Slide 1

Administrivia

- Reminder: Homework 2 design due today (midnight), code next Tuesday.
- (Review quiz — and notice that there's a sample solution online.)

Slide 2

Homework 2 Code — Some Tips

- Eclipse will suggest adding a variable called `serialVersionUID` to some of your classes. Do that. (Notice there's one of these in some of the provided code.) Value can be anything. We will talk later about what this means and how to make use of it.
- Notice that x/y coordinates of framework are opposite of row/column. `getSize()` in screen class should return width by height.
- To confirm that your code works:
  - Start the game, and verify that the playing field is what you defined (dimensions, plus appearance of blocks — for now, solid colors are okay).
  - Try running the screen editor (directions in "project description" document). If it comes up, and shows all the kinds of blocks you defined, all is well. (Actually it doesn't have to do that if you don't plan to use it — it just has to not crash.)

Slide 3

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.

Slide 4

Processes Versus Threads

- Two basic ways to implement this idea of concurrent execution — "processes" and "threads".
- "Processes" don't (usually) share memory, and must communicate in some fairly restricted way.
- "Threads" do share memory, which is convenient but has potential pitfalls ("race conditions").

Slide 5

Multithreading in Java

- Much interest recently in "multithreaded" programming, because of hardware changes — having more than one CPU/core not just for high-end systems.
- Interestingly enough, Java has included support for multiple threads from the beginning. Interaction among Java threads based on "monitors" (see textbooks on operating systems, parallel programming — idea goes back to 1975 papers by Hoare and Brinch Hansen). Java leaves out some aspects of full-fledged idea, but keeps enough to be useful.

Slide 6

Threads in Java

- `Thread` class provides basic functionality. To start a new thread, make a `Thread` object and call its `start` method. Two choices:
  - Create a `Thread` with an object that implements `Runnable` — `run` method has code to execute.
  - Define a subclass of `Thread` that has a `run` method with code to execute.
- Interthread interaction based on (implicit) locks:
  - Every object (and every class) has a lock.
  - `synchronized` methods must acquire lock — so only one at a time can run.
  - `wait` gives up the lock and sleeps; `notify` and `notifyAll` wake up one/all sleeping thread(s).

Slide 7

Threads in Java, Continued

- Other useful methods:
  - `Thread.sleep` makes current thread sleep for some interval.
  - `t.join` wait for `Thread t` to finish.
  - `t.interrupt` interrupts `Thread t` (which can check whether it has been interrupted with `isInterrupted` — safe/approved way for one thread to stop another.
- Can set thread priorities — sometimes useful, but not a substitute for proper synchronization.
- Lots of new threads-related stuff in Java 1.5 / 5.0 (`java.util.concurrent` package).

Slide 8

Uses for Threads in Java

- Formerly many uses for multithreading in GUIs (e.g., animation), but now most can be accomplished with new features of GUI classes (e.g., timers). Still useful, however, if you want something that might take a while to execute in the background.
- Multithreading also potentially useful for improving performance of computationally intensive code.
- Examples as time permits . . . .

Slide 9

Minute Essay

- `synchronized` methods/blocks can solve some problems. Can you think of ways in which they could lead to other problems?

Slide 10

Minute Essay Answer

- (What I had in mind was the possibility of what's called deadlock — threads holding locks in a way that every thread is waiting for some other thread, and no thread can proceed.)