Slide 1

Administrivia

- Reminder: Homework 5 code due today. Homework 6 design due next Tuesday.

Slide 2

Java GUI Libraries — Recap

- Many, many classes for GUI components — pre-defined components (e.g., `JButton`), containers (e.g., `JPanel`).

  (Can also define your own "custom components". More about them shortly.)
- How things are arranged on screen is controlled by "layout manager". Can nest containers, giving them different layout managers.
- How things work depends on "event listener" methods. Good place to use anonymous inner classes.

Slide 3

Java GUI Libraries — Design Tips

- Probably better not to mix AWT and Swing unless necessary (e.g., unless you're doing an AWT-only program, prefer `JFrame` to `Frame`).
- To find out how to use components — skim online API, Oracle/Sun tutorials (follow links from API), look for examples similar to what you want to do.
- For small programs, okay to put GUI and underlying data all in one class. For larger programs, consider separating them — "Model/View/Controller" design pattern.
- GUI components that must be accessed by more than one method — e.g., by listener methods — should be instance variables. Other components can often be declared locally in constructor.
- (Examples as time permits.)

Slide 4

Sidebar: Concurrency Basics

- Textbooks on operating systems talk about "processes" — "threads of control" executing "concurrently", i.e., at the same time (in fact or in effect).

  Each is a sequence of steps, like the (sequential) programs you've written.
- How does it work? Conceptually, all processes not waiting for something (such as I/O) run at the same time. Operating system basically simulates one CPU per thread, with real CPU(s) switching back and forth among them.
- This turns out to be a good mental model for managing applications, and activities of the O/S itself. It also means you could get better performance with more than one CPU/core — can potentially have more than one thing actually running at the same time.
- But there are some potential pitfalls, involving interaction among processes/threads.

Slide 5

Sidebar Continued

- Two basic models — one in which the concurrently-executing things don't share (much) memory and one in which they do. Sharing memory has benefits but also some serious potential pitfalls ("race conditions").
- Java provides some support for both models, but at this point its support for the shared-memory model is more relevant, because . . . (to be continued).

Slide 6

Java GUI Classes and Multithreading

- Currently Java GUI classes are implemented in terms of an "event dispatch thread" (EDT) — something that listens (to some part of the operating system/environment?) for "events" (from keyboard or mouse, e.g.) and "dispatches" them by calling appropriate methods associated with GUI components. There could be other threads active at the same time.
- Not all of what's under the hood is "thread-safe" (okay to call from multiple concurrently-executing things), so Oracle/Sun recommends that all changes to GUI components be done in the EDT. This happens automatically with listener methods. Accesses from the "main" thread and from other threads should use `SwingUtilities.invokeLater`.

Slide 7

Multithreading and the Game Framework

- Listener methods run in the EDT. Other methods run in a different thread.
- Problem? Maybe. Concurrent access to simple primitive types (`boolean`, `int`) is pretty safe — the worst that's likely to happen is that changes made by one thread aren't immediately visible to others. (Probably any variables that are used both in listener methods and by the rest of the game should be declared `volatile` to help with this.) But anything involving more complicated data structures is probably a bad idea without explicit synchronization (to be discussed soon).

Slide 8

Graphics in Java — Custom Components

- Predefined components (`JButton`, etc.) do a lot, but what if you want something that's not provided? in particular, you want to control the image yourself?
- Make a custom component — define a subclass of a component that provides some of the needed functionality, and override the method that defines what's displayed.

  E.g., subclass `JPanel` and override `paintComponent`, to include your code to "paint" the panel.
- Call `repaint` when ready to redisplay.

Slide 9

Custom Painting

- Method to override is

  `public void paintComponent(Graphics g)`.

  `g` is a "graphics context" that you can draw on. (Actually it's a `Graphics2D`.) Tutorial recommends calling `super.paintComponent(g)` before doing anything else.
- Can get dimensions of panel with `getSize`, `getHeight`, `getWidth`, `getInsets`.

Slide 10

Custom Painting, Continued

- Can set colors, draw shapes, lines, text, etc., etc. — see `Graphics` and `Graphics2D` classes. Coordinate system is similar to what you're using in your game. See code in `BasicBlock` for simple example.
- General advice — look over the methods of `Graphics` and `Graphics2D`; if confused, follow links to tutorial(s) and look for a suitable example to adapt.

Slide 11

Drawing and Filling Shapes

- "Draw" means draw outline only; "fill" to draw and fill.
- `Graphics` provides methods for doing simple shapes. `Graphics2D` provides more general methods. (Look at some shapes in `java.awt.geom`.)
- You already know (from your game) about simple ways to control color of what's painted. The `Graphics2D` class provides a lot more options (next slide).

Slide 12

Drawing and Filling Shapes, Continued

- `Graphics2D` provides, among other things:
  - `setPaint` to fill shapes with simple color, gradient fill, etc.
  - `setStroke` to draw outlines with different widths, etc.
  - `setFont` to draw text in different fonts. (This works for text components such as `JLabel` too.)
- And there's more — "clipping", affine transformations (e.g., rotation — transformations in which parallel lines stay parallel), etc., etc.
- (Examples as time permits.)

Minute Essay

- Homework 6 will ask you to add something to your game — probably a panel or panels along the edges of the playing field, or an addition to the menu bar — that makes use of the Java GUI libraries. How do you think this might be useful for your game?