

Slide 1

Administrivia

- Reminder: Homework 6 design due today, code next week.

Slide 2

GUIs and Graphics in Java — Recap/Review

- Java libraries include many, many classes for both GUIs and graphics — many predefined components, plus support for building custom classes.
- Sample programs on the course “sample programs” page include some we won’t have time to talk about but that could be helpful in working on your game (e.g., there is an example showing how to make pop-up dialogs, and another showing how to load an image from a file).
- (Examples as time permits.)

Homework 6 — GUI Features

Slide 3

- Overall layout of game is `BorderLayout`, with screen in middle and “game status panels” on four sides — returned by `getGameStatusPanel` (in `Player`), usually a `JPanel`.
- Menu bar is in `GameSetup`, can be modified.
- Screen editor program has support for “editing properties” (of screens, blocks, entities) — `getEditPropertiesPanel`. Could use this to give slightly different properties to different instances (e.g., walls of different colors, enemies with different speeds).
- Homework 6 asks you to use these features to (1) display something, and (2) get input from the user (either in the game or in the screen editor).

Why Parallel Computing

Slide 4

- It's been an article of faith for a *long* time that eventually we'd hit physical limits on speed of single CPUs, despite interpretation of Moore's law as “CPU speed doubles every 1.5 years.”
- But — strictly speaking, Moore's law says that the number of transistors that can be placed on a die doubles every 1.5 years.
- Historically that has meant — more or less — doubling speed and memory size. That seems to be at an end (for now?) — tricks hardware designers use to get more speed require higher power density, generate more heat, etc.
- So, what to do with all those transistors? Provide hardware support for doing more than one thing at a time (“parallelism” or “concurrency”).

Slide 5

One Approach — Multicore Chips

- Key idea here — chip includes several “cores”, all sharing one connection to memory.
- Each “core” is a processor in the sense we talk about in introductory courses and Computer Design; each typically has its own first-level cache.
- To fully exploit this for a single application, need multiple threads (or processes).

Slide 6

Another Approach — Hyper-Threading

- Key idea here — chip includes hardware support for having more than one thread at a time “active”, but strictly speaking only a single processing core. Replicated components include program counter, ALU.
- What this allows is very fine-grained concurrency (“more than one logical CPU”), which can hide latency. (Note, though, that performance improvements range from zero to about 30 percent.)
- To fully exploit this for a single application, need multiple threads (or processes).

Another Approach — Clusters

- In addition to hardware support for shared-memory parallelism — Ubiquity of networking makes almost any PC part of a “cluster”.
- Taking advantage of this also requires multiple processes.

Slide 7

Another Approach — GPGPU

- GPUs (“graphics processing units”) provide yet another kind of hardware parallelism, typically in the form of many processing elements that carry out the same sequence of operations more or less in lockstep, but each on its own data.
- Recently these processing elements have become sophisticated enough to be used for general-purpose computing (“GPGPU”).

Slide 8

Concurrency Basics — Review

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.

Processes Versus Threads

Slide 10

- 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 11

Multithreading in Java

- Interestingly enough, Java has included support for multiple threads from the beginning — probably because it's a good mental model for GUIs.
- 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 12

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).

Threads in Java, Continued

Slide 13

- 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 useful classes in `java.util.concurrent` and related packages, added in Java 1.5/5.0.

Uses for Threads in Java

Slide 14

- 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

Minute Essay

- None — sign in.

Slide 15

Minute Essay

- None — sign in.

Slide 16