Poll Everywhere

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

What is the (tight) space complexity of Dijkstra's algorithm?

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

both pqueue and map grow to size O(|V|)

O(1)

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

Dijkstra's Algorithm Runtime Complexity

```
while(!frontier.isEmpty()) {                                   O(|V|) iterations
    V v = frontier.remove();                                   O(log|V|) · O(|V|)
    for (WeightedEdge<V> edge : v.outgoingEdges()) {           O(|E|) total iterations (across all outer loop iterations)
        V neighbor = edge.head();                              O(1) · O(|E|)
        double dist = discovered.get(v.label()).distance() + edge.weight();   O(1) · O(|E|)
        if (!discovered.containsKey(neighbor.label())) {       O(1) · O(|E|)
            discovered.put(neighbor.label(), new PathInfo(dist, v.label()));   O(1) · O(|V|)
            frontier.add(neighbor, dist);                      O(log|V|) · O(|V|)
        } else if (discovered.get(neighbor.label()).distance > dist)  {   O(1) · O(|E|)
            discovered.put(neighbor.label(), new PathInfo(dist, v.label()));   O(1) · O(|E|)
            frontier.updatePriority(neighbor, dist);           O(log|V|) · O(|E|)
}}}
```

O(|E| · log|V|)

Exam Reminders:

Prelim 2 is tonight, 7:30-9:00 !

Rooms assigned by first letter of your NetID:
Baker Lab 200 (a-p), 219 (q-s), 135 (t-z)

Bring your **Cornell ID Card** and a couple **writing utensils** (pencils, erasers, pens)
Exam is closed-book

More information and review materials linked on website / Ed

You've learned a lot so far! Time to show it off!

Lecture 24: Graphical User Interfaces

CS 2110

April 21, 2026

Today's Learning Outcomes

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

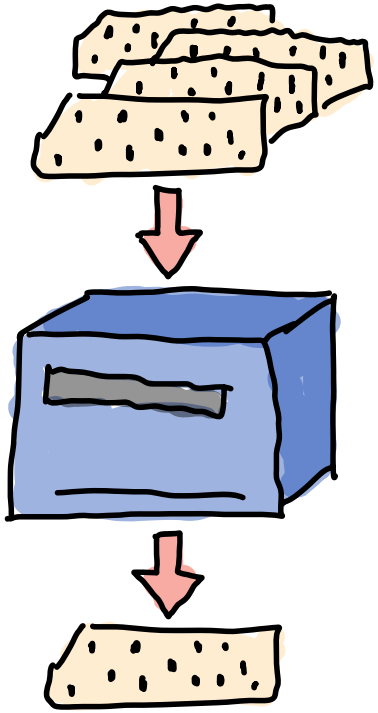
96. Write code to set up the component hierarchy of a graphical application given a mock-up of its design.

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

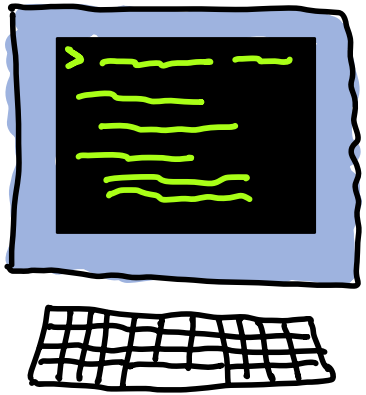
99. Write code that performs custom painting on graphical components.

Application Paradigms

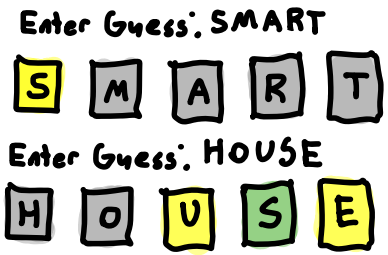
Batch Applications



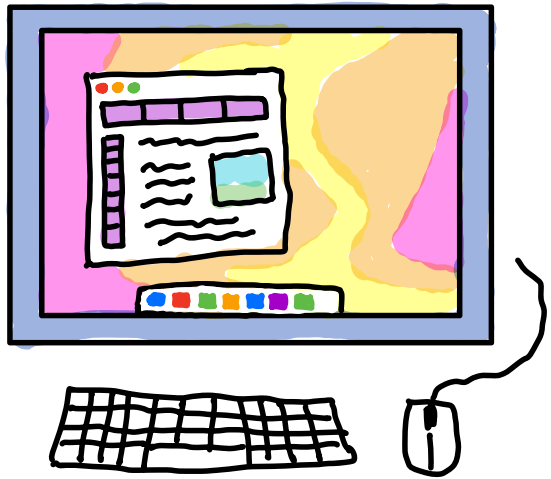
Command-Line Interfaces (CLI)



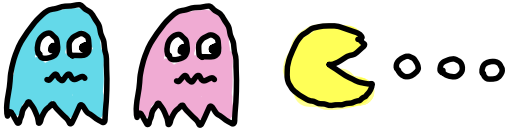
AI



Graphical User Interfaces (GUI)



AI

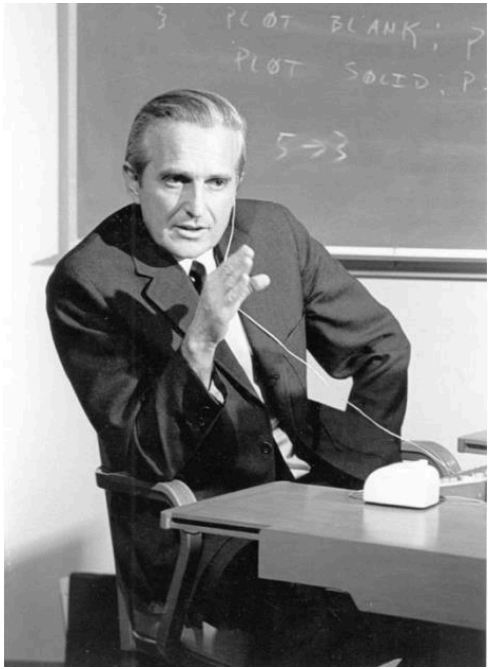


The *Mother of All Demos*: 1968

Foundation of "Personal Computing"

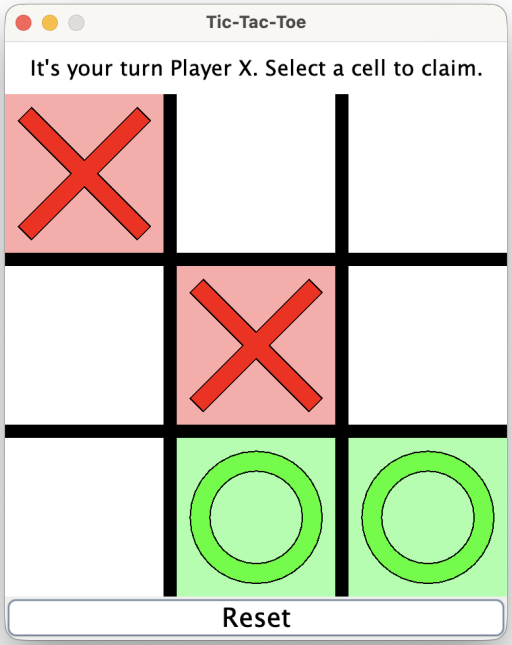
*"Computers can **augment** the capabilities of the human mind"*

- Computer mice
- Application windows
- Hyperlinks
- Word processors
- File menus
- Copy-paste
- Teleconferencing
- Document sharing



Douglas Engelbart

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

We can divide a GUI application into three 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
  - image
  - textbox

(Today)

Controller

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

(Next Lecture)

Coding Demo: TicTacToe Model

GUI Frameworks

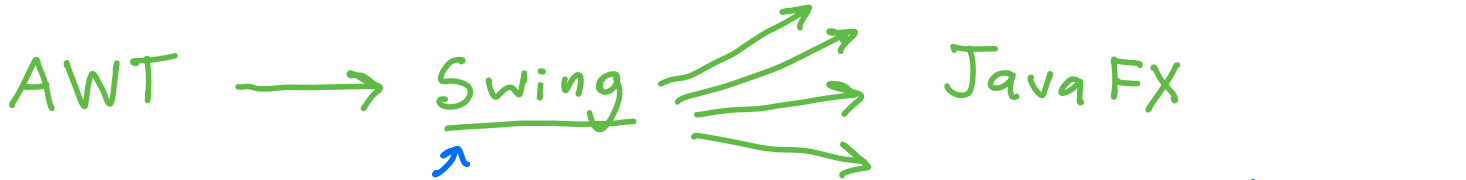
GUI applications are a lot more complicated than other programs we've written this semester

- Interaction with hardware and operating system
- Many layers of abstraction
- Good views need many interoperable + customizable widgets
- High coupling between parts of view + controller

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

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

Java:



Coding Demo: The `JFrame` Class

Swing Widgets

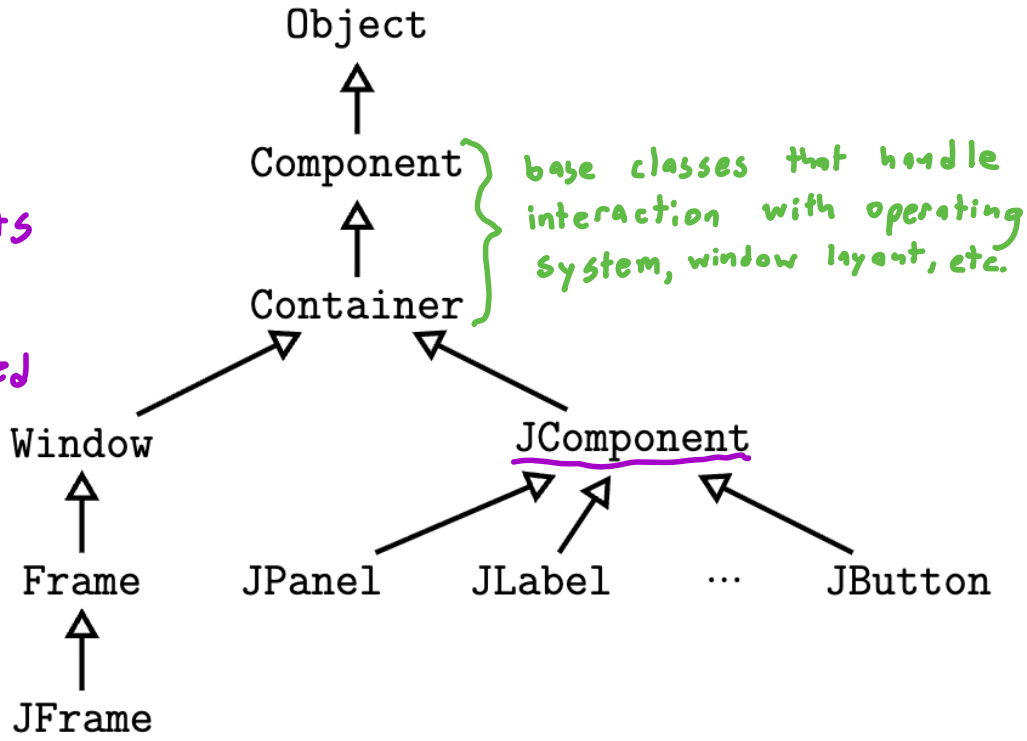
The JComponent class serves as the parent of all Swing widgets

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



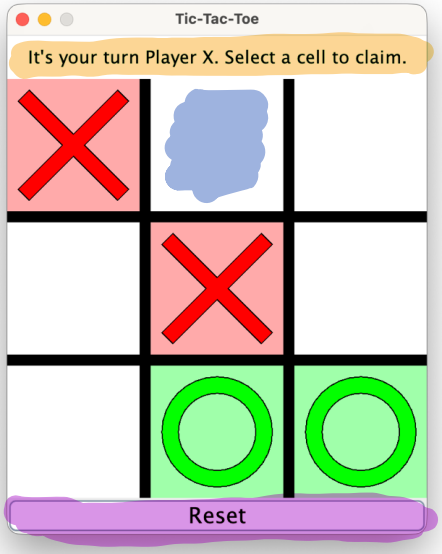
Poll Everywhere

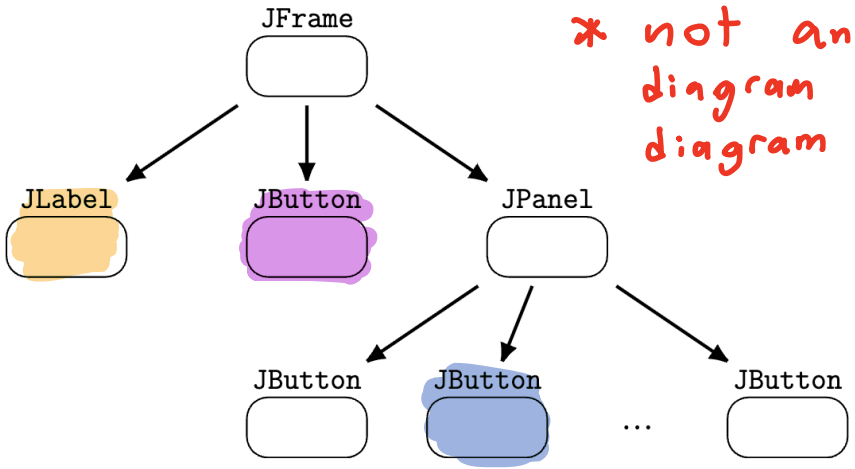
PollEv.com/javabear text **javabear** to **22333**

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





* not an inheritance diagram or a node diagram

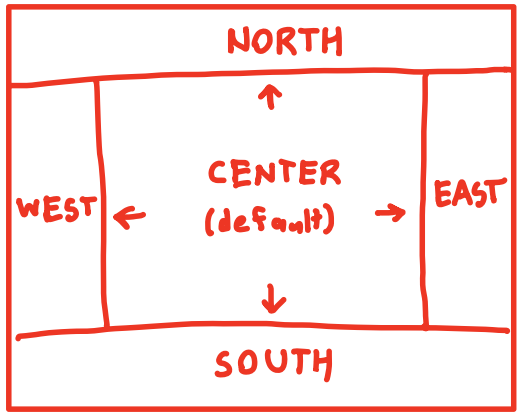
In this tree, children represent containment

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

Layout Managers

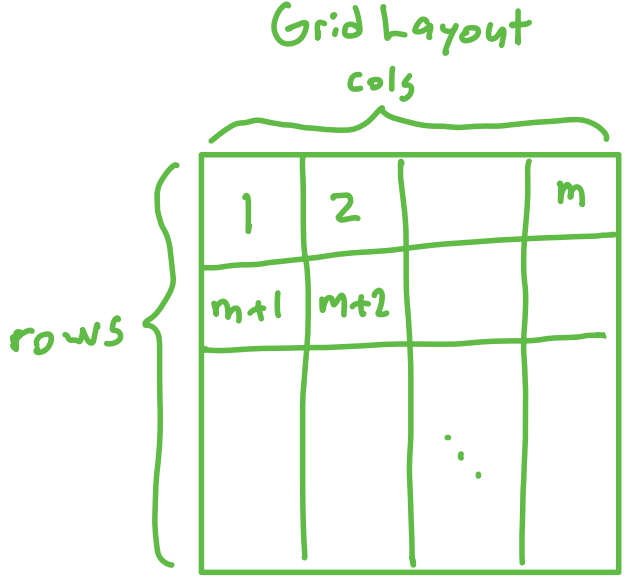
Separate objects associated with a Container that determine how to arrange its child components.
We'll consider two:

Border Layout
(JFrame default)



CENTER grows to fill frame

Grid Layout



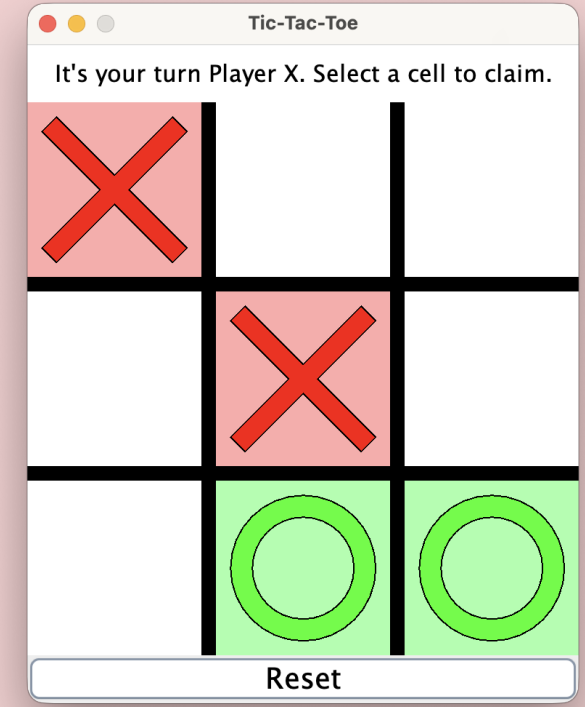
All cells have same size

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?



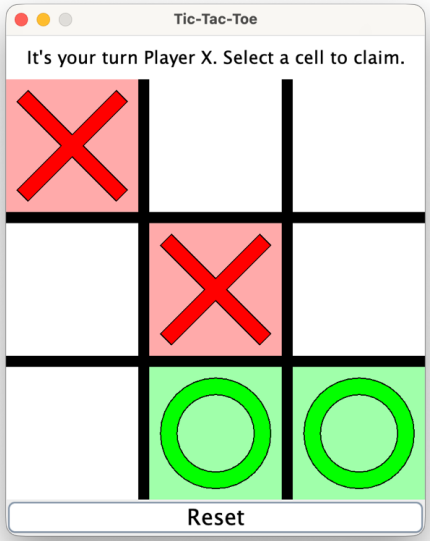
Frame: BorderLayout **Panel:** BorderLayout **(A)**

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

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

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

Coding Demo: Building out the View



- custom JComponent subclasses
- set LayoutManagers and use their contents to set positioning in add()
- call pack() at end of frame constructor

Connecting the View and Model

Multiple approaches

- Maintain the model in a separate class, accessible to all view components through alias references
  - 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 model within view classes
  - Sometimes leads to cleaner code when these are closely linked
  - Can get unwieldy in larger projects, but often easier in small ones

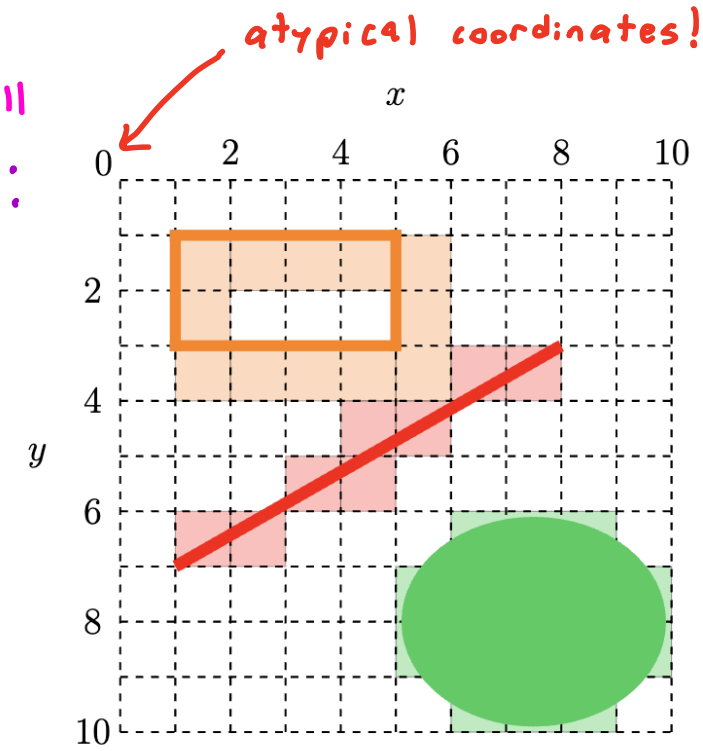
* we'll do a bit of both

Custom Painting

Override paintComponent() method
- first line must be super.paintComponent() call

Graphics object works like paint brush:
- setColor() "dips" into one color paint
- use draw___() and fill___() methods to 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