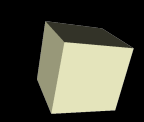
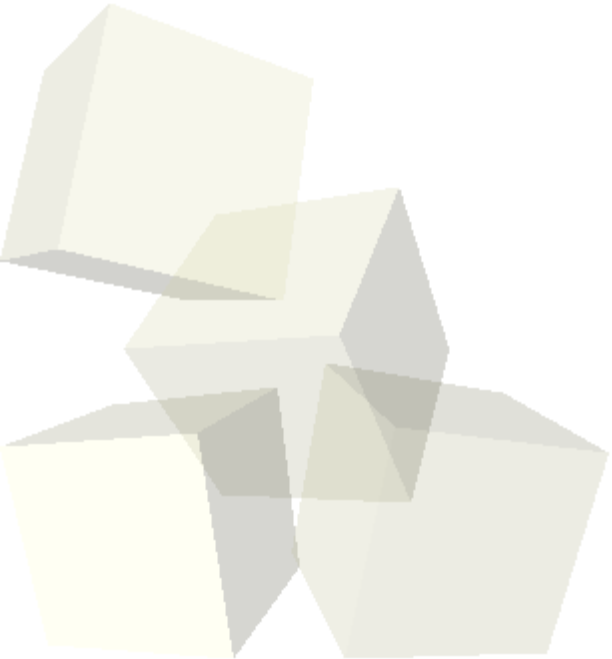


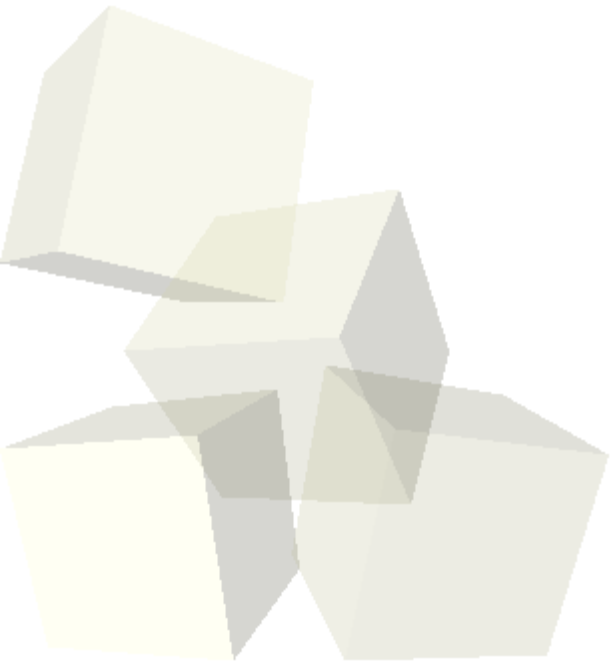
# Threads

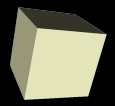
2/14/2008



# Opening Discussion

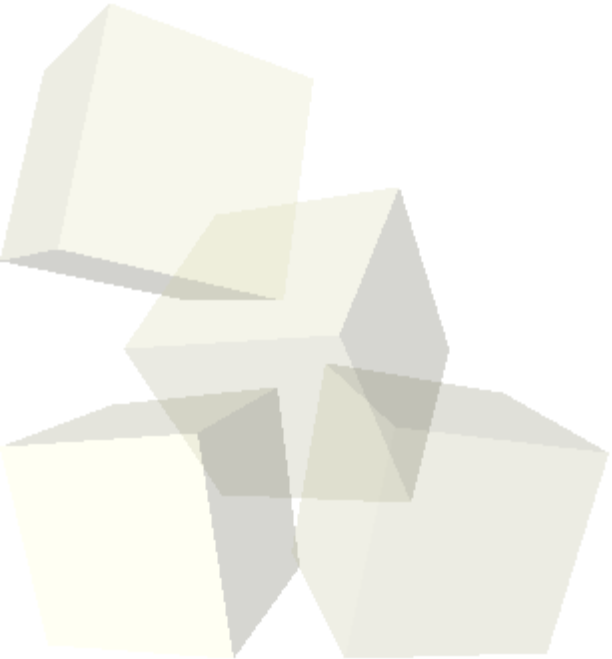
- Let's look at solutions to the interclass problem.
- Do you have any questions about the reading?
- Do you have any questions about the assignment?
- What is the most significant change in computer hardware in the last three years? What impact does this change have on developers?

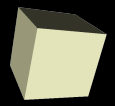




# Threads

- All the programming you have done so far has been in a single thread of execution. That is to say that the program goes from one line to the next doing one at a time in order. In a program with multiple threads the same thing happens, but in multiple places at once.





# Why Threads?

- On a machine with a single processor and a single core threads simply give the impression of two things happening at once. With the widespread arrival of multicore processors, most of the new machines have the ability to actually do two or more things at once, assuming programs have more than one thread.
- For at least a while, the future is about adding more cores to processors so software is going to have to change and that means programmers have to change as well.

# The Thread Class

- Most modern languages have the ability to do multithreading.  It is easier in some than in others.
- Java provides a simple way of doing this with the java.lang.Thread class.
- This class can also be useful even if you aren't using multiple threads because it has static methods that will impact the behavior of the current thread.
- The Thread class has been in Java since the language was created. Java 5 added a newer library to help with this task. We will cover it later in the semester.

# Spawning Threads

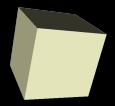
- To start a new thread, you simply need to create a Thread object and pass it an instance of a java.lang.Runnable object.
- Runnable is an interface with one method in it, run. When the thread object's start method is called, the other thread becomes active, and it will begin execution at the run method of the Runnable object. Control returns to the original thread and they execute in parallel.

# Thread Problems

- The primary problem one runs into with multithreaded programs is that threads share memory and more than one thread can access a piece of memory at once. This isn't a problem if they are just reading, but if any thread is writing you can have bad situations.
- An extreme condition would be to consider two threads operating on an array. Worst case is both are sorting the array at the same time. You could imagine one sorting while another tries to do a binary search and the results are similarly bad.
- The simplest (and most common) example is a bank account where a race condition occurs.

# Synchronization

- The way to prevent two threads from accessing the same piece of memory at the same time is to synchronize the critical pieces of the code. You can put the synchronized keyword in front of methods or make synchronized blocks.
- Each object and class in Java can have a monitor that is locked when synchronized code is being executed. Only one thread can hold the lock on the monitor at a given time. This insures that you never have two threads executing critical code on a single object at the same time.
- Too much synchronization slows things down or causes deadlock.

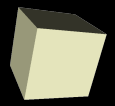


# Wait and Notify

- We can get even more control over how threads behave with the wait and notify methods.
- The wait method will stop the execution of a thread until some other thread tells it to continue execution. The notify and notifyAll methods are how threads tell other threads that they are supposed to wake up.
- All of these must be called by a thread that holds the monitor to the object they are being invoked on. Typically that means that are called from inside synchronized code.
- Wait should be called inside a while loop. Use notifyAll.

# Thread Coding

- The sorting code we wrote last time can provide a great test for threading. In this case we want to use our slow sorts so that we can actually time how long it takes them to run.
- I want you to add to your main some code that will create N (where N is an int variable) arrays of Doubles of length ARRAY_SIZE (you can declare that as a static final variable) then fill them with random values.
- First sort them one at a time, then refill them and sort them in N threads. Use System.nanoTime to time how long each of those takes.



# Minute Essay

- Why is learning how to do multithreaded programming so essential today?
- The design for assignment #2 is due today.
- Interclass Problem – Write code that prints the numbers 1-100. Then call that code in two threads that are executing in parallel and see what happens. Try printing with both System.out and System.err.