Intro to Java Swing Graphics

Today, we will learn how to create graphical user interfaces (GUIs) in Java using the Swing library. This is the same library that JetBrains uses (with heavy customization) to create the GUI for IntelliJ IDEA!

Content Learning Targets

After completing this activity, you should be able to say:

- I can create a small graphical user interface (GUI) using Java Swing.
- I can draw simple objects and shapes to the screen in Java Swing.
- I can read and interpret the Java API documentation for Swing and AWT classes.
- I can explain the model-view-controller (MVC) pattern for user interfaces.
- I can apply MVC to implement small, non-interactive GUIs.

Process Skill Goals

During the activity, you should make progress toward:

- Reading Java API documentation and making inferences. (Information Processing)

Facilitation Notes

First hour:

Model 1 introduces Java Swing, including a simple GUI program that draws shapes. Model 2 gives teams opportunities to practice drawing new things in Swing.

Second hour: Model 3 introduces the Model-View-Controller (MVC) design pattern, starting with a live-coding demo of a Cat viewer program. It ends with open-ended exploration time, including the ***HourTimerComponent*** project, translation/rotation, and more.

Model 1 Intro to Java Swing

Questions (20 min) **Start time:**

1. Inspect and run the file ***src/EmptyFrameViewer.java***.

a) What built-in Java class does `EmptyFrameViewer` use that represents a window?

JFrame

b) Where does the text passed to the `setTitle` method appear?

along the top of the window

c) Adjust the values passed to `setLocation`. What do these values represent?

the x and y coordinates of the top-left corner of the window

d) Comment out the call to `setVisible`. What happens when you run the program?

no window appears and it terminates immediately

e) Look at the Java API documentation for `JFrame`. What other options are available for `setDefaultCloseOperation`?

`DO_NOTHING_ON_CLOSE`, `HIDE_ON_CLOSE`, `DISPOSE_ON_CLOSE`

2. Open ***src/MyViewer.java*** and complete the TODOs to create a window that is 400 pixels wide and 600 pixels tall, at location (350, 150) on the screen, with the title "I've been framed!".

3. Open ***src/MyComponent.java***. MyComponent extends `JComponent`, which is a built-in Java class that represents a blank area of the screen that we can draw on.

a) What method must we override to draw on a `JComponent`? paintComponent

b) What parameter does this method take? a `Graphics` object

c) In `drawSimpleSquare()`, where does `Rectangle` come from? `java.awt`

d) Change the last line from `g2.draw(square)` to `g2.fill(square)`. What changes?

`fill` fills in the shape, whereas `draw` only draws the outline

e) Examine the `drawUsingColors` method. Experiment with changing the color used to fill the rectangle (search online for RGB/RGBa color pickers).

4. Fill in the blanks using words from the provided bank (not all will be needed):

Word bank: canvas, lightbulb, paintbrush, pencil, picture frame, screen, table, wall, window

`JFrame` is to a picture frame as `JComponent` is to a canvas. A `Graphics2D` object is like a paintbrush that can be used to draw on the `JComponent`.

5. Draw (paper, whiteboard, PlantUML, etc.) a UML class diagram showing the relationships between MyViewer, MyComponent, JFrame, and JComponent. (Normally, we will leave out built-in classes like JFrame and JComponent from our UML, but this time, include them.)

See ***UMLMyViewer.png*** in the same directory as MyViewer and MyComponent.

Model 2 Practice with Java Swing

Questions (25 min) Start time:

6. Split into pairs/trios, and complete the TODOs in ***src/MyComponent.java*** to explore other objects you can draw with the `Graphics2D` object. As needed, refer to the solution version of MyComponent in the ***sol*** directory (commented out). Talk to the other pair/trio on your team for help and ideas.

Model 3 The Model-View-Controller (MVC) Pattern

The Model-View-Controller (MVC) (MVC) pattern is a common design pattern for user interfaces. Java Swing uses this pattern, and it is commonly used in web applications as well. The idea is to separate the ***model*** (the data and logic) from the ***view*** (the graphical representation) and the ***controller*** (the code that handles user input and updates the model and view).
Follow along during the live coding ***Cat*** program demo. The starter code is in the ***src/liveCoding*** directory, and solution files are in ***sol/liveCoding***.
The remainder of the time is open-ended exploration. Refer to the ***slides*** directory for tips, including the handy crosshairs technique. You can work on the ***src/hourTimer*** project, the ***src/translateRotate*** project, go back to ***src/swingIntro***, or create a new package (right click `src` and select `New > Package`) and experiment with your own ideas. Some suggestions to explore:

- Repeat the MVC pattern from our Cat demo for a house, car, simple animal, or similar.
- Recreate a simple flag by drawing shapes on a component. Pick a country or state (or even Terre Haute!).
- You might notice that the edges of shapes are jagged. Research `Graphics2D.setRenderingHint` and "anti-aliasing", and try to make them smoother.