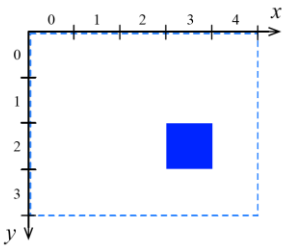
Simple Graphics Programming

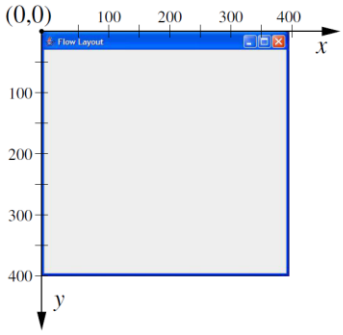
Graphics Coordinate System



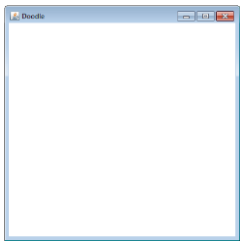
Pixel at (3,2)

Width of area 5 × 4

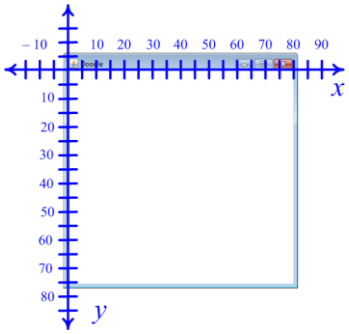
Graphics Coordinate System



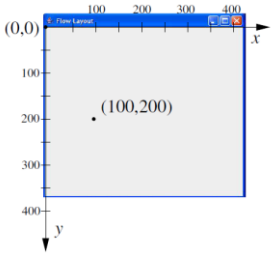
`JPanel` Coordinate System



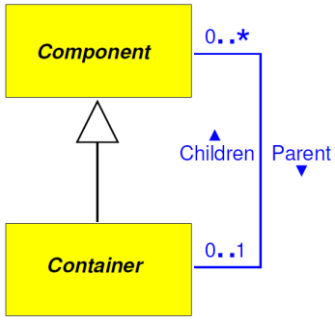
`JPanel` Coordinate System



`JPanel` Coordinate System



Component Containers



8

Selected `Graphics` class methods

`setColor``(Color c)`
Sets this graphics context's current color to the specified color

`drawLine``(int x1, int y1, int x2, int y2)`
Draws a line, using the current color, between the points `(x1, y1)` and `(x2, y2)` in this graphics context's coordinate system

`drawRect``(int x, int y, int width, int height)`
Draws the outline of the specified rectangle

`drawOval``(int x, int y, int width, int height)`
Draws the outline of an oval

`fillRect``(int x, int y, int width, int height)`
Fills the specified rectangle

`fillOval``(int x, int y, int width, int height)`
Fills an oval bounded by the specified rectangle with the current color

`fillPolygon``(int[] xPoints, int[] yPoints, int nPoints)`
Fills a closed polygon defined by arrays of x and y coordinates

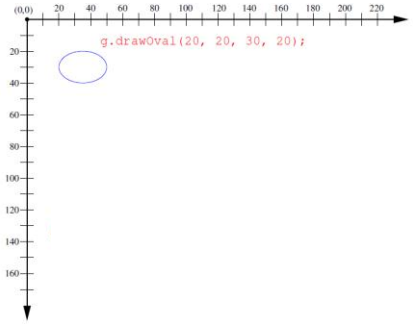
`drawString``(String str, int x, int y)`
Draws the text given by the specified string, using this graphics context's current font and color

10

Selected `Graphics` class methods

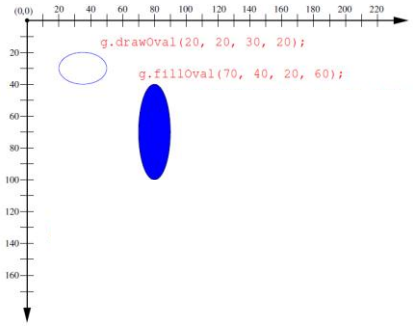
12

Selected `Graphics` class methods



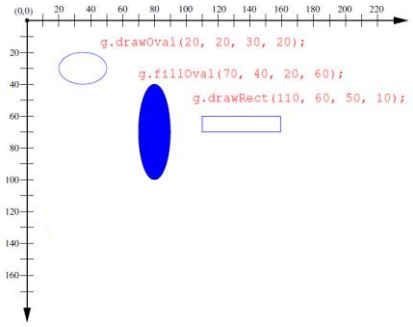
13

Selected `Graphics` class methods



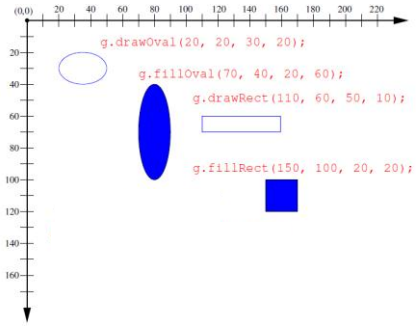
14

Selected `Graphics` class methods

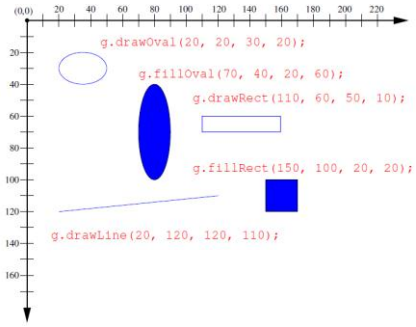


15

Selected `Graphics` class methods

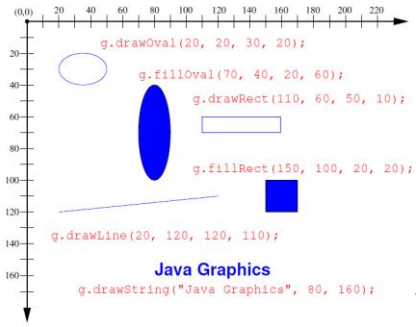


Selected `Graphics` class methods



17

Selected `Graphics` class methods



18

Blocking User Input

- Console-based user input

```
Scanner scan = new Scanner(System.in);
int x = scan.nextInt();
```

- Even some graphical user input

```
String input = JOptionPane.showInputDialog("Enter value");
int x = Integer.parseInt(input);
```

19

Event-driven Programming

- An event can occur at any time
  - User input is only one kind of event
- At any given time the program must be able to respond to a diverse set of possible events
- The traditional, console-based approach is incapable of handling this style of program behavior

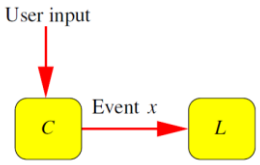
20

Examples of Events

- User mouse movements, mouse button clicks
- User keystrokes
- User pressing a graphical button
- User selecting a menu item
- User typing into a text field
- Timer expiration
- Request from window manager or program itself to repaint the component

21

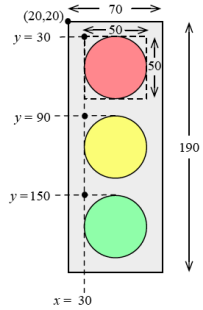
Java's Event Model



`JPanel` class input methods

Event	Method
Mouse button pressed	`mousePressed()`
Mouse button released	`mouseReleased()`
Mouse button clicked	`mouseClicked()`
Mouse pointer moved over viewport from outside	`mouseEntered()`
Mouse pointer moved out of viewport	`mouseExited()`
Mouse moved while button depressed	`mouseDragged()`
Mouse moved with no buttons depressed	`mouseMoved()`
Key typed	`keyTyped()`

Better traffic light visualization

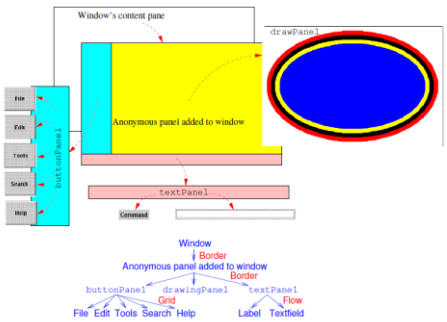


Anonymous Inner Classes

- Make a subclass and an instance of that subclass on the fly
- No need to make new named class
- Syntax may look odd initially, but it really makes sense

Layout Managers

- `FlowLayout`
- `BorderLayout`
- `GridLayout`
- `BoxLayout`
- No layout (`null`)



Next . . .

Some standard classes . . .

28