Slide 1

Administrivia

- Reminder: Homework 6 code due today.
- (Homework 7 due dates on Web, writeup coming soon.)

Slide 2

Homework 7

- Writeup should be complete, but a short overview:
  - Objective is to write an alternate implementation for priority queue ADT and compare its performance to that of first implementation.
  - To compare performance, need to (1) add something to code to measure execution time, and (2) increase work being done by priority queue to the point where performance differences will show up.
- You can find code for a heap-based priority queue lots of places, but you will probably learn more if you write your own.
- Be sure to save a copy of your existing code before doing this, because you shouldn't include most of these changes in what you turn in as a "final" game (Homework 8).

Slide 3

Command-Line Arguments

- Many mechanisms for starting programs provide a way of passing them information without using files or standard input — "command-line arguments". Example — when you type at the command line

  ```
  ls -l myfile
  ```

  `-l` and `myfile` are passed to the `ls` in this way.

- C programs can receive command-line arguments by declaring `main` as

  ```
  int main(int argc, char *argv[])
  ```

  or equivalent, where `argc` is the number of arguments and `argv` is an array of C-style strings. By convention the zero-th argument is something identifying the program (e.g., its name). So in the `ls` example above, there would be three arguments . . .

Slide 4

Command-Line Arguments, Continued

- Java `main` methods also receive command-line arguments via arguments passed to `main`. `main` must always be declared with an argument of type `String[]`, which is a Java array containing the arguments. A Java equivalent of `ls` would get only two arguments for the example of the previous slide.

- Eclipse unfortunately doesn't make it that easy to invoke programs with command-line arguments that vary from execution to execution, but it's possible. An alternative is to run the program from the command line:

  ```
  java MainClass arg1 arg2
  ```

  or for your game something like

  ```
  java -classpath bin:PAD2.jar MainClass arg1
  arg2
  ```

  (Replace ":" with ";" on Windows.)

Slide 5

Networking Basics

- Inter-computer communication based on layered approach and “protocols”:
  - Application level — HTTP, FTP, telnet, SMTP, POP, IMAP, NTP, etc., etc.
  - Transport level — TCP (Transmission Control Protocol), UDP (User Datagram Protocol).
  - Network level — IP (Internet Protocol — addressing, routing of packets).
  - Link level — device drivers, etc.
- Messages are routed to
  - A machine (“host”), identified by IPA or name.
  - A process, identified by “port number” (16 bits). 0 — 1023 are “well-known ports”, others available for applications.

Slide 6

Networking Basics — TCP and UDP

- UDP — independent messages, no guarantees about reliability or message order — analogous to (snailmail) letter.
- TCP — point-to-point channel, guarantees reliability and message order — analogous to phone call. Endpoints called “sockets”.

Slide 7

Networking in Java

- Classes for communicating at application level — e.g., `URL` (“show URL” example).
- Classes for communicating at network level:
  - TCP — `Socket`, `ServerSocket`.
  - UDP — `Datagram*`.
- RMI (Remote Method Invocation).

Slide 8

Networking in Java — Sockets

- Client/server model:
  - Server sets up “server socket” specifying port number, then waits to accept connections. Connection generates socket.
  - Client connects to server by giving name/IPA and port number — generates a socket.
  - On each side, get input/output streams for socket. Program must define protocol for the two sides to communicate.
- Simple example in binary-I/O program from last. More complex example — chat program (next time).

Minute Essay

- None — quiz.