GUIs Part Three

Layout Managers

Mouse Events

Timer Events

Dialog Boxes

LAYOUT MANAGERS

Layout Managers

- Allow you to arrange components within a container by determining the size and position of the components
- Each container has a default layout manager, but you can change it

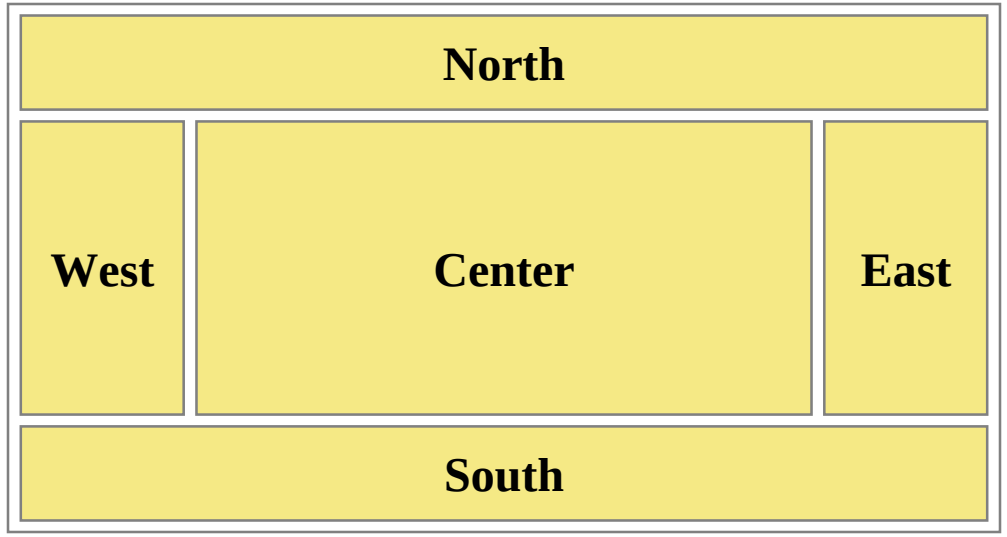
```
panel.setLayout(new WhateverLayout());
```

FlowLayout

- The default manger for JPanel
- Adds to a row until there is no more room, then goes on to the next row
- Components are displayed in the order they are added to the container
- This layout manages the placement for you- you don’t really get to choose

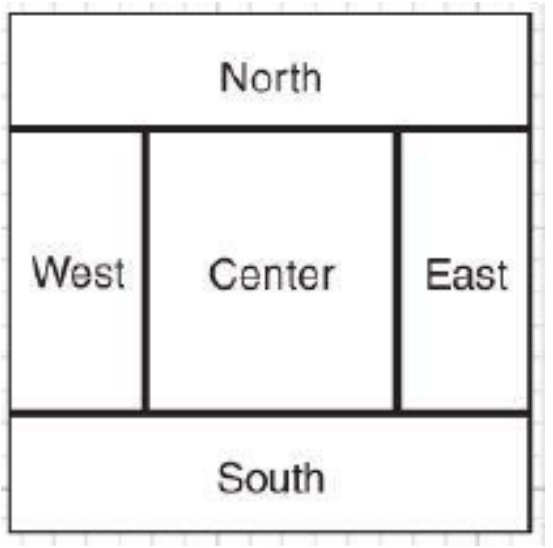
Border Layout

- Border layout defines five areas into which components can be added



Border Layout

- The default manager for JFrame
- You choose the position of each component
  - BorderLayout.NORTH, SOUTH, WEST, EAST, CENTER
- Edges laid out first and all remaining space is occupied by the center
- When a container is resized, the center changes its size
- Only **one** component per area
  - So usually, we put in a panel



Border Layout

- You add components with the `BorderLayout` constant that describes the location:

```
panel.add (button1, BorderLayout.CENTER)
```

- Leaving out a constant assumes a constant of `CENTER`

Grid Layout

- Arranges components in rows and columns
- All components are given the same size
- The size of each cell is determined by the overall size of the container
- As components are added, they fill the grid from left-to-right and from top-to-bottom (by default)
- You set this manager by specifying the number of rows and columns:

```
panel.setLayout(new GridLayout (rows, cols));
```

Nesting Panels/Layouts

- It’s very common to use lots of nested panels and multiple layouts to get the look you want.
- Use a panel to group together a set of components. Then add that panel to another panel. And so on.

Practice

- Review the LayoutDemo program.
- Modify the FormInput program to use nested panels for better display.

Multiple Classes

- Often, we'll have one "drawing panel" and one "control panel."
- We send a reference to our drawing panel object to the control panel constructor.
- Then the control panel can make changes to the drawing panel by invoking methods on the drawing panel object.
- We add *both* panels to our frame.

Practice

- Modify the random circle drawing program so the buttons are not part of the drawing space, but underneath in their own panel.

MOUSE EVENTS

Event Source

- The movement of the mouse!
- No visible component

Event Object

- A MouseEvent object is automatically generated whenever there is action with the mouse.
  - When the mouse is clicked, moved, dragged, etc.
- MouseEvent object methods:
  - getPoint()
  - getClickCount() // use if you want to know if double-clicked, for example

Mouse Listeners

- MouseListener

public void mousePressed(MouseEvent event)	**the mouse button is pressed down**
public void mouseReleased(MouseEvent event)	**the mouse button is released**
public void mouseClicked(MouseEvent event)	**the mouse button is pressed down and released without moving the mouse in between**
public void mouseEntered(MouseEvent event)	**the mouse pointer is moved onto (over) a component**
public void mouseExited(MouseEvent event)	**the mouse pointer is moved off of a component**

- MouseMotionListener

public void mouseMoved(MouseEvent event)	**the mouse is moved**
public void mouseDragged(MouseEvent event)	**the mouse is moved while the mouse button is pressed down**

Registering The Listener

- Add the listener(s) to your JPanel

```
this.addMouseListener(new MyMouseListener());
this.addMouseMotionListener(new MyMouseListener());
```

- Important Note: You must add the **correct** listener!

Adapter Classes

- With the two listener classes, we have to include **all** of the methods. Even though we usually only want one or two.
- Instead, we can use an adapter class.

```
public class MyMouseClass extends MouseAdapter {
    @Override
    public void mouseClicked(MouseEvent event) {
    }
    @Override
    public void mouseMoved(MouseEvent event) {
    }

}
```

Practice

- Write a GUI that counts the number of clicks inside of a panel. Print the number of clicks and the integer coordinates of each click.
- Write a “rubber line” GUI.
- Write a program to draw a “tail” on the cursor as it moves.

TIMER EVENTS

The `Timer` Class

- The `Timer` class is a GUI component that has no visual representation
- A `Timer` object generates an `ActionEvent` at specified intervals
  - You can listen for that event and take action when it happens
- Timers can be used to manage events that are based on a timed interval, like animation

The Timer Class (cont.)

- The Timer class has several useful methods:
  - start
  - stop
  - setDelay
- Example, to create events that happen every five seconds:

```
Timer timer = new Timer(5000, new TimerListener());
timer.start();
…
public class TimerListener implements ActionListener {
  public void actionPerformed(ActionEvent event) {
      // the code you want to happen every 5 seconds
    }
}
```

Practice

- Write a GUI that counts down from 60 seconds repeatedly.
  - Add pause and resume buttons.
  - Change the color of the label every five seconds. Use a second timer!
- Write a rebound animation.

DIALOG BOXES

Dialog Boxes

- A dialog box is a pop-up window.
- Dialog boxes:
  - convey information
  - confirm an action
  - allow the user to enter data
  - allow the user to choose a file
- Dialog boxes have a single, specific purpose
  - User interactions with dialog boxes are usually brief

Dialog Boxes

- The `JOptionPane` class provides static methods to create three types of dialog boxes.
- Message
  - displays a message to the user

```
JOptionPane.showMessageDialog(
        Component parent, Object message)
```

- Input
  - returns a String with the user's input

```
JOptionPane.showInputDialog(Component parent,
 Object message)
```

Dialog Boxes (continued)

- Confirmation
  - shows a message with Yes/No/Cancel options

```
JOptionPane.showConfirmDialog(
        Component parent, Object message)
```

  - returns an int

```
JOptionPane.YES_OPTION
JOptionPane.NO_OPTION
```

Practice

- Write a program that uses dialog boxes to obtain two integer values. Use a message-style dialog box to display the sum and product of the two values. Use a confirm-style dialog box to see whether the user wants to process another pair of values.
- Modify the drawing circle program to allow the user to input the number of circles to draw.

Troubleshooting Checks (List in Progress)

- Did you add the component (e.g., button) to the container (e.g., panel)?
- Did you add the panel to the frame?
- Did you register a listener to **each** button or panel?
  - Is it the *right* listener (mouse vs. mouse motion)?
- Did you add **each** radio button to a ButtonGroup **and** a panel?
- Did you accidentally re-declare a component inside the constructor or inside of a method in the listener class? (Almost always, you want to invoke a method on the existing variable, not re-declare the variable.)
- Did you create the correct frame object in main?
- Did you start the timer?
- Drawing:
  - Did you invoke super.paintComponent(pen)?
  - Did you invoke repaint() when you want to update the display?

Troubleshooting Checks- The Final List!

- Did you add the component (e.g., button) to the container (e.g., panel)?
- Did you add the panel to the frame?
- Did you register a listener to **each** button or panel?
  - Is it the *right* listener (mouse vs. mouse motion)?
- Did you add **each** radio button to a ButtonGroup **and** a panel?
- Did you accidentally re-declare a component inside the constructor or inside of a method in the listener class? (Almost always, you want to invoke a method on the existing variable, not re-declare the variable.)
- Did you create the correct frame object in main?
- Did you start the timer?
- Drawing:
  - Did you invoke super.paintComponent(pen)?
  - Did you invoke repaint() when you want to update the display?