CS18000: Problem Solving and Object-Oriented Programming

Complex GUIs

Video 1
GUI Concept

Constructing Graphical User Interfaces

Frames

Panels

Widgets

Review

- Review from earlier lecture on GUIs
  - JOptionPane
  - JFileChooser
- One-shot, pop-up dialogs

Paradigm Shift: User in Charge

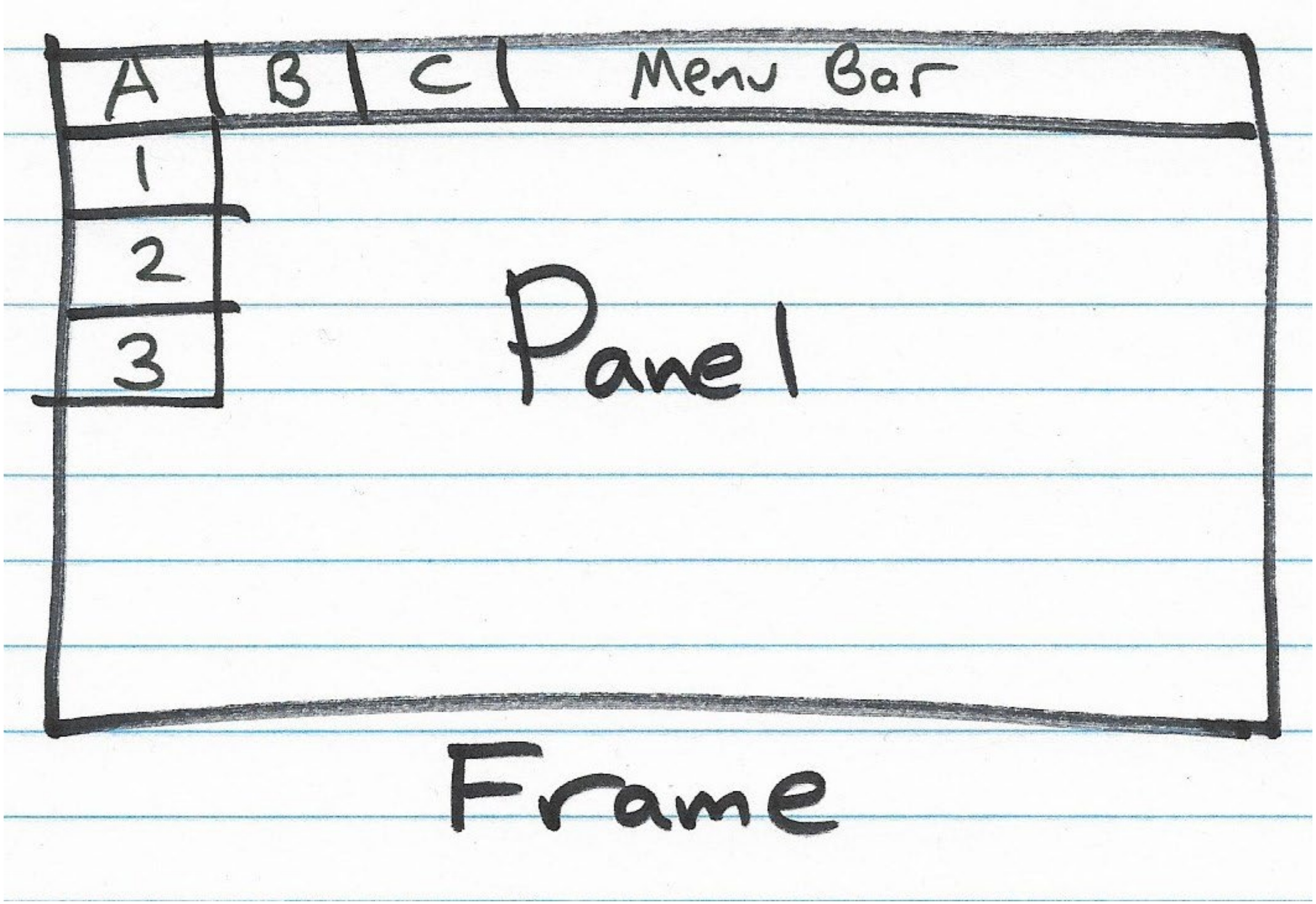
- Text-Based Interface: program prompts, user responds
- Graphical User Interface (GUI): user directs program what to do next
- Program must respond to a variety of user-initiated events
  - Keystrokes
  - Button clicks
  - Mouse movements

Model-View-Controller

- A software paradigm for constructing GUIs
- Not rigid: Has many variations
- Divides responsibilities into three pieces:
  - *Model*: the data (or database) underlying the application
  - *View*: the GUI components visible to the user
  - *Controller*: the “glue” that implements the “business rules” of the application
- Controller…
  - updates view when model changes
  - updates model when user interacts with view
- Idea: Separates responsibilities to manage complexity; allows specialists in each area

GUI Concept: Interface Hierarchy

- A GUI is composed of a hierarchical set of interface elements called *components or window gadgets (widgets)*
- At the top-level is…
  - A *frame*
  - A window that interacts with the user's desktop
- Inside a frame is (among other things)…
  - A *menu bar*
  - A *panel* to layout the top-level components
- Then come the interface widgets…
  - User-interface elements
  - Generally visible to the user
  - Include labels, buttons, text fields, scroll bars, canvases, etc.
  - A panel is also a widget, to permit creation of sub-layouts


A
B
C
Menu Bar
1
2
3
Panel
Frame

GUIs in Java

- Two packages of classes
  - java.awt: original “Abstract Window Toolkit”
  - javax.swing: newer, better and built on AWT
    - These are the “J” classes
    - In most cases, we will use these

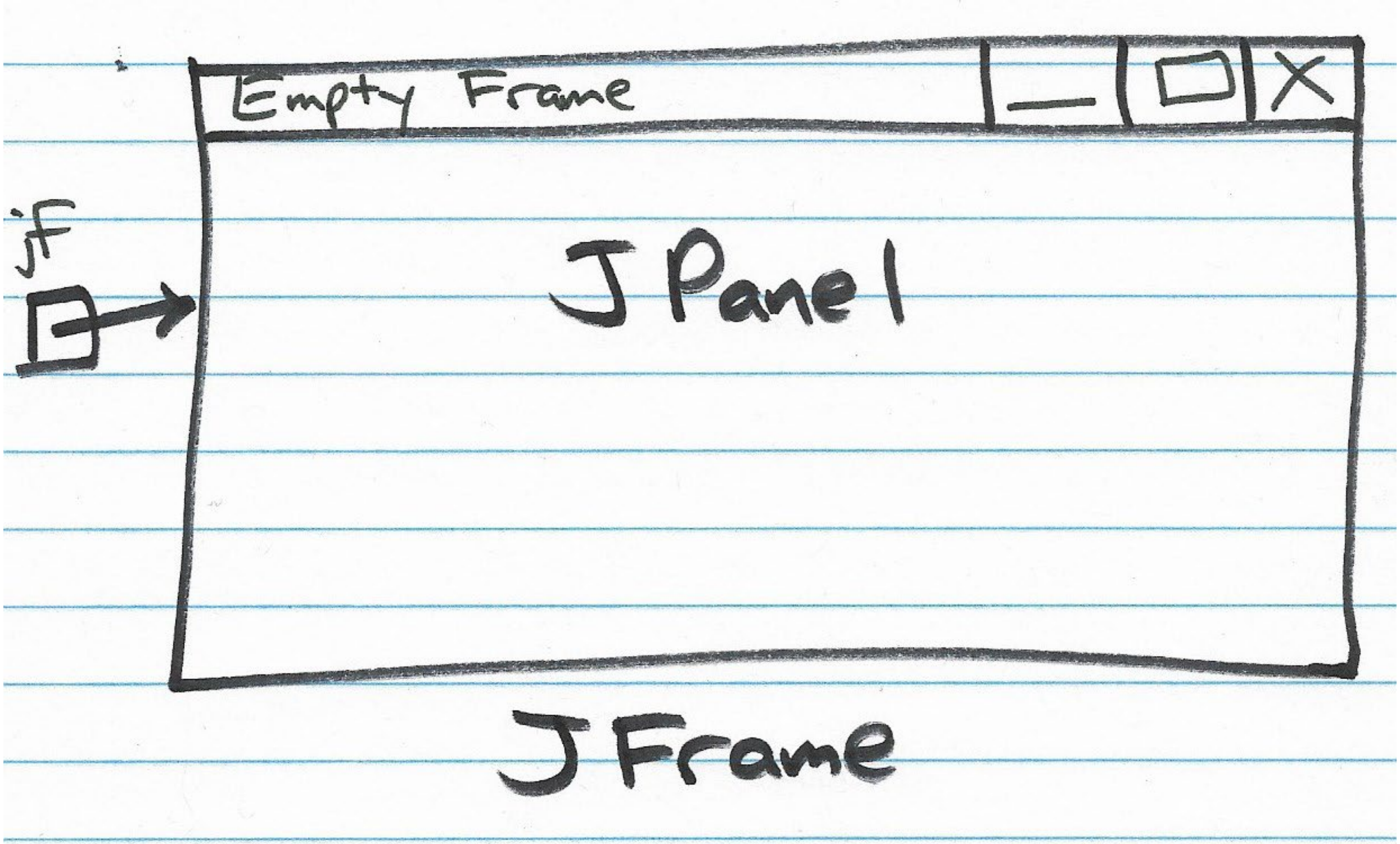
Video 2
JFrames and JPanels

Class JFrame

- Basic top-level window
- Interacts with “window manager”
- Houses and lays out interactive controls
- Two approaches to using:
  - Create raw JFrame object (we will use)
  - Extend JFrame then create object (also common)

Example: EmptyFrame

```
import javax.swing.JFrame;

public class EmptyFrame {
    public static void main(String[] args) {
        JFrame jf = new JFrame("Empty Frame");
        jf.setSize(640, 480);
        jf.setDefaultCloseOperation(
            JFrame.DISPOSE_ON_CLOSE);
        jf.setVisible(true);
    }
}
```


Empty Frame
jF
JPanel
JFrame

JFrame Operations

- setDefaultCloseOperation: window close
  - Use DISPOSE_ON_CLOSE (graceful shutdown)
  - Not EXIT_ON_CLOSE (equivalent to System.exit())
- setSize: set width and height (e.g., 640x480)
- setResizable: true or false
- setVisible: true or false (true also “validates”)
- setTitle: String to appear in title bar
- add: adds components to the component pane

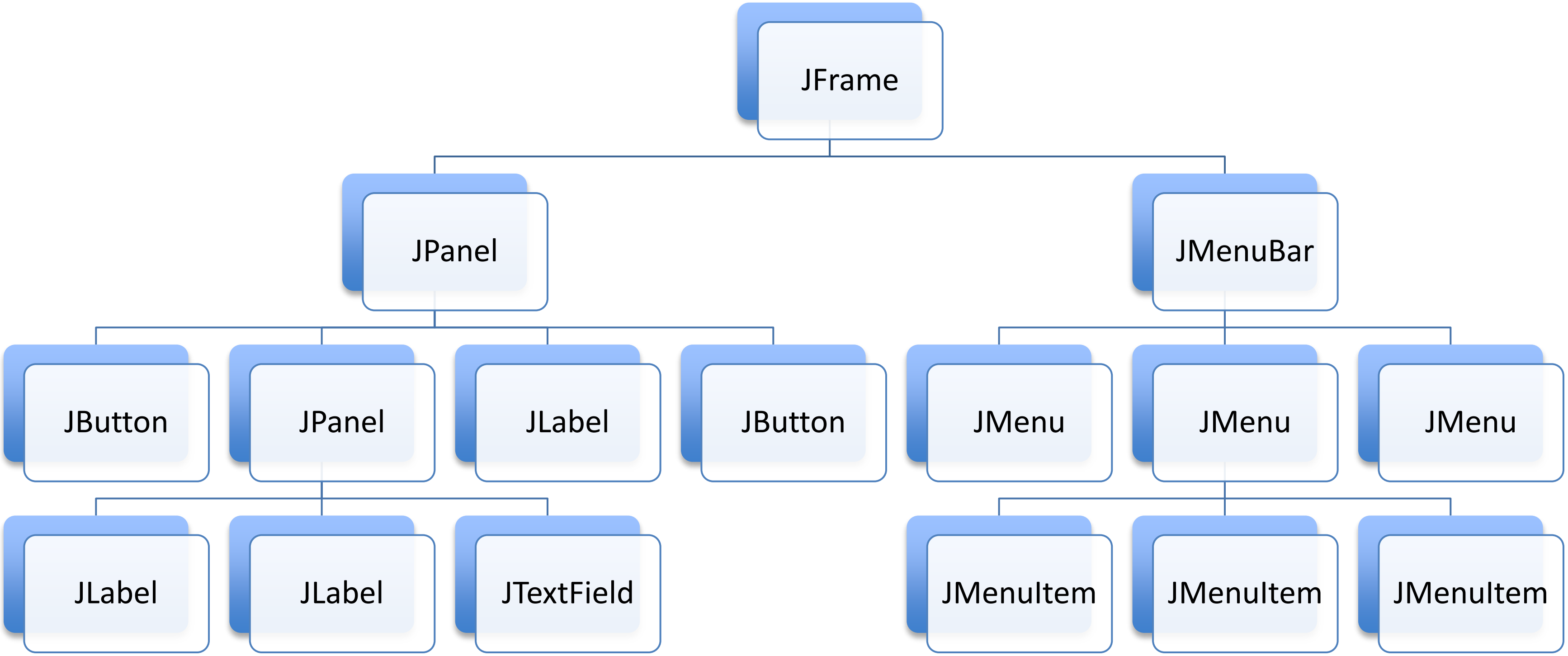
Panels and Layout Managers

- Panels are used to group widgets for layout
- Panels are hierarchical
  - may contain sub-panels
- Layout managers define the rules
  - how widgets and sub-panels are placed relative to one another

Class JPanel

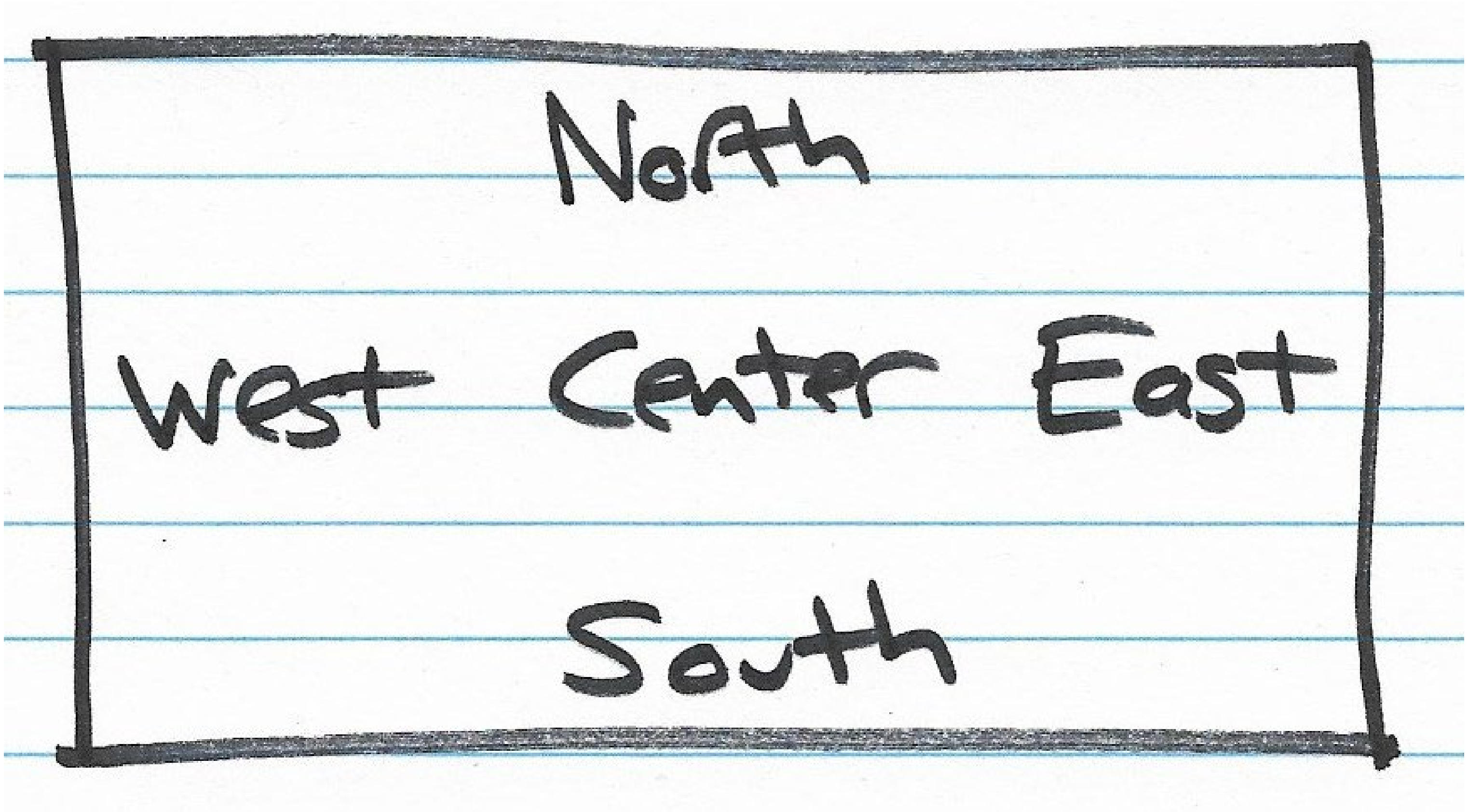
- Java panel class
- Special features of JFrame
  - jf.getContentPane() is a JPanel
  - jf.add(…) automatically adds to content pane
  - Default Content pane layout manager is "BorderLayout"

A Java GUI: A Tree of Components



Example Layout Manager: BorderLayout

- Divides pane into five regions
  - Center
  - North, East, South, West
- Can add one component to each region
- jf.add(component, BorderLayout.CENTER)
- More about layout managers later



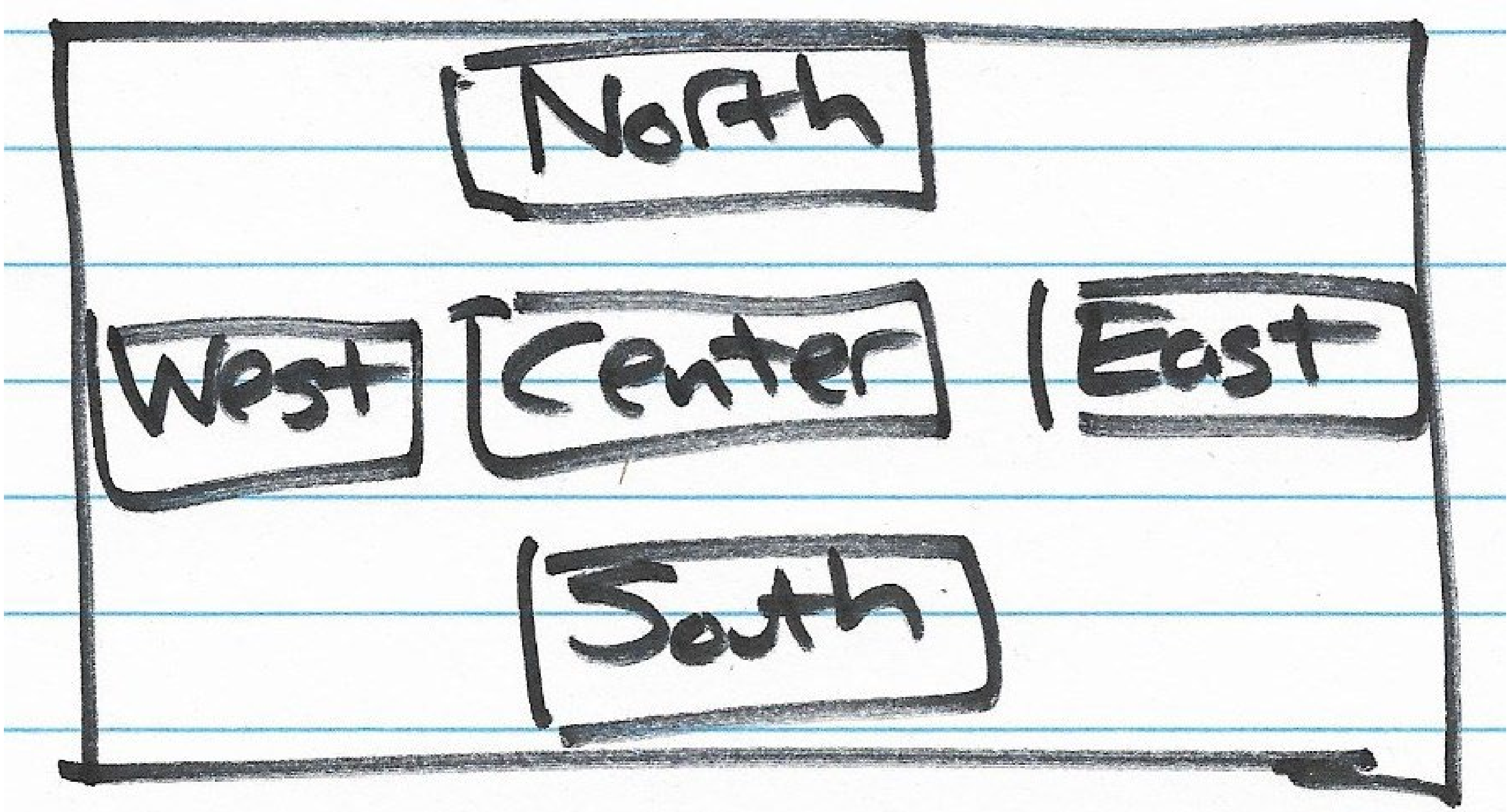
North

West Center East

South

Example: Adding Buttons to Borders

```
JButton jbCenter = new JButton("Center");
JButton jbNorth = new JButton("North");
JButton jbSouth = new JButton("South");
JButton jbEast = new JButton("East");
JButton jbWest = new JButton("West");

jf.add(jbCenter, BorderLayout.CENTER);
jf.add(jbNorth, BorderLayout.NORTH);
jf.add(jbSouth, BorderLayout.SOUTH);
jf.add(jbEast, BorderLayout.EAST);
jf.add(jbWest, BorderLayout.WEST);
```


North
West
Center
East
South

Widgets for Interaction

- JLabel
- JButton
- JTextField
- JTextArea

Also, radio buttons, scroll bars, toggles, …

Video 3
Event Handling

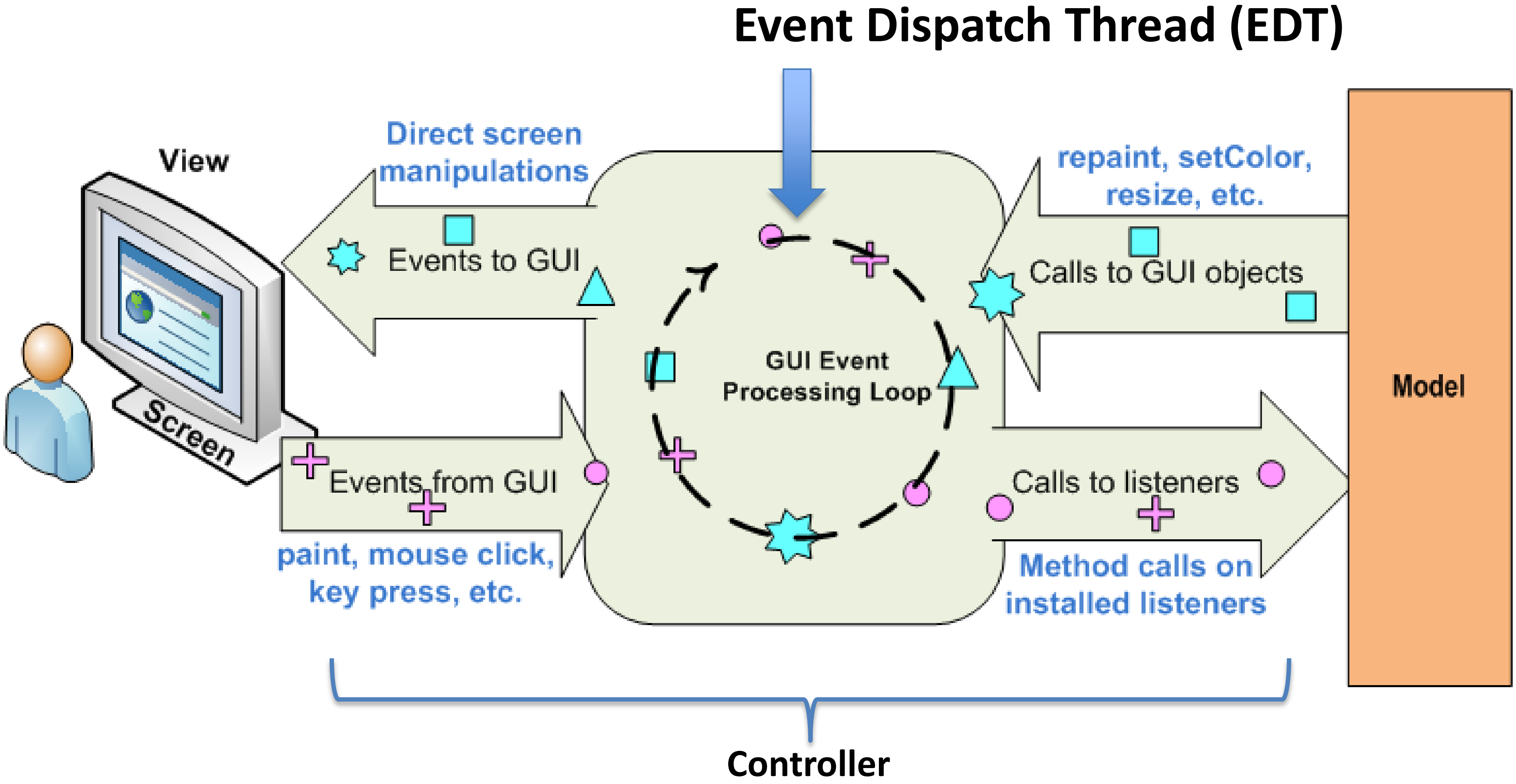
Constructing Graphical User Interfaces

Events

Event Handling

- Events connect the user to your program
- Event sources (from the user)
  - Keystrokes
  - Mouse actions (buttons and movement)
- Event listeners (your program)
  - A method in your code
  - Linked to a widget (or other event source)
  - Processes events generated by that widget

Java Event Handling


Event Dispatch Thread (EDT)
View
Screen
Direct screen manipulations
Events to GUI
Events from GUI
paint, mouse click, key press, etc.
GUI Event Processing Loop
repaint, setColor, resize, etc.
Calls to GUI objects
Calls to listeners
Method calls on installed listeners
Model
Controller

Source: http://www.clear.rice.edu/comp310/JavaResources/GUI/

EDT: The Event Dispatch Thread

- The Java GUI is controlled by its own thread, the EDT
- Typical scenario:
  - Main thread
    - Builds JFrame and lays out interface
    - Makes the JFrame visible
    - Returns from main method; main thread exits
  - EDT continues running
    - Interacts with user
    - Invokes “listeners” (or “call backs”) to handle events
- Thus, your event-handling code runs on the EDT

A Better Way to Launch a JFrame

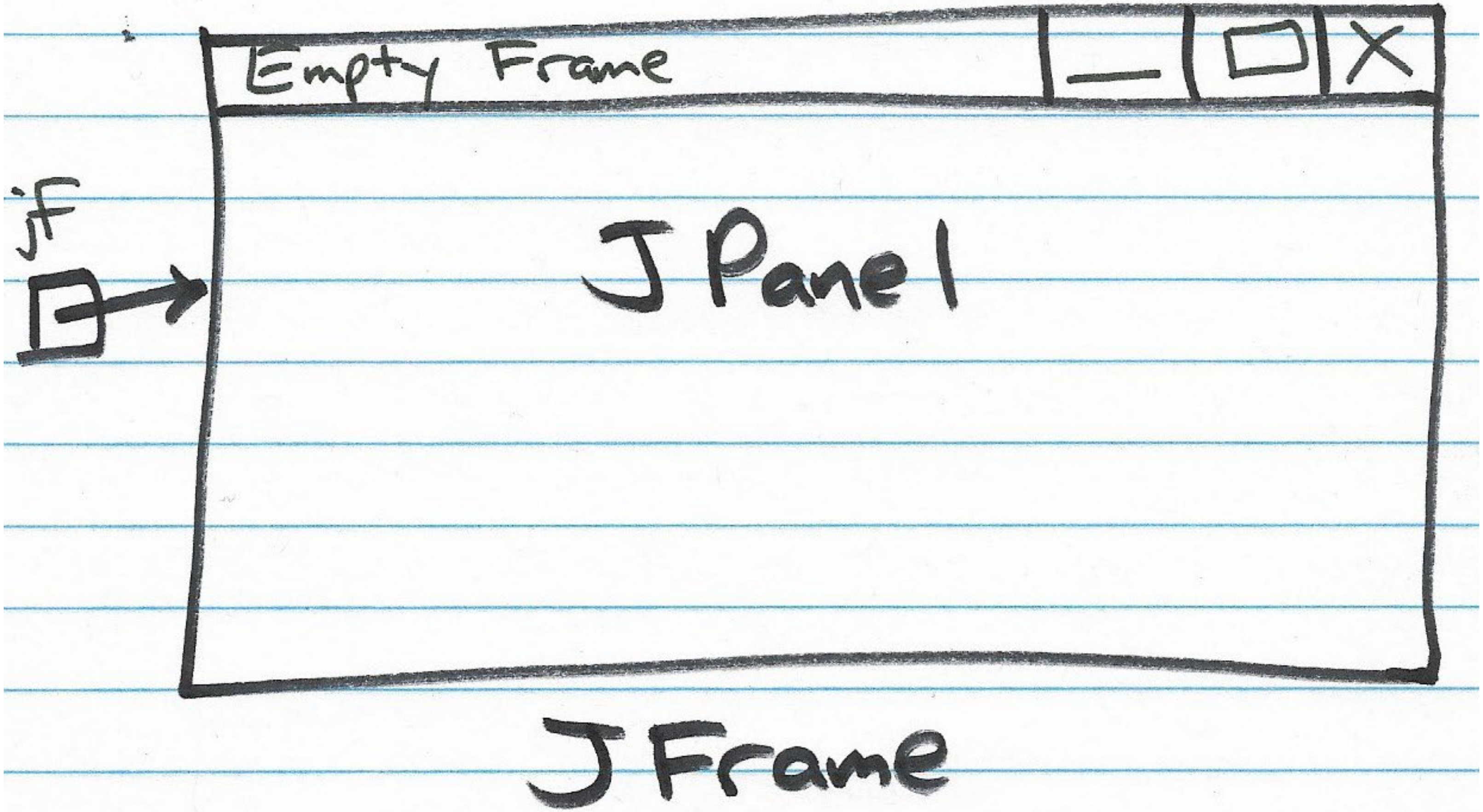
- In slide 12, JFrame launched from main() method
- This usually works ok, but sometimes runs into problems including deadlock
- It is better to launch the JFrame so that it runs on the EDT
- This is done by using SwingUtilities.invokeLater(Runnable method)
- Causes method to be executed on the EDT

Example: EmptyFrame (1)

```
import javax.swing.SwingUtilities;
import javax.swing.JFrame;

public class EmptyFrame {
    public static void main(String[] args) {
// Execute all GUI-related code on the EDT.
// This causes the run() method to execute inside the EDT.
     SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            createGUI();
         }
      });
    }
```

Example: EmptyFrame (2)

```
    public void createGUI() {
        JFrame jf = new JFrame("Empty Frame");
        jf.setSize(640, 480);
        jf.setDefaultCloseOperation(
            JFrame.DISPOSE_ON_CLOSE);
        jf.setVisible(true);
    }
}
```


Empty Frame
jF
JPanel
JFrame

Video 4
ActionListener Interface

Observers (“Listeners”) in Java

- ActionListener (buttons)
- MouseListener (component entry/exit)
- MouseMotionListener (component)
- ItemListener (check boxes)
- DocumentListener (text fields)
- KeyListener (text boxes)

ActionListener Interface

- Must implement:
  - public void actionPerformed(ActionEvent e)
- ActionEvent includes methods:
  - getSource(): widget (object) generating event
  - getActionCommand(): associated string
  - getWhen(): time of event
- source.setActionCommand(String s) sets the String returned by getActionCommand()

Example: PushMe (1)

```
import javax.swing.SwingUtilities;
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JLabel;
import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class PushMe implements ActionListener {
    static JFrame frame;

    public static void main(String[] args) {
      SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            createGUI();
         }
      });
    }
```

Example: PushMe (2)

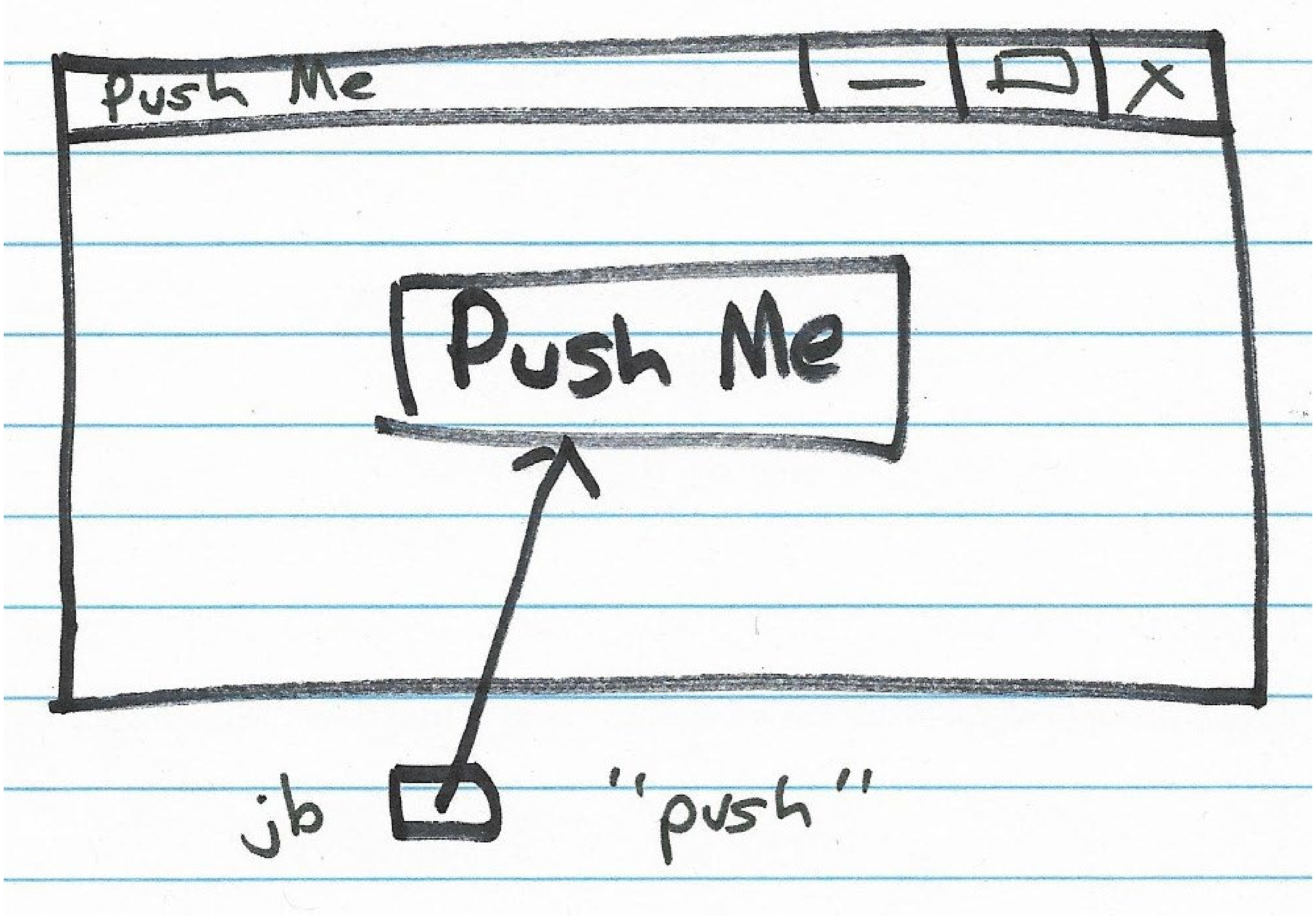
```
public void createGUI() {
       frame = new JFrame("Push Me");
       frame.setSize(200, 100);
       frame.setDefaultCloseOperation
  (JFrame.DISPOSE_ON_CLOSE);
       JButton button = centeredButton();
       button.addActionListener(new PushMe());
       frame.setVisible(true);
       frame.setLocationRelativeTo(null);
   }
```

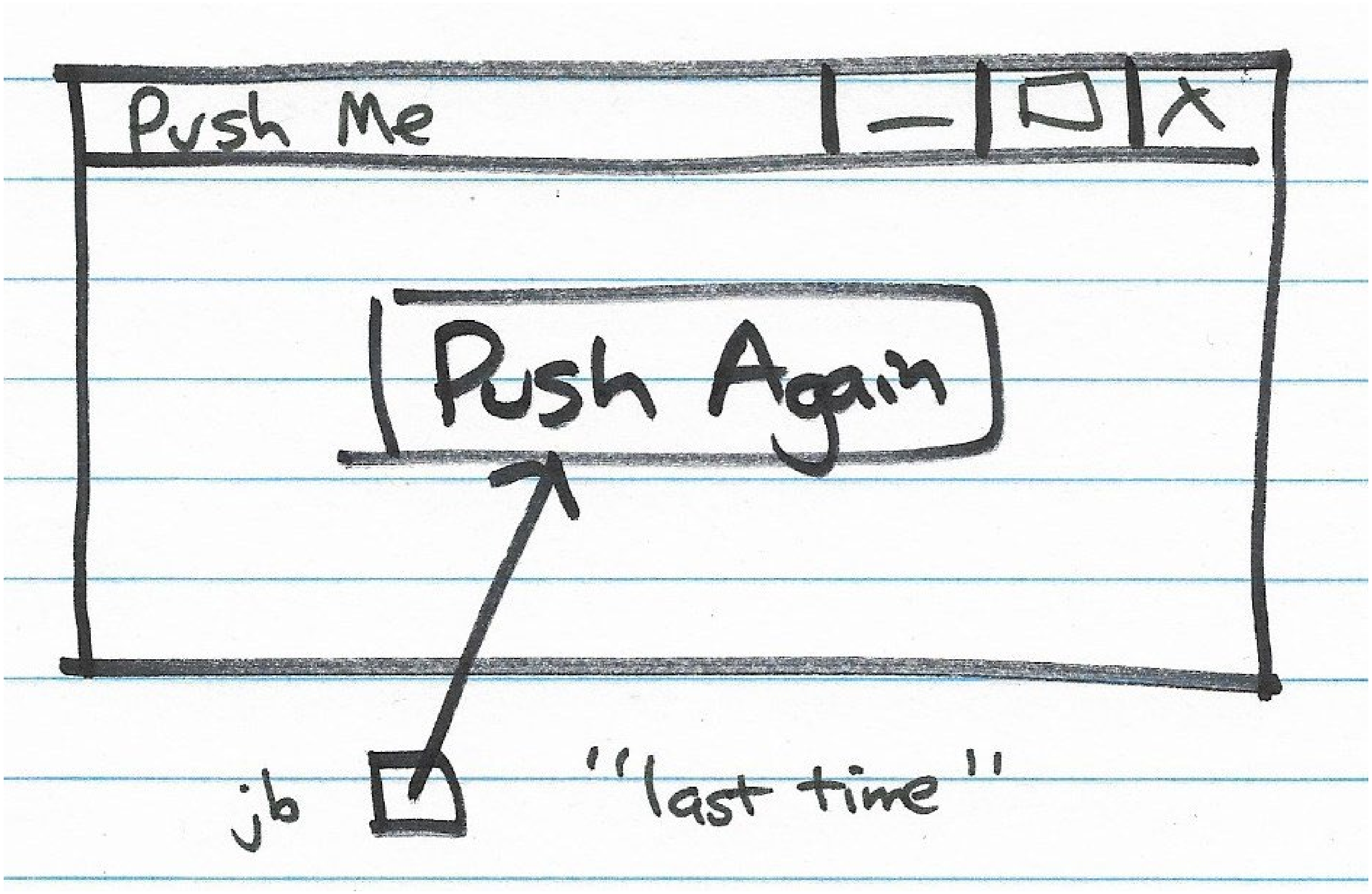
Example: PushMe (3)

```
public void actionPerformed(ActionEvent e) {
    JButton b = (JButton) e.getSource();
    if (b.getActionCommand().equals("last time"))
        frame.dispose();
    if (b.getActionCommand().equals("push")){
        b.setActionCommand("last time");
        b.setText("Push Again");
    }
}
```

Example: PushMe (4)

```
    static JButton centeredButton() {
        String[] location = { BorderLayout.NORTH,
            BorderLayout.EAST, BorderLayout.SOUTH,
            BorderLayout.WEST };
        for (String s : location) {
            frame.add(new JLabel("     "), s);
        }
        JButton jb = new JButton("Push Me");
        jb.setActionCommand("push");
        frame.add(jb);
        return jb;
    }
}
```


Push Me
—
□
X
Push Me
jb
"push"


Push Me
Push Again
jb
"last time"

Video 1
Source to Listener Relationships

Source to Listener Relationships

- One-to-One
  - One event source sends to one listener
  - Simple
- Many-to-One
  - Many event sources send to one listener
  - Allows single piece of code to handle multiple buttons
- One-to-Many
  - One source sends to many listeners
  - Less used, but allows independent actions on same button press

Using the ActionListener

- Technique 1: Create a named class that implements ActionListener
  - Create object
  - Attach object to one or more buttons
- Technique 2: Create an object of a nested class and attach to a button
- Technique 3: Create an object of an unnamed (anonymous inner) class and attach to a button

Example: Implement ActionListener

```
public class ListenerTechnique1 implements ActionListener {
    public static void main(String[] args) {
        // initialization omitted
        JButton button = new JButton("Push Me");
        button.addActionListener(new ListenerTechnique1());
        button.setActionCommand("doit");
        // finish and make visible omitted
    }

    public void actionPerformed(ActionEvent ae) {
        System.out.printf("Button pressed: %s\n",
            ae.getActionCommand());
    }
}
```

Example: Use Nested Class

```
// this class is nested inside main method (for example)…

class OneShot implements ActionListener {
    public void actionPerformed(ActionEvent ae) {
        System.out.printf("Button pressed: %s\n”,
            ae.getActionCommand());
        }
    }

button.addActionListener(new OneShot());
```

One-Shot ActionListener

- Don’t care about name
- Only want to create one object
- Java allows simplification…

- Declare the method, class, create object, and add action listener all in one step!
- Uses Anonymous Inner Class

Anonymous Inner Class

- Declare the method, class, create object, and add action listener all in one step!

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
        // do something with ae
    }
});
```

Video 2
Layout Managers

Constructing Graphical User Interfaces

Layout Managers

Worker Threads

Adapter Classes

- Problem
  - Some interfaces have many methods
  - Your use may only need one of them
  - Interface requires implementations for all
- Solution
  - Adapter class provides default (empty) implementations for all
  - You create a subclass of the adapter class, overriding the ones you want to change

Example: MouseAdapter Class

- Provides implementations for
  - mouseEntered(…)
  - mouseClicked(…)
  - mouseExited(…)
  - mousePressed(…)
  - mouseReleased(…)
- You override only the ones you need

Layout Managers

- Containers like JPanel have a Layout Manager
- Layout Manager called by container to position and size each of its “children”
- Several Layout Managers available…
  - BorderLayout (center plus N-E-S-W)
  - FlowLayout (left-to-right, top-to-bottom)
  - GridLayout (m x n grid of equal size)
  - …others (BoxLayout, GridBagLayout, …)
- In general, re-layout as sizes change

Important

FlowLayout

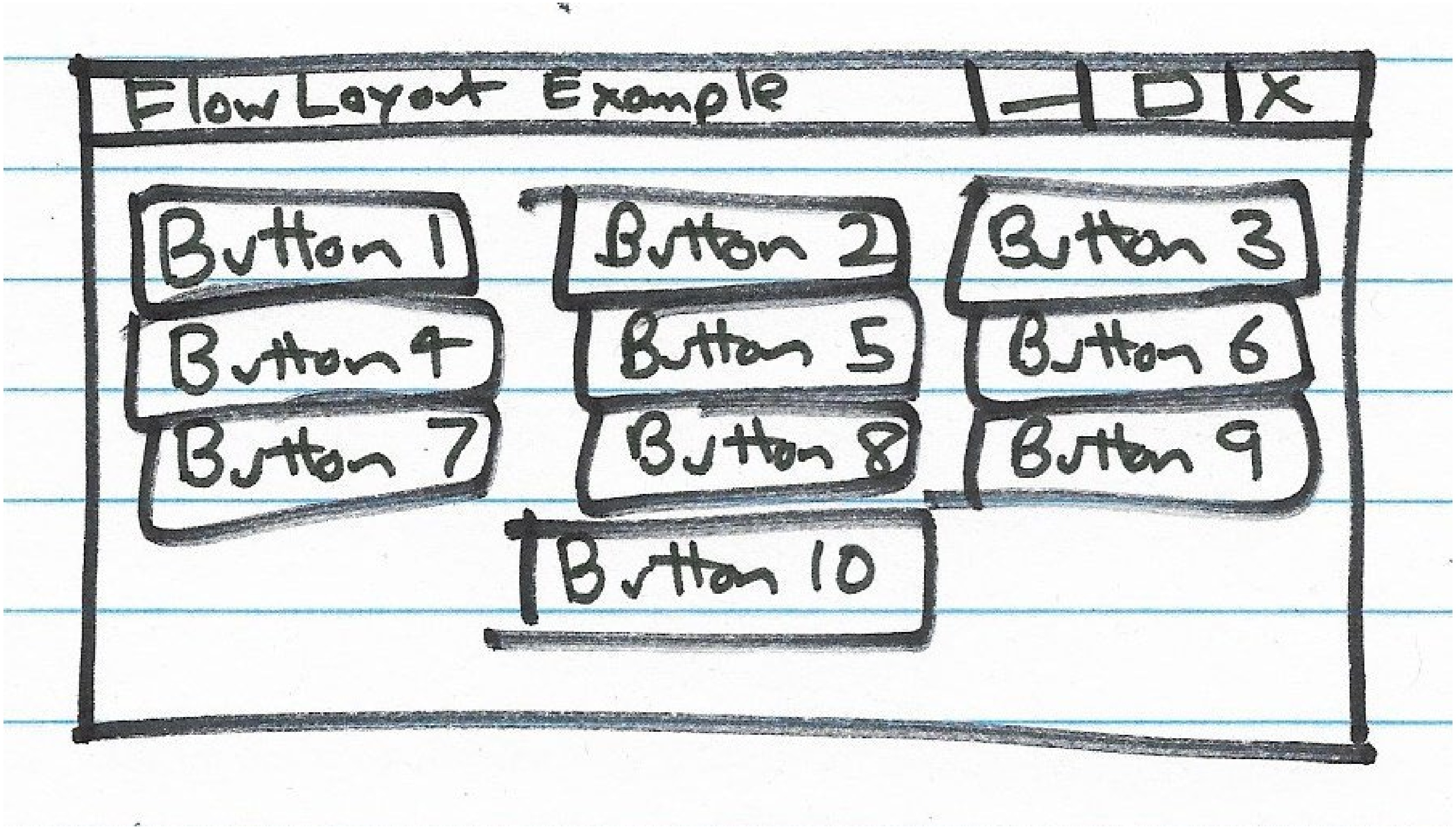
- Default layout manager (except for JFrame content pane)
- Added widgets “flow” together, one after another
- By default…
  - Left to right to fill space, then top to bottom
  - Each line is centered
  - Widgets are left at “preferred” size

Example: FlowLayout (1)

```
import javax.swing.SwingUtilities;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;

public class FlowLayoutExample {
    public static void main(String[] args) {
      SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            createGUI();
         }
      });
    }
```

Example: FlowLayout (2)

```
 public void createGUI() {
        JFrame frame = new JFrame("FlowLayout Example");
        frame.setSize(500, 300);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        JPanel panel = new JPanel(); // defaults to FlowLayout
        for (int i = 1; i <= 10; i++) {
            JButton button = new JButton("Button " + i);
            panel.add(button);
        }

        frame.add(panel);
        frame.setVisible(true);
    }
}
```


FlowLayout Example
Button 1
Button 2
Button 3
Button 4
Button 5
Button 6
Button 7
Button 8
Button 9
Button 10

GridLayout

- Created with rows x cols dimensions
- Added widgets are arranged in given number of rows and columns
- Each component takes all the available space within its cell, and each cell is exactly the same size

```
setLayout(new GridLayout(0,4));
```

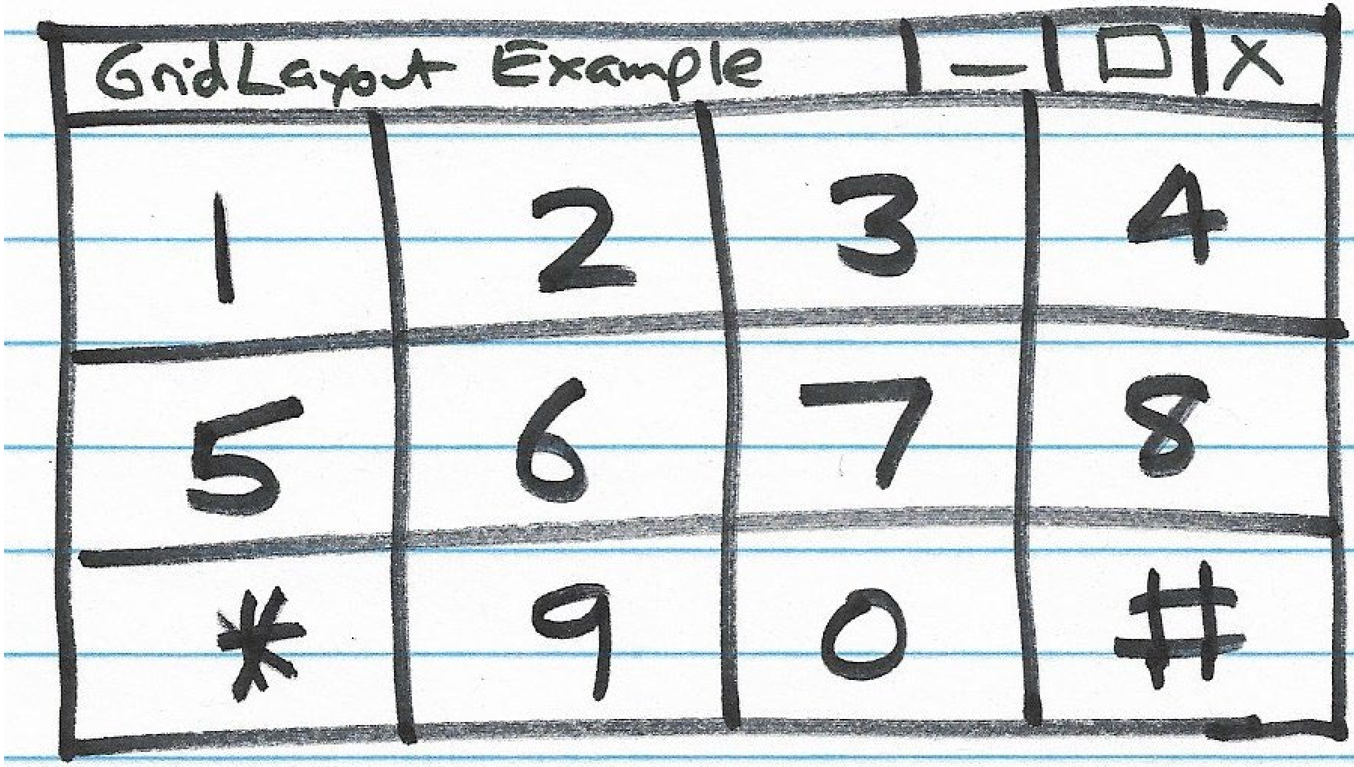
- four columns per row, as many rows as it takes

Example: GridLayout (1)

```
import javax.swing.SwingUtilities;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import java.awt.GridLayout;

public class GridLayoutExample {
    public static void main(String[] args) {
      SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            createGUI();
         }
      });
    }
```

Example: GridLayout (2)

```
  public void createGUI() {
        JFrame frame = new JFrame("GridLayout Example");
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        JPanel panel = new JPanel(new GridLayout(3, 4));

        for (int i = 1; i <= 12; i++) {
            JButton button = new JButton("Button " + i);
            panel.add(button);
        }

        frame.add(panel);
        frame.pack();  // set top-level window to “right” size
to fit
        frame.setVisible(true);
    }
}
```



1	2	3	4
5	6	7	8
*	9	0	#

Changing the JFrame's LayoutManager

What if I don't want to use BorderLayout for the JFrame's top level JPanel?

I can set its LayoutManager to any other as shown below…

```
public void createGUI() {
    JFrame jf = new JFrame();
    JPanel jp = (JPanel) jf.getContentPane();
    jp.setLayout(new FlowLayout());
    jp.add(...); // uses FlowLayout
    ...
}
```

Video 3
BorderFactory

Factory Pattern

- A design pattern for creating objects
- Uses static method rather than “new”
- BorderFactory example:
  - BorderFactory.createLineBorder(Color.RED)
  - BorderFactory.createTitledBorder("Sub Panel")
- Returns a suitable (perhaps “new”) object
- Allows reuse of “read-only” (shared) objects

Example: Using Sub-Panels (1)

```
import javax.swing.SwingUtilities;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.BorderFactory;
import javax.swing.border.Border;
import java.awt.Color;

public class SubPanelExample {
    public static void main(String[] args) {
      SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            createGUI();
         }
      });
    }
```

Example: Using Sub-Panels (2)

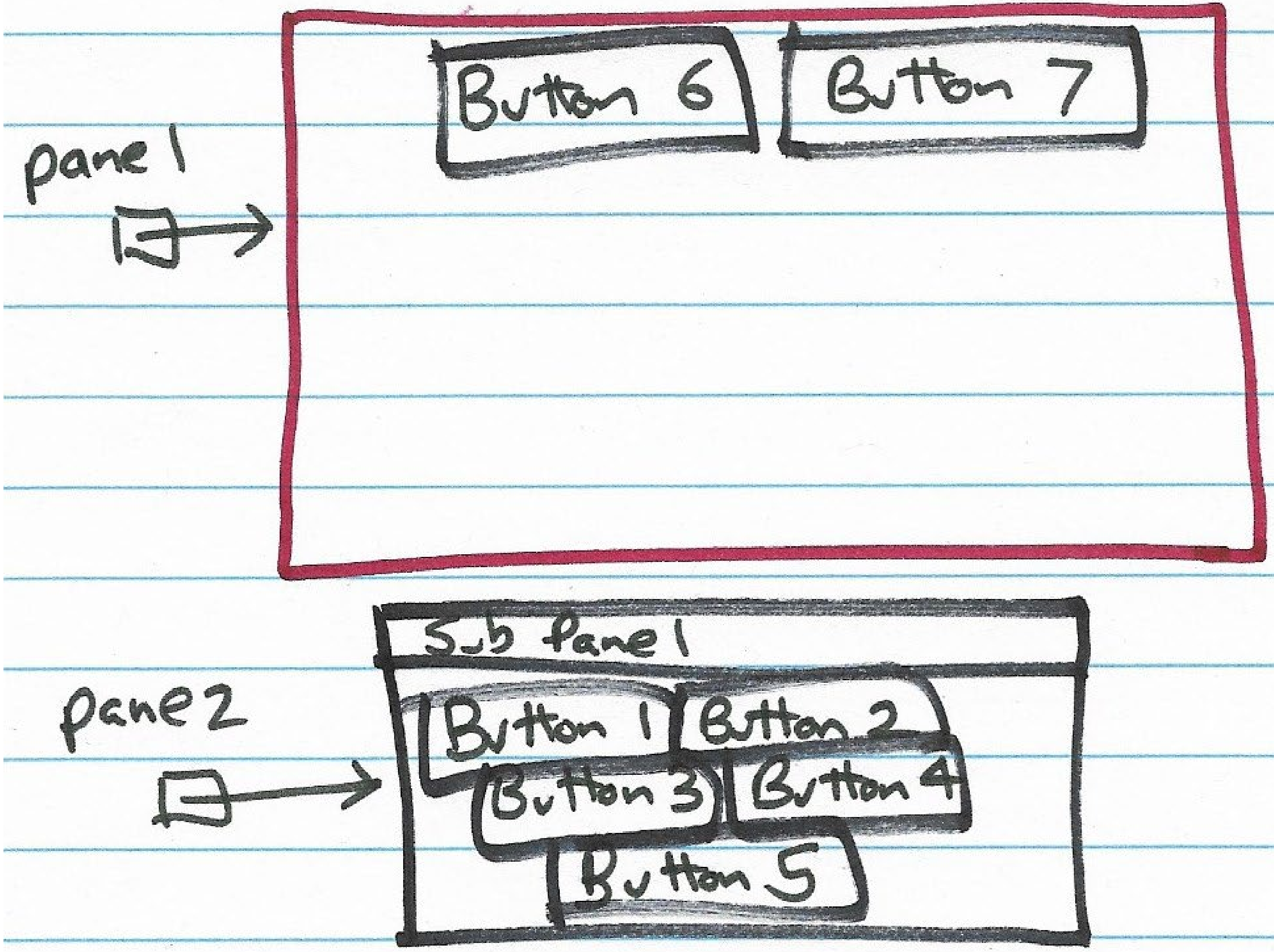
```
 public void createGUI() {
        JFrame frame = new JFrame("SubPanel Example");
        frame.setDefaultCloseOperation
   (JFrame.DISPOSE_ON_CLOSE);

        JPanel pane1 = new JPanel();
        JPanel pane2 = new JPanel();

// continued ...
```

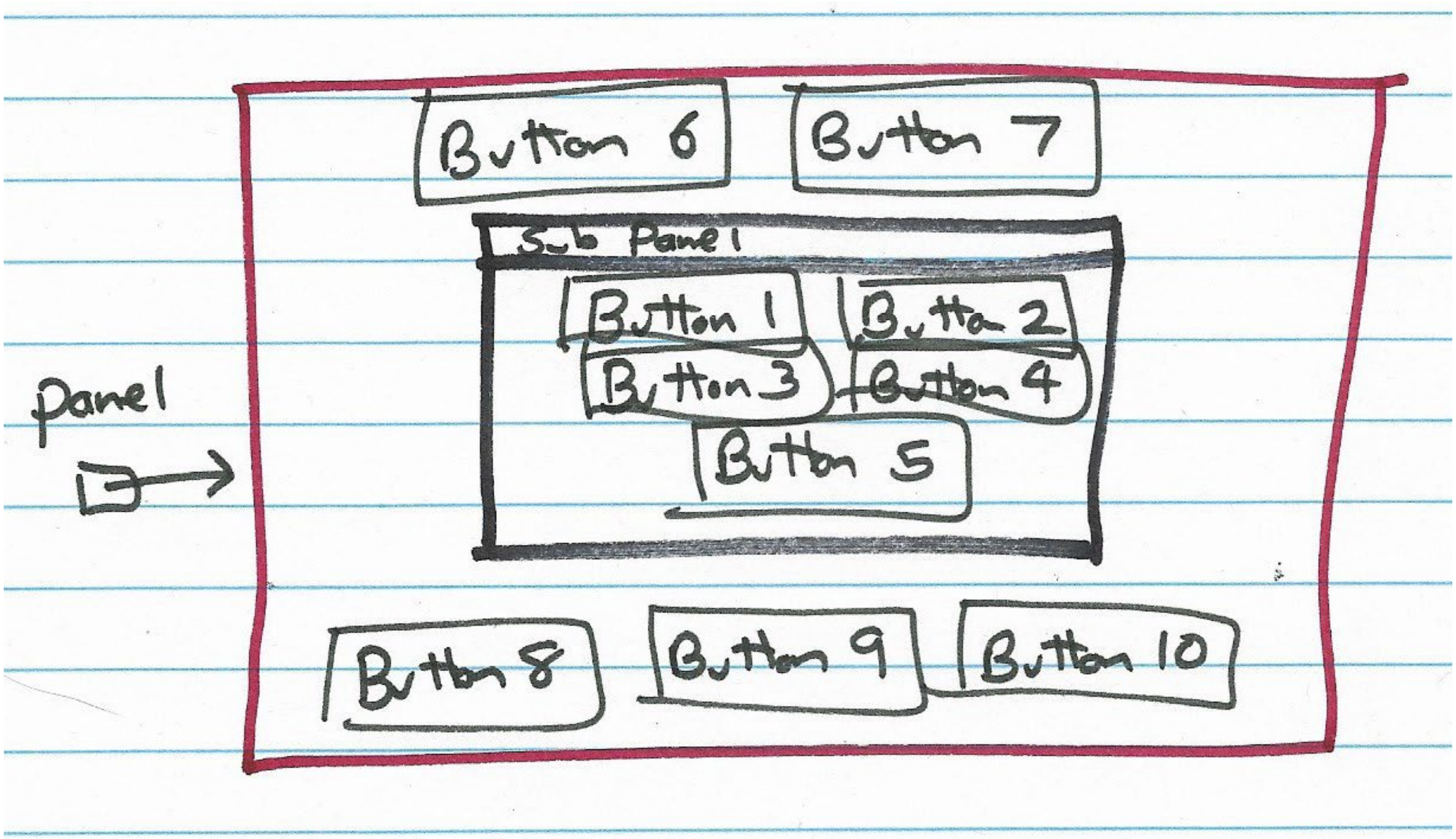
Example: Using Sub-Panels (3)

```
// ... continued

        Border b1 = BorderFactory.createLineBorder(Color.RED);
        Border b2 = BorderFactory.createTitledBorder("Sub Panel");

        pane1.setBorder(b1);
        pane2.setBorder(b2);

        addButtons(pane2, 5);
        addButtons(pane1, 2);
        pane1.add(pane2);
        addButtons(pane1, 3);

        frame.add(pane1);
        frame.setVisible(true);
    }

// continued ...
```

Example: Using Sub-Panels (4)

```
// ... continued

    static int counter = 0;

    static void addButtons(JPanel pane, int count) {
        for (int i = 1; i <= count; i++)
            pane.add(new JButton("Button " + ++counter));
    }
}
```


pane1
Button 6
Button 7
Sub Panel
pane2
Button 1
Button 2
Button 3
Button 4
Button 5


Button 6
Button 7
Sub Panel
Button 1
Button 2
Button 3
Button 4
Button 5
panel
Button 8
Button 9
Button 10

Video 4
Canvas and Graphics Classes

Canvas Class

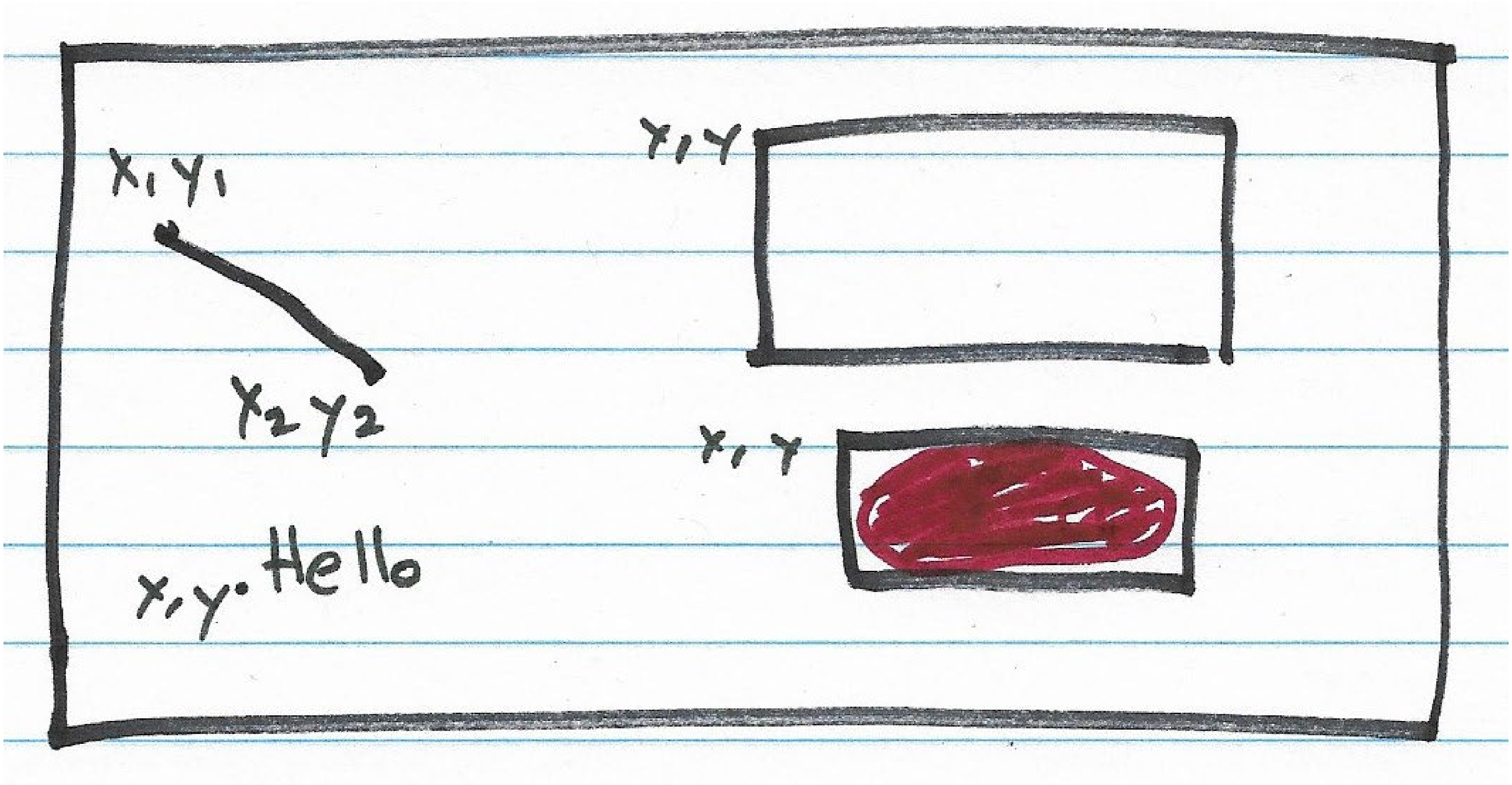
- A blank rectangular area that can be added to a Component (e.g., a Panel)
- Permits drawing operations by subclassing and overriding “void paint(Graphics g)” method

Graphics Class

- “Knows” how to draw on a given Canvas
- Coordinates in pixels (for our purposes)
  - Upper left is 0,0
  - x moves to right; y moves down
- Graphics context includes
  - Current color
  - Current font (when drawing text)
  - Other properties

Graphics Class Operations

- Call from within paint(), running on EDT
- Examples…
  - g.drawLine(x1, y1, x2, y2)
  - g.drawRect(x, y, width, height)
  - g.fillOval(x, y, width, height)
  - g.drawString(s, x, y)
- When model changes, call repaint(), allowable from non-EDT, which calls paint() later


x1 y1
x2 y2
x, y
x, y
x, y. Hello

Example: View (1)

```
import javax.swing.SwingUtilities;
import javax.swing.JFrame;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Graphics;

public class View extends Canvas {
    Model model;

    View(Model model) {
        this.model = model;
        SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            createGUI();
         }
      });
    }
```

Example: View (2)

```
 public void createGUI() {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation
   (JFrame.DISPOSE_ON_CLOSE);
        frame.setSize(640, 480);
        frame.add(this);
        repaint();
        frame.setVisible(true);
    }
// continued...
```

Example: View (3)

```
// ... continued

    /**
     * The paint method is called on the EDT in response to a call to
     * repaint().
     */
    public void paint(Graphics g) {
        int x = model.getX();
        int y = model.getY();
        int width = model.getWidth();
        int height = model.getHeight();

        g.setColor(Color.RED);
        g.fillOval(x, y, width, height);
    }
}
```