Objectives
- Assign 8: Refactoring for Extensibility
- Programming Paradigms
- Introduction to GUIs in Java
  - Event handling

Assign 8: Understanding Code
- Focus: how open is the code to adding new kinds of bets and how closed it is to modification?
  - How many classes know about the Bet class?
  - What code would need to be added to Game to allow the user to make another kind of bet that paid one to one odds and was based on whether the number spun was high (between 19 and 36) or low (between 1 and 18)?

Assign 8: Bug
- Determining if Odd/Even Bet was won is incorrect
  - What is the problem?

Assignment 8 Questions?

PROGRAMMING PARADIGMS
Programming Paradigms

- Our focus has been Object-oriented and Procedural paradigms
- Other paradigms
  - Event-driven
    - GUIs, Web applications
  - Distributed
    - Web applications, Grid, Cloud
  - Concurrent
  - Parallel
  - Aspect-oriented

GUIs in Java

Swing & AWT

- Swing does not completely replace AWT
- Using the Swing graphics programming model
  - Improves performance
  - Allows more efficient development of GUIs
- We will use Swing mostly
  - Leverage AWT

Swing: Made up of Components

- Top-level components
  - Hold GUI elements
  - Examples: JFrame, JWindow, JDialog, JApplet

- GUI Elements
  - Things user interacts with
  - Examples: JButton, JLabel, JMenuBar

Java GUI Libraries: AWT & Swing

- AWT: Abstract Windowing Toolkit
  - Original GUI toolkit
  - Relies on operating system to render GUIs
    - Benefit: Match look and feel of platform
  - Classes in java.awt.*

- Swing: added to Java2
  - Classes in java.awt.swing.*
  - Extends AWT
  - Provides Java look and feel for applications
    - But can plug in other look & feels
Frames
- Frame: Most basic unit of graphics programming
- Example of a container
  - A container contains other UI components
- A top-level window
  - Not contained within another window
- Swing’s JFrame class implements a frame

Example Frame
```java
public class Game extends JFrame implements KeyListener {
    public static void main(String[] args) {
        Game session = new Game();
        session.init();
    }
    public void init() {
        // Top-left corner is (0,0)
        // width/height: XBOUND, YBOUND
        setBounds(0, 0, XBOUND, YBOUND);
        // Shows the window
        setVisible(true);
    }
}
```

Frame Inheritance
- Class hierarchy
  ```java
  java.lang.Object
  |-- java.awt.Component
  |   |-- java.awt.Container
  |       |-- java.awt.Window
  |       |-- java.awt.Frame
  |       |-- javax.swing.JFrame
  ```
- JFrame is derived from java.awt.Frame
  - Frame class is derived from Container class
    - Container: anything that can contain UI components

Components & Containers
- Component
  - Abstract class
  - Everything you see is a component
    - All nonmenu-related AWT components
  - Many methods
    - Some deprecated: be careful
- Container
  - Concrete implementation of Component
  - Base class of many classes

Container Methods
- add(Component c)
- setSize
  - Sets size of frame in pixels
- setLocation
  - Sets location of frame
    - Coordinates of top-left corner
- setBounds
  - Sets both size and location of frame
    - Provides information needed for setSize and setLocation

Window Methods
- Top-level window
- No borders
- No Menu Bar
- dispose()
  - Closes window and reclaims resources associated with it
- toBack()
  - Sends window to back, may lose focus/activation
- toFront()
  - Bring to front, make this the focused window
Frame’s Methods

- Top-level window with title and borders
- setTitle(String title)
  - Sets title of frame (displayed in title bar)
- setResizable(boolean resizable)
  - Can the user resize the frame?

Anatomy of an Application GUI

GUI

Internal structure

JFrame
- ContainsContentPane
  - A Container object that holds components you add, placing them in the frame
  - The part of the frame that holds UI components

Implementing a GUI Component

1. Create it
2. Configure it
3. Add children (if container)
4. Add to parent (if not JFrame)
5. Listen to it

(order important)

Implementing a GUI Component

1. Create it
   - JButton b = new JButton();
2. Configure it
   - b.setText(“press me”);
   - b.setForeground(Color.blue);
3. Add it to parent
   - panel.add(b);
4. Listen to it
   - Events: Listeners

JFrame

Building a GUI

1. Create (top down):
   - Frame
   - Container
   - Components
   - Listeners
2. Add (bottom up):
   - Listeners into components
   - Components into panel
   - Panel into frame
Example Code

```java
// create the components
JFrame f = new JFrame("title");
f.setBounds(0, 0, 100, 100);
Container pane = f.getContentPane();
JButton b = new JButton("press me");

// add button to panel
pane.add(b);
// show the frame
f.setVisible(true);
```

More GUI Components

- **Choice**
  - Drop-down list
- **FileDialog**
  - Opening and saving files
- **List**
  - Scrolling
  - Allows multiple selections
- **ScrollPane**
  - Scrollbars
- **TextField**
  - Single line of text
- **TextArea**
  - Multiple lines of text

Menus

- **MenuBar**
  - Thing across top of frame
  - Frame: `setMenuBar(MenuBar mb)`
- **Menu**
  - The dropdown part
  - A sequence of `MenuItem`
  - `Menu` is a subclass of `MenuItem`, so can have submenus

Practice: Combining Components

- Create a panel with three buttons on it

Placement of Components

- How does the panel know where to place a button?
- How does the panel know where to place the next button?
- How does the panel know where to place any component that is added to it?
Layout Managers

- Java uses *layout managers* to place components inside a container
- `LayoutManager` automatically handles placement of components
  - When a component is added to a container (through `add`), layout manager decides where to place the component

Default Layout Managers

- JFrame’s content pane: `BorderLayout`
- JPanel’s: `FlowLayout`

The Flow Layout Manager

- Default layout manager for a `panel`
- Lines components up *horizontally* until no more room in container
  - Then starts a new row of components
- If user resizes component, layout manager automatically reflows components

The Flow Layout Manager

- Can choose how to arrange components in a row
  - Default: center each row
  - Other options: left or right align
- Change alignment using `setLayout`

```java
setLayout(new FlowLayout(FlowLayout.LEFT));
```
- `setflow` for gaps to put around components

Border Layout Manager

- Default layout manager of the content pane for JFrame
- Lets you choose where you want to place each component

```
North
West       Center      East
South
```

Border Layout Regions

- Edge components are laid out first
- Center occupies remaining space
Border Layout Rules

• Grows all components to fill available space
• If container is resized, edge components are redrawn and center region size recomputed
• To add a component to a container using a border layout
  ➢ Ex: JFrame’s content pane

```java
Container contentPane = getContentPane();
contentPane.add(button, BorderLayout.SOUTH);
```

Adding Components Using a Border Layout

• If no region specified, assumes center region

What happens if we add multiple components, e.g., three buttons, without specifying a region?

➢ Recall: border layout grows component to fit specified region

A Border Layout Limitation

• Last button added grows to completely fill center region
• First two buttons were discarded/overwritten by each subsequently added component

Changing Layout Managers

• Any container can use any layout manager
• Use `setLayout` to change layout manager before adding components

```java
// sets layout to a new flow layout manager that
// aligns row components to the left and uses a 20 pixel
// horizontal separation and 20 pixel vertical separation
setLayout(new FlowLayout(FlowLayout.LEFT, 20, 20));
```

```java
// sets layout to a new border layout manager that
// uses a 45 pixel horizontal separation between
// components (regions) and a 20 pixel vertical separation
setLayout(new BorderLayout(45, 20));
```

Combining Panels

• Panels act as (smaller) containers for UI elements
• Can be arranged inside a larger panel by a layout manager
• Use additional panels to customize look
  ➢ Create a panel
  ➢ Add some buttons to it
  ➢ Add that panel to a region in content pane

Combining Panels

```java
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

JPanel panel = new JPanel();
panel.add(JButton button);
panel.add(JButton button);
JScrollPane scrollPane = new JScrollPane(panel);
scrollPane.setViewportView(new JTextArea());
```
Combining Panels

Using Additional Panels

- Get fairly accurate and precise placement of components
- Use nested panels with

<table>
<thead>
<tr>
<th>Layout</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>BorderLayout</td>
<td>Content panes and enclosing panels</td>
</tr>
<tr>
<td>Flow Layouts</td>
<td>Panels containing buttons and other UI components</td>
</tr>
</tbody>
</table>

FlexibleLayout.java

Another Layout Manager: Grid

- Divides container into columns and rows of equal size, which collectively occupy the entire container region
- Rows and columns are aligned like a table
  - When container is resized, the “cells” grow and/or shrink
  - Cells always maintain identical sizes
- Example:
  ```java
  panel.setLayout(new GridLayout(5, 4)); // 5 rows, 4 cols
  ```

Adding Components to a Grid Layout

- Components added *sequentially*
  - 1st *add* adds component to 1st row, 1st col
  - 2nd *add* adds component to 1st row, 2nd col
- And so forth until 1st row is filled
- Then 2nd row begins with the 1st column
- Continues until the entire container is filled

Grid Layout Rules

- Components are resized to take up entire cell
- Restrictive but can be useful for some applications
- Example: Create a row of buttons of identical size
  1. Make a panel that has a grid layout with one row
  2. Add a button to each cell
  3. Set horiz/vert separation so buttons are not touching

Layout Manager Heuristics

null
- none, programmer sets x,y,w,h

FlowLayout
- Left to right, Top to bottom

BorderLayout
- w: c: e: s

CardLayout
- One at a time

GridBagLayout
- JButton
Looking Ahead

• Assignment 8 – due Wednesday
• Exam - Friday