a3 # group-project # gui

Building Menus with JMenuBar

TCSS 305 Programming Practicum

Menus are one of the most familiar GUI patterns — File, Edit, Help, and so on. Nearly every desktop application uses them, and users expect them to behave consistently. Swing provides a clean API for building menus, but the class hierarchy has a surprising twist that is worth understanding. This guide covers building menus from scratch, connecting them to actions, and handling the common Exit pattern cleanly.

---

1 The Menu Component Hierarchy

Before writing any code, you need to understand two different hierarchies: the **containment hierarchy** (what holds what in the UI) and the **inheritance hierarchy** (what extends what in the code).

1.1 The Containment (UI) Hierarchy

In the user interface, menus form a straightforward nesting structure:

```
JFrame
  └── JMenuBar
        ├── JMenu ("File")
        │     ├── JMenuItem ("New")
        │     ├── JMenuItem ("Open")
        │     ├── ── separator ──
        │     └── JMenuItem ("Exit")
        ├── JMenu ("Edit")
        │     ├── JMenuItem ("Undo")
        │     └── JMenu ("Find")          ← submenu
        │           ├── JMenuItem ("Find...")
        │           └── JMenuItem ("Replace...")
        └── JMenu ("Help")
              └── JMenuItem ("About")
```

- A `JFrame` has one `JMenuBar`
- A `JMenuBar` holds one or more `JMenu` items

- A `JMenu` holds `JMenuItem` items (and separators)
- A `JMenu` can hold other `JMenu` items — these become submenus

1.2 The Inheritance Paradox

Now look at the class inheritance:

```
javax.swing.JComponent
  └── javax.swing.JMenuBar

javax.swing.JComponent
  └── javax.swing.AbstractButton
        └── javax.swing.JMenuItem
              └── javax.swing.JMenu
```

> **The Inheritance Paradox**
>
> In the UI, a `JMenu` acts as a **container** — it is the parent that holds menu items. But in the inheritance hierarchy, `JMenu` **extends** `JMenuItem` — it is a child! This means a `JMenu` IS-A `JMenuItem`, which is why you can add a `JMenu` to another `JMenu` (creating submenus) or to a `JMenuBar`. This is a practical example of inheritance enabling polymorphism.

Here is the contrast side by side:

Perspective	Relationship
UI containment	`JMenu` is a **parent** — it holds `JMenuItem` objects
Class inheritance	`JMenu` is a **child** — it extends `JMenuItem`

This "paradox" is actually elegant: because `JMenu` IS-A `JMenuItem`, you can place a `JMenu` anywhere a `JMenuItem` is expected. That is exactly what makes submenus possible. The `JMenuBar` and `JMenu` containers accept `JMenuItem` objects, and since `JMenu` is a `JMenuItem`, it slots right in.

Menu Bar Placement on macOS

On macOS, native applications place the menu bar at the **top of the screen**, not at the top of the application window. Swing does not do this by default — `setJMenuBar()` places the menu bar at the top of the `JFrame`, which looks out of place on macOS. To get a native macOS menu bar at the top of the screen, you need a third-party Look and Feel such as FlatLaf. This is outside the scope of this course, but feel free to research it on your own.

---

2 Creating a JMenuBar

Creating a menu bar is straightforward:

```
final JMenuBar menuBar = new JMenuBar();
```

The critical step is how you **attach** it to the frame.

Attaching to JFrame — `setJMenuBar`, NOT `add`

There is only one correct way to attach a menu bar to a frame:

```
// CORRECT: Places the menu bar in its own dedicated region
above the content pane
frame.setJMenuBar(menuBar);

// WRONG: Treats the menu bar as a regular component — breaks
menu behavior!
frame.add(menuBar, BorderLayout.NORTH);
```

Using `add()` instead of `setJMenuBar()` treats the menu bar as an ordinary component. This breaks keyboard navigation, mnemonic keys, and look-and-feel integration. The menu bar may appear to work visually, but it will not behave correctly. Always use `setJMenuBar()`.

`JMenuBar` vs `MenuBar`

`MenuBar` is the old AWT class — do not use it. `JMenuBar` is the Swing class — always use this one. The same pattern applies throughout: `JMenu` vs `Menu`, `JMenuItem` vs `MenuItem`. Always use the J-prefixed Swing versions.

---

3 Creating Menus and Menu Items

3.1 JMenu — The Dropdown Trigger

A `JMenu` is the clickable label that appears on the menu bar and opens a dropdown when clicked:

```
final JMenu fileMenu = new JMenu("File");
```

You can assign a **mnemonic** so the user can open the menu with the keyboard:

```
fileMenu.setMnemonic(KeyEvent.VK_F);  // Alt+F opens the File menu
```

The mnemonic letter is typically underlined in the menu text. Choose a letter that appears in the menu name — `F` for File, `E` for Edit, `H` for Help.

> **Mnemonics on macOS**
>
> On macOS, mnemonics do **not** work the same way as on Windows and Linux. macOS does not use the Alt key for menu navigation — it uses the system menu bar and the Command key instead. When running a Swing application on macOS, mnemonic underlines will not appear and Alt+letter shortcuts will not activate menus. Your menus still work fine with mouse clicks, but do not rely on mnemonics as the only keyboard access path if your application needs to run on macOS.

3.2 JMenuItem — The Actionable Item

A `JMenuItem` is an individual actionable item inside a menu dropdown:

```
final JMenuItem openItem = new JMenuItem("Open...");
```

You wire up behavior with an `ActionListener`, just like a `JButton` — because it IS a button via inheritance! Look back at the inheritance tree in Section 1: `JMenuItem` extends `AbstractButton`, the same parent class as `JButton`. That shared ancestry is why menu items and buttons use the same `addActionListener` pattern.

```
openItem.addActionListener(e -> {
    // Handle the "Open" action
    System.out.println("Open selected");
});
```

3.3 Assembling the Menu

The assembly follows the same **Create, Configure, Add** pattern used throughout Swing:

```
// Create the menu bar
final JMenuBar menuBar = new JMenuBar();

// Create a menu
final JMenu fileMenu = new JMenu("File");
fileMenu.setMnemonic(KeyEvent.VK_F);

// Create menu items
final JMenuItem newItem = new JMenuItem("New");
newItem.addActionListener(e -> handleNew());

final JMenuItem openItem = new JMenuItem("Open...");
openItem.addActionListener(e -> handleOpen());

final JMenuItem exitItem = new JMenuItem("Exit");
exitItem.addActionListener(e -> handleExit());

// Add items to the menu
fileMenu.add(newItem);
fileMenu.add(openItem);
fileMenu.addSeparator();   // Visual divider between groups
fileMenu.add(exitItem);

// Add the menu to the bar
menuBar.add(fileMenu);

// Attach the bar to the frame
frame.setJMenuBar(menuBar);
```

Notice the call to `addSeparator()`. Separators are thin horizontal lines that visually group related items. In the example above, Exit is separated from New and Open because it is a fundamentally different action (closing the application vs. working with files).

---

4 Submenus

Because `JMenu` IS-A `JMenuItem` (the inheritance paradox from Section 1), creating a submenu is as simple as adding a `JMenu` to another `JMenu`:

```
// Parent menu
final JMenu editMenu = new JMenu("Edit");
editMenu.setMnemonic(KeyEvent.VK_E);

// Submenu
final JMenu findMenu = new JMenu("Find");
```

```
final JMenuItem findItem = new JMenuItem("Find...");
final JMenuItem replaceItem = new JMenuItem("Replace...");
findMenu.add(findItem);
findMenu.add(replaceItem);

// Add the submenu to the parent — it appears with an arrow indicator
editMenu.add(findMenu);
```

When the user hovers over the "Find" entry in the Edit menu, a secondary dropdown appears to the right with "Find..." and "Replace...".

> **Tip**
>
> Avoid nesting more than one level of submenus. Deeply nested menus are difficult to navigate and frustrating to use. If you find yourself needing multiple submenu levels, consider reorganizing your menu structure or using a dialog instead.

---

5 Specialized Menu Items

Swing provides menu items beyond the basic `JMenuItem` for common UI patterns.

5.1 JCheckBoxMenuItem — Toggle On/Off

A `JCheckBoxMenuItem` displays a checkmark when selected. Use it for options the user can toggle, such as "Show Grid" or "Word Wrap":

```
final JCheckBoxMenuItem gridItem = new JCheckBoxMenuItem("Show Grid");
gridItem.addActionListener(e -> {
    final boolean showGrid = gridItem.isSelected();
    canvas.setGridVisible(showGrid);
    canvas.repaint();
});
viewMenu.add(gridItem);
```

Call `isSelected()` to check the current state.

5.2 JRadioButtonMenuItem — Exclusive Selection

A `JRadioButtonMenuItem` works like a radio button — only one item in a group can be selected at a time. Use a `ButtonGroup` to enforce mutual exclusivity:

```
final ButtonGroup thicknessGroup = new ButtonGroup();

final JMenu thicknessMenu = new JMenu("Line Thickness");
for (final int size : new int[]{1, 2, 4, 8}) {
    final JRadioButtonMenuItem item =
            new JRadioButtonMenuItem(size + " px");
    item.addActionListener(e -> canvas.setLineThickness(size));
    thicknessGroup.add(item);   // Enforce mutual exclusivity
    thicknessMenu.add(item);    // Add to the menu for display
}
```

Note

The `ButtonGroup` does not display anything — it only enforces the "one selected at a time" rule. You still need to add each item to a `JMenu` (or other container) for it to appear in the UI.

5.3 Menu Items with Icons

You can add an icon to any menu item by passing it to the constructor:

```
final ImageIcon saveIcon = new ImageIcon("icons/save.png");
final JMenuItem saveItem = new JMenuItem("Save", saveIcon);
```

Icons appear to the left of the menu item text. Use small icons (typically 16x16 pixels) to match standard menu item sizing.

---

6 Connecting Exit to JFrame (Factory Method Pattern)

This section covers an important architectural pattern. Your menu-building code often lives in a `JPanel` subclass, but the Exit action needs to close the `JFrame`. How do you bridge that gap cleanly?

6.1 The Problem

Consider a common application structure: you have a `JPanel` subclass that serves as the content pane. This panel knows about the application's UI components and is a logical place to build the menu. But the Exit menu item needs to close the `JFrame` — and the panel should not store a reference to the frame as a field.

6.2 The Solution: A Factory Method

Create a method on the panel that accepts the `JFrame` as a parameter and returns a fully-built `JMenuBar`:

```
/**
 * Creates and returns the menu bar for this application.
 *
 * @param theFrame the JFrame that contains this panel
 * @return the configured JMenuBar
 */
private JMenuBar createMenu(final JFrame theFrame) {
    final JMenuBar menuBar = new JMenuBar();

    // Build the File menu
    final JMenu fileMenu = new JMenu("File");
    fileMenu.setMnemonic(KeyEvent.VK_F);

    final JMenuItem exitItem = new JMenuItem("Exit");
    exitItem.addActionListener(e ->
            theFrame.dispatchEvent(
                    new WindowEvent(theFrame, WindowEvent.WINDOW_CLOSING)));
    fileMenu.add(exitItem);

    menuBar.add(fileMenu);
    return menuBar;
}
```

The caller — typically in `createAndShowGui` or similar setup code — wires it together:

```
private static void createAndShowGui() {
    final JFrame frame = new JFrame("My Application");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    final MyContentPane contentPane = new MyContentPane();
    frame.setContentPane(contentPane);

    // The panel builds the menu; the frame attaches it
    frame.setJMenuBar(contentPane.createMenu(frame));

    frame.pack();
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
}
```

6.3 Why Dispatch WINDOW_CLOSING?

The Exit action dispatches a `WindowEvent.WINDOW_CLOSING` event rather than calling `System.exit(0)` or `frame.dispose()`:

```
exitItem.addActionListener(e ->
        theFrame.dispatchEvent(
                new WindowEvent(theFrame, WindowEvent.WINDOW_CLOSING)));
```

Why `WINDOW_CLOSING` Instead of `System.exit(0)`?

Dispatching `WINDOW_CLOSING` triggers the **exact same path** as the user clicking the X button on the window. This means:

- Any `WindowListener` registered for cleanup (saving files, releasing resources) runs properly
- The `defaultCloseOperation` set on the frame is respected
- There is **one consistent way** to close the window, no matter how it is triggered — the X button and the Exit menu item behave identically

Calling `System.exit(0)` bypasses all of this. It terminates the JVM immediately without giving listeners a chance to run. Calling `frame.dispose()` closes the window but may not trigger the same cleanup sequence depending on how listeners are registered.

Tip

Notice that the `JPanel` does **not** store the `JFrame` as a field. The frame reference is only needed during menu construction and is captured by the lambda expression. This keeps the dependency minimal — the panel does not maintain a permanent reference to its container.

---

7 JOptionPane for Quick Dialogs

Menus often need to display simple dialogs — an "About" box, a confirmation prompt, or an error message. Swing's `JOptionPane` provides static methods for common dialog types so you do not need to build dialog windows from scratch.

7.1 Message Dialogs

The most common use is a simple informational popup:

```
final JMenuItem aboutItem = new JMenuItem("About...");
aboutItem.addActionListener(e ->
        JOptionPane.showMessageDialog(
                this,                    // parent component
```

```
                "My Application v1.0\nTCSS 305",  // message
                "About",                  // title
                JOptionPane.INFORMATION_MESSAGE));  // icon type
helpMenu.add(aboutItem);
```

The first argument — the **parent component** — determines where the dialog appears on screen. Passing `this` (the panel) centers the dialog over the application window. Passing `null` centers it on the screen, which can be disorienting if the application window is off to one side.

7.2 Other Dialog Types

`JOptionPane` offers several other static methods for common dialog patterns:

Method	Purpose	Returns
`showMessageDialog(parent, message)`	Display information	`void`
`showConfirmDialog(parent, message)`	Yes/No/Cancel prompt	`int` (YES_OPTION, NO_OPTION, etc.)
`showInputDialog(parent, message)`	Text input prompt	`String` (or `null` if cancelled)

```
// Confirm dialog example
final int choice = JOptionPane.showConfirmDialog(
        this,
        "Are you sure you want to clear?",
        "Confirm Clear",
        JOptionPane.YES_NO_OPTION);

if (choice == JOptionPane.YES_OPTION) {
    canvas.clear();
}
```

Summary

Concept	Key Point
JMenuBar	Attach with `setJMenuBar()`, not `add()`

Concept	Key Point
JMenu	Acts as container (UI) but extends `JMenuItem` (inheritance)
JMenuItem	Actionable item — add `ActionListener` for behavior
JMenuBar vs MenuBar	Always use J-prefixed Swing classes, not AWT
Submenus	Add a `JMenu` to another `JMenu` — works because of IS-A
JCheckBoxMenuItem	Toggle option with checkmark; use `isSelected()`
JRadioButtonMenuItem	Exclusive selection; use `ButtonGroup` for mutual exclusivity
Exit pattern	Pass `JFrame` to factory method, dispatch `WINDOW_CLOSING`
JOptionPane	Quick dialogs for About, confirmation, input, etc.

Further Reading

External Resources

- Oracle Swing Tutorial: How to Use Menus — Comprehensive official tutorial covering menu bars, menus, and menu items
- Oracle Swing Tutorial: How to Make Dialogs — Official tutorial covering JOptionPane and custom dialogs
- Oracle Swing Tutorial: How to Use Buttons, Check Boxes, and Radio Buttons — Covers ButtonGroup, which also applies to radio button menu items

---

References

Primary Texts:

- Horstmann, C. S. (2022). *Core Java, Volume I: Fundamentals* (12th ed.). Oracle Press. — Chapter on Swing UI components, including menus, menu items, and dialog boxes.

- Bloch, J. (2018). *Effective Java* (3rd ed.). Addison-Wesley. — Item 18: Favor composition over inheritance (relevant to understanding the JMenu/JMenuItem inheritance design).

Language Documentation:

- Oracle JDK 25: JMenuBar — JMenuBar class API reference
- Oracle JDK 25: JMenu — JMenu class API reference
- Oracle JDK 25: JMenuItem — JMenuItem class API reference
- Oracle JDK 25: JCheckBoxMenuItem — JCheckBoxMenuItem class API reference
- Oracle JDK 25: JRadioButtonMenuItem — JRadioButtonMenuItem class API reference
- Oracle JDK 25: JOptionPane — JOptionPane class API reference
- Oracle JDK 25: WindowEvent — WindowEvent class API reference

Tutorials:

- Oracle Swing Tutorial: How to Use Menus — Official menu tutorial with examples
- Oracle Swing Tutorial: How to Make Dialogs — Official dialog tutorial

---

This guide is part of TCSS 305 Programming Practicum, School of Engineering and Technology, University of Washington Tacoma.