Slide 1

Administrivia

- Reminder: Homework 5 design due today, code Thursday.
- Homework 6 due dates posted (next week).

Slide 2

Priority Queue — Review

- Value — list of elements, of some type we can put in order.
- Operations:
  - Add element.
  - Remove element with lowest (or highest) value.
  - “Is empty?”

  (Look at game framework `PriorityQueue` interface for a slightly different, but equivalent, list. You will write one of these for Homework 5.)
- How to implement? (List kept in order by value seems best. Finish code.)

GUIs and Event-Driven Programming

- In PAD I (and in most previous in-class examples this semester) we usually focus on programs with simple text-based input and output — a basically synchronous interaction with the user.
- Programs with GUIs, though, are typically somewhat different — the main program (which is sometimes hidden in library code) is often just a loop that waits for keyboard/mouse input delivered by the program's environment (operating system, graphical environment, window manager, etc.).
- This leads to an "event-driven" programming model that can seem rather different from what's used for text-based programs. (But it's rather like what you're doing in the game project.)

Java GUI Libraries

- Java has evolved over its short lifetime, and sometimes there seems to be more than one way to do something. One example — resizable arrays (`Vector` versus `ArrayList`). Another — two groups of GUI-related classes:
  - Abstract Window Toolkit (AWT) — older, "look and feel" consistent with platform's windowing system.
  - Swing — newer, more extensive, look and feel more aimed at being consistent across platforms. Makes use of AWT components.
- Many, many classes to build GUIs:
  - GUI elements —- buttons, labels, text boxes, menus, etc., etc., etc., etc.
  - "Containers" to group elements and arrange them for display.
  - "Listeners" and "events" to allow program to respond to user input.

Slide 5

Some GUI Classes

- `Component` — base class.
- `Container` – component that can contain other components.
- `JFrame` — window with titlebar, etc.; usually the "main" window for an application.
- `JDialog` — popup dialog box.
- `JPanel` – very simple container, useful for grouping things, providing custom graphics.
- `JMenuBar`.
- Etc., etc., etc., etc. — far more than we can cover in this course! Read the API. Some classes have links to online tutorials too.

Slide 6

Using the GUI Classes — Appearance

- When using predefined components, key issue is how they're grouped using containers and how things are laid out within each container.
- Preferred method for laying things out — layout manager, which places elements in some reasonable way, does something reasonable if container is resized.
  - Simple layouts include `FlowLayout`, `GridLayout`, `BorderLayout`, `BoxLayout`.
  - `GridBagLayout` provides more control, but is more complex.

  Some of them expand components to fit, others lay them out at their minimum size. See API and tutorials for more info.
- Often makes sense to group elements hierarchically — `JPanel` is useful for that.

Slide 7

Using the GUI Classes — Behavior

- Runtime system (JVM) translates each user action (keyboard or mouse input) into an “event” and then calls method(s) in “event listener” objects.
- So, to tell the runtime system what should happen when, e.g., a `JButton` is clicked, call button’s `addActionListener` with an object `listener` that implements `ActionListener` interface. Now when the button is clicked, `listener`’s `actionPerformed` method is called.
- Several approaches to defining listener objects. One is to have “main” class implement required interface. Another is to use anonymous inner classes.

Slide 8

Examples

- (Examples as time permits.)
- Before going further, we need what seems like a detour . . .

Sidebar: Concurrency Basics

Slide 9

- 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.

Sidebar Continued

Slide 10

- 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).

Minute Essay

- None — quiz.