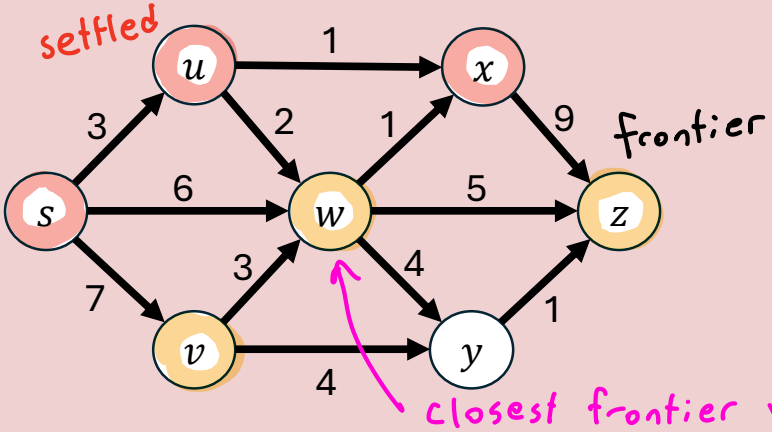
Poll Everywhere

PollEv.com/javabear text **javabear** to 22333

Suppose we pause Dijkstra's algorithm from s on the following graph when the discovered map has the state shown on the right.



discovered

vertex	s	u	v	w	x	z
dist	0	3	7	5	4	13
prev	null	s	s	u	u	x

Which vertex will be next one to be visited?

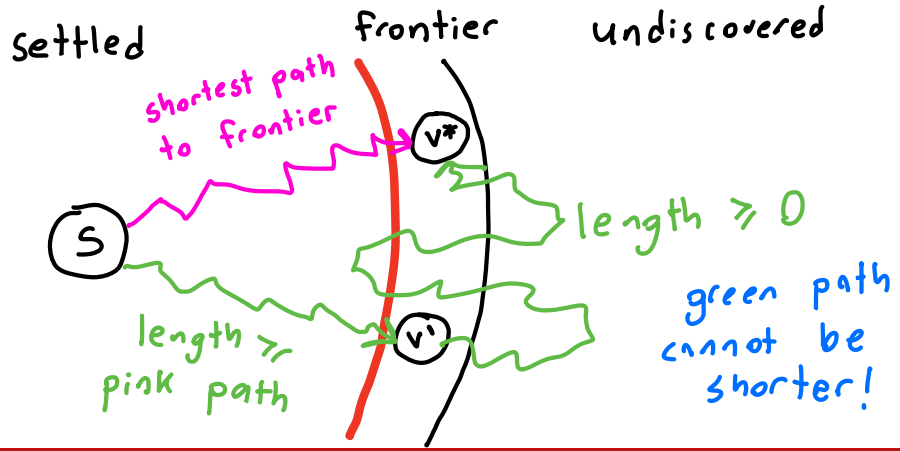
Dijkstra's Invariant

At the start of each main loop iteration:

1. We've located the shortest path to all *settled* vertices.
2. The shortest *known* path to the *closest* `frontier` vertex v^* is guaranteed to be the true shortest $s \rightsquigarrow v^*$ path.

Proof Sketch:

- consider any alternate path to v^*
- at some point, this path must cross boundary to some frontier vertex v'



Properties of Dijkstra's Algorithm

Vertices are visited/settled in increasing *distance* order (just like BFS).

- Shortest path to any unsettled vertex passes through frontier, so has distance ≥ any settled vertex.

The best *known* distance to a vertex v can only decrease as the algorithm runs (and we locate new paths to v)

For any discovered but unsettled vertex v,

length of shortest path to closest frontier vertex ≤ length of shortest s⇝v path ≤ length of shortest known s⇝v path

Dijkstra's Algorithm Runtime Analysis

```
while(!frontier.isEmpty()) {
  V v = frontier.remove();
  for (WeightedEdge<V> edge : v.outgoingEdges()) {
    V neighbor = edge.head();
    double dist = discovered.get(v).distance() + edge.weight();
    if (!discovered.containsKey(neighbor)) {
      discovered.put(neighbor, new PathInfo<>(dist, v));
      frontier.add(neighbor, dist);
    } else if (discovered.get(neighbor).distance() > dist) {
      discovered.put(neighbor, new PathInfo<>(dist, v));
      frontier.updatePriority(neighbor, dist);
}}}
```

O(|V|) iterations, one per discovered vertex

O(log|V|) · O(|V|)

O(|E|) total iterations (across all outer loop iterations)

O(1) · O(|E|)

O(1) · O(|E|)

O(1) · O(|V|)

O(log|V|) · O(|V|)

O(1) · O(|E|)

O(1) · O(|E|)

O(log|V|) · O(|E|)

Overall: O(|E|·log|V|)

* Since priority values only decrease, we can use a more clever heap variant called a Fibonacci heap to reduce runtime to O(|E| + |V|log|V|).

Poll Everywhere

PollEv.com/javabear text **javabear** to 22333

Which is the tightest bound on the space complexity of Dijkstra's algorithm?

dominated by size of discovered Map and frontier PQueue

$O(1)$	**(A)**
$O(\|V\|)$	**(B)**
$O(\|V\| + \|E\|)$	**(C)**
$O(\|V\| \log \|V\|)$	**(D)**



Lecture 24: Graphical User Interfaces

CS 2110

November 17, 2026

Today's Learning Outcomes

86. Categorize aspects of a GUI application as part of its *model*, *view*, or *controller*.

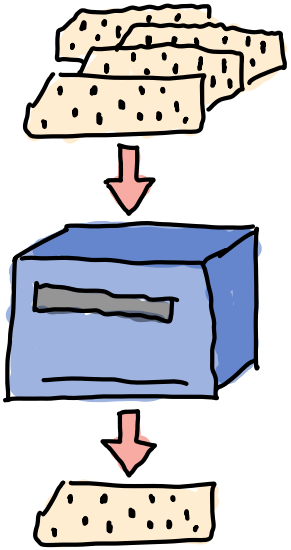
87. Write code to set up the component hierarchy of a graphical application given a mock-up of its design.

88. Describe the work performed by a layout manager and its interaction with elements in the component hierarchy.

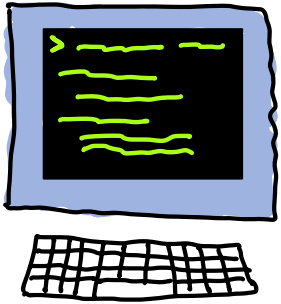
90. Write code that performs custom painting on graphical components.

Application Paradigms

Batch Applications



Command-Line Interfaces (CLI)



Enter Guess: 1122

○ ○ ● ●

A1

Enter Guess: 3311

● ○ ● ●

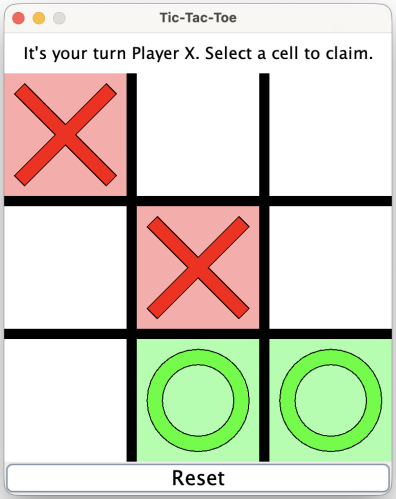
Graphical User Interfaces (GUI)



A6



GUI Applications



Brainstorm: What new considerations are there to make a graphical application like this?

- Create window and show on screen
- Add window title
- Get close, minimize buttons to work
- Add label to top of window "body"
- Draw with different shapes, colors
- Add button
- Make button do something when clicked
- Keep track of board state, turns
- Let players select cells, then redraw screen
- ⋮

MVC Separation

Since GUI applications are more involved, it helps to organize their code into three different aspects:

Model

- State representation of the "back end" of the application
- variables
- objects
- interfaces
- invariants

* basically, everything we've done so far

View

- The collection of windows and widgets that display the state of the program to the user
- widget = graphical component that presents user info or way to interact with program
  - button
  - slider
  - textbox

(Today)

Controller

- Application logic that responds to user inputs or other events
- Requests updates of the model or the view

(Next Lecture)

Poll Everywhere

PollEv.com/javabear text **javabear** to 22333



Our `TicTacToe` game should have a reset button that removes all symbols from the board when clicked.

Which aspect of the application would this functionality fall under?

This functionality needs all 3!

- Model **(A)**
- View **(B)**
- Controller **(C)**
- Uh .... **(D)**



Coding Demo: TicTacToe Model



GUI Frameworks

GUI applications are way more complicated than other programs we've written.

- Interactions with hardware (keyboard, mouse, etc.) and the operating system.
- Good views incorporate many interoperable and customizable widgets.
- Many layers of abstraction.
- High *coupling* between parts of the view and controller.

Frameworks are large software libraries that we use to build GUIs.

Webdev: React, Angular, etc. build on top of HTML/CSS/Javascript

Java: AWT → Swing → Java FX

Our choice, built into language, low barrier to entry



Coding Demo: The `JFrame` Class



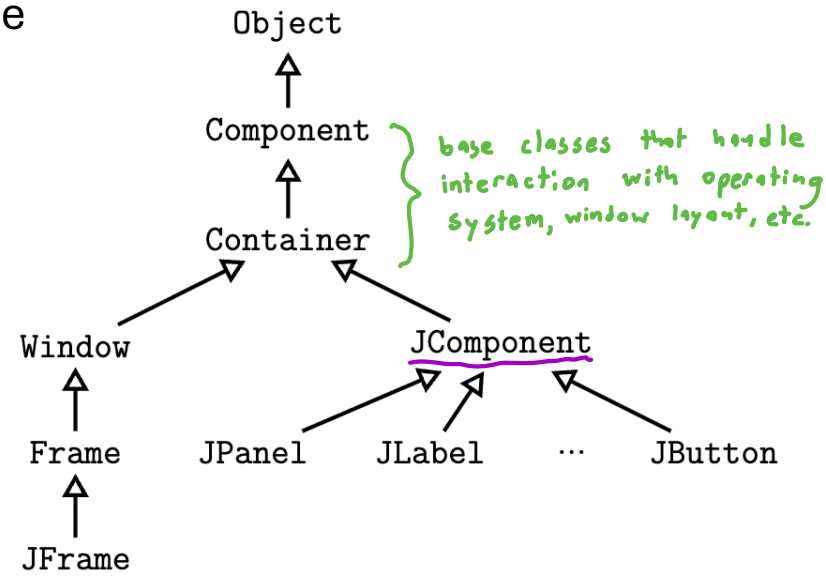
Swing Widgets

The JComponent class serves as the parent of all Swing widgets.

Three main ones:

- JPanel
  holds + arranges other widgets
- JLabel
  displays formatted text
- JButton
  does some action when clicked

many others (checkbox, slider, text pane, ...)



GUI frameworks are one of the best examples of the benefits of deep inheritance hierarchies

Poll Everywhere

PollEv.com/javabear text **javabear** to 22333

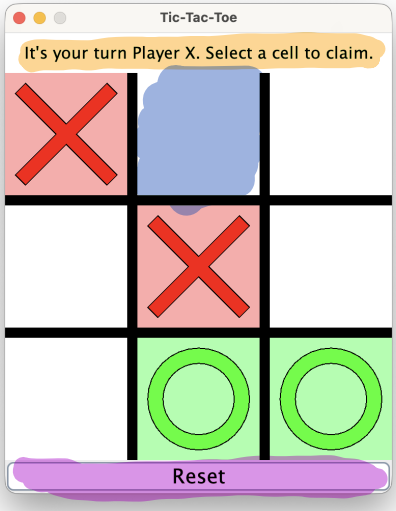


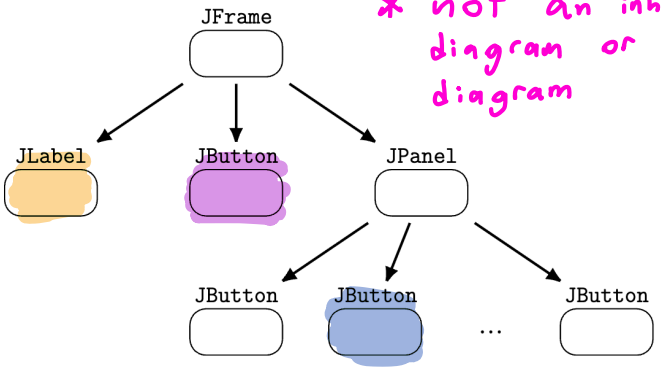
What is one consequence of the fact that all Swing widgets extend `JComponent`?

- Every widget must be drawn the same way — overriding — ✗ **(A)**
- It is impossible for third parties to make new Swing widgets — ✗ **(B)**
- Every widget will support a common set of core behaviors — ○ **(C)**
- Widgets must override every method of `JComponent` — ✗ **(D)**

The Component Hierarchy

= A tree that describes all the widgets in a GUI application window.
Parent-child relationships in this tree represent *containment*.





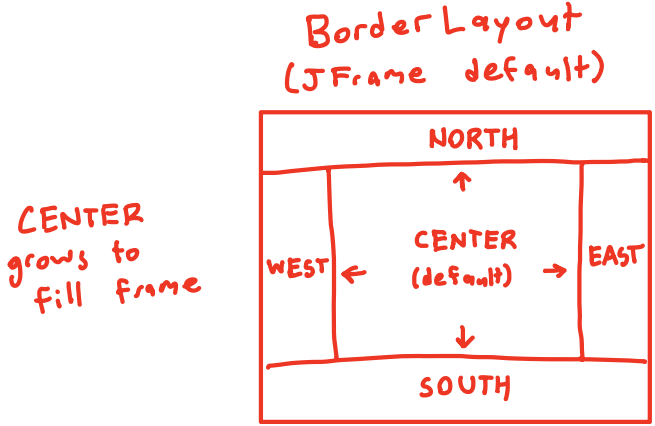
* not an inheritance diagram or a node diagram

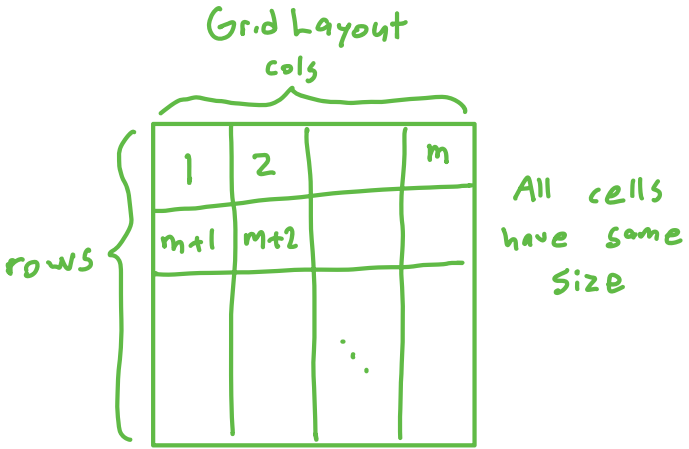
- The frame contains a label, a button, and a panel with 9 more buttons

Layout Managers

= Separate objects associated with a `Container` that determine how to arrange its child components.

We'll consider two:



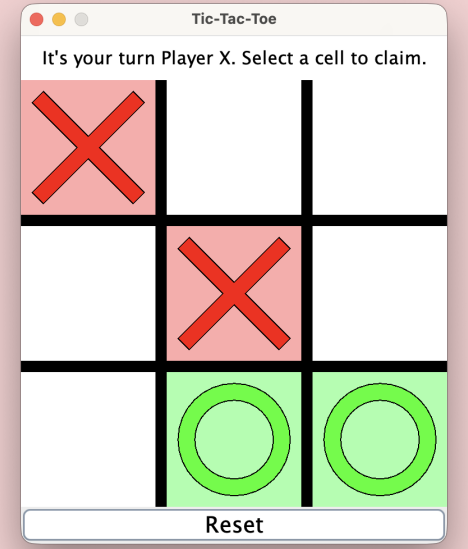


Call pack() on JFrame to ask layout manager to arrange components

Poll Everywhere

PollEv.com/javabear text **javabear** to 22333

Which are the most appropriate choices of layout managers in our Tic-Tac-Toe game?



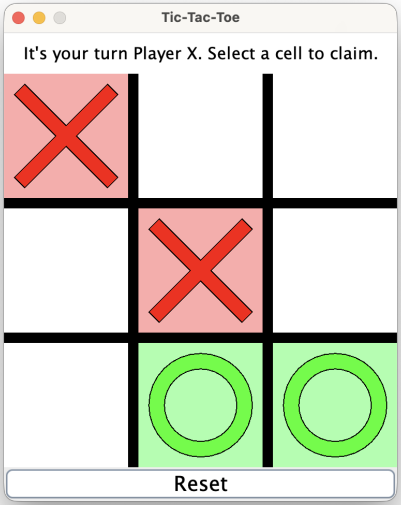
Frame: BorderLayout **Panel:** BorderLayout **(A)**

Frame: BorderLayout **Panel:** GridLayout **(B)**

Frame: GridLayout **Panel:** BorderLayout **(C)**

Frame: GridLayout **Panel:** GridLayout **(D)**

Coding Demo: Building out the View



Key Ideas:

- Custom `JComponent` subclasses
- Set `LayoutManagers` and use their constants to assign positioning in `add()`
- Call `pack()` at the end of the frame constructor

Connecting the View and Model

Multiple Approaches:

Maintain the model in a separate class (or classes), accessible to all view components through alias referencing.

- Model has many methods to query and update state
- Helps with separation of responsibilities in larger projects
- Having model updates reflect in view requires observer pattern → (next lecture)

Store relevant bits of the model within the relevant view classes.

- Sometimes leads to cleaner code when these are closely linked
- Can get unwieldy in larger projects, but often easier in small ones

* we'll demonstrate a bit of both

Custom Painting

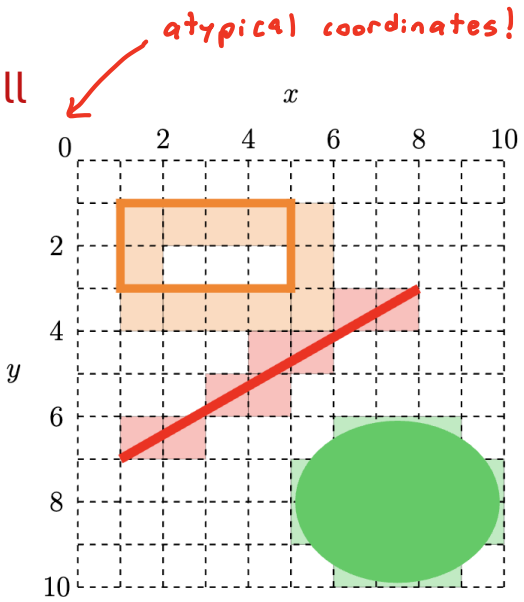
Override the `paintComponent()` method.

* First line must be `super.paintComponent()` call

`Graphics` object works like a paint brush:

- `setColor()` "dips" into one color of paint
- `draw____()` and `fill___()` methods add lines/shapes in that color

```
g.setColor(Color.RED);
g.drawLine(1,7,8,3);
g.setColor(Color.GREEN);
g.fillOval(5,6,5,4);
```



* Never call `paintComponent()` directly, instead call `repaint()`



Coding Demo: Painting Demo

