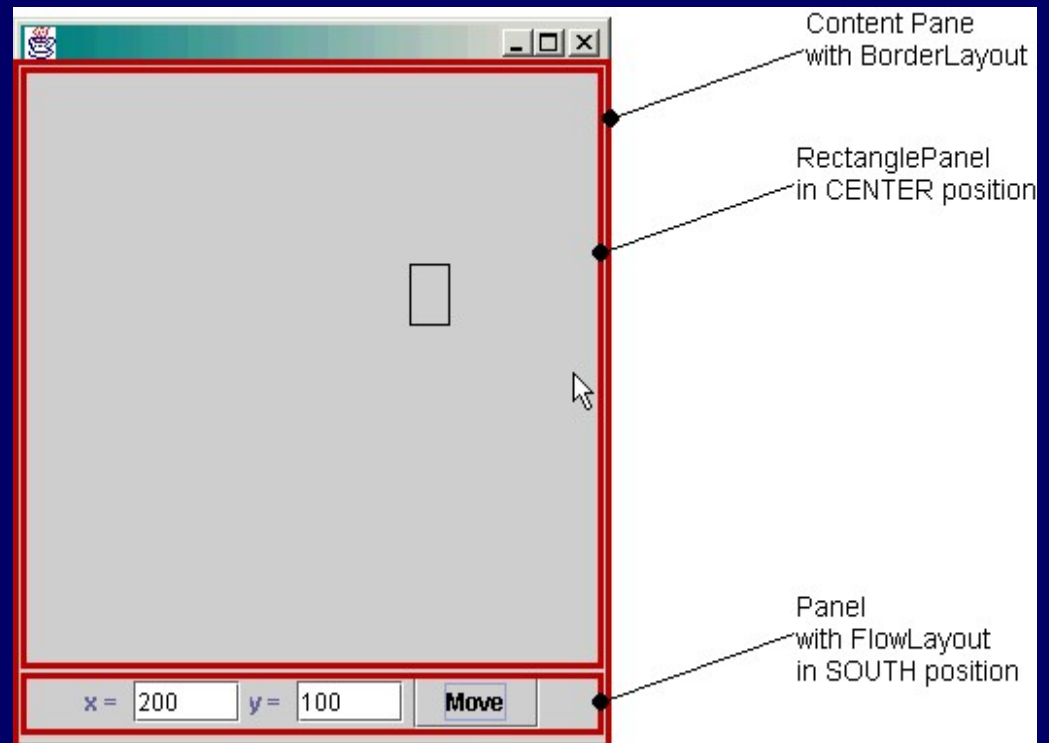
- Objects that components tell about events
 - Must have methods a component can invoke
 - i.e., *implements* one of the `Listener` interfaces
 - Must be `added` to a component's listener list
- Different listeners apply to different messages
 - `ActionListener` – `actionPerformed`
 - e.g., [ClickListener](#) (with [ButtonTester.java](#))
 - `MouseListener` – `mouseClicked`, `mouseEntered`, ...
 - e.g., [MouseSpy demo](#) (from 1st edition of textbook)
- Note: also can extend `Adapter` class instead of implement listeners directly – saves busywork ☺

Inner class listeners

- Can access private instance variables of outer class
 - Stores implicit reference to outer class object
 - So can often “handle” events more easily
 - e.g., [RectangleComponentViewer.java](#)
- Notice they can access `final` local variables too
 - Another example: [TimerTester2.java](#)
 - *Must* be `final` so no ambiguity about value
 - For example, no opportunity for variable to go out of scope while object exists

Laying out GUI components

- Depends on layout manager
 - Defaults:
 - JFrame: BorderLayout
 - JPanel: FlowLayout
 - Others:
 - GridLayout
 - GridBagLayout
- Can set new, and even make custom



Choosing a layout manager

- e.g., a grid layout for calculator buttons:

```
panel.setLayout( new  
    GridLayout(4,3) );  
panel.add(button7);  
panel.add(button8);  
panel.add(button9);  
panel.add(button4);  
...
```

- e.g., [CS10Display.java](#)



Text components

- `JLabel` – not for user input, just display
- `JTextField` – for one row of text
 - `new JTextField()`, or `(int columns)`
 - `getText()`, `setText(String)`, `setFont(Font)`, `setEditable(boolean)`, ... (mostly inherited)
 - `ActionEvent` on `<enter>`
- `JTextArea` – for multiple rows of text
 - `new JTextArea()`, or `(int rows, int columns)`
 - Same methods inherited from `JTextComponent`
 - Generates no events, so usually use with a button

Choices

- Choice objects generate `ActionEvents`
- Handlers for `JCheckBox` and `JRadioButton`: use boolean method `isSelected()`
 - Note: put radio buttons in a `ButtonGroup` – so just one can be selected at a time
 - For same reason – should visually group them in a panel with a border, like `EtchedBorder`
- For `JComboBox`: use `getSelectedItem()`
 - Note: returns `Object` – usually cast to `String`
- e.g., [ChoiceFrame.java](#) (see `FontViewer.java`, pp. 794-798)

Menu

- Steps to implement swing menus:
 - 1. Add a **menu bar** to a JFrame

```
JMenuBar bar = new JMenuBar();
setJMenuBar(bar);
```
 - 2. Add **menus** to the menu bar

```
JMenu fileMenu = new JMenu("File");
bar.add(fileMenu);
```
 - 3. Add **menu items** to the menus, & listeners to items

```
JMenuItem openItem = new JMenuItem("Open");
fileMenu.add(openItem);
openItem.addActionListener(listener);
```
- e.g., [MenuFrame.java](#) (see FontViewer2.java, pp. 803-7)

Sliders, and more swinging

- Note - good text section 18.4, pp. 808-814: should read while browsing API on web
 - About how to “discover” `swing` features/usage
 - Focuses on `JSlider` – generates `ChangeEvent`s, so `addChangeListener(listener)`, where `listener` implements `stateChanged` method
 - Requires `javax.swing.event` for “change” events
 - e.g., [SliderFrame.java](#) (see `ColorViewerFrame.java`, pp. 812-814)
- Explore `swing` as needed, or even just for fun
 - Buy a book, or look at API classes starting with “J”
 - Or just run the [SwingSet](#) demo from the JDK