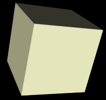
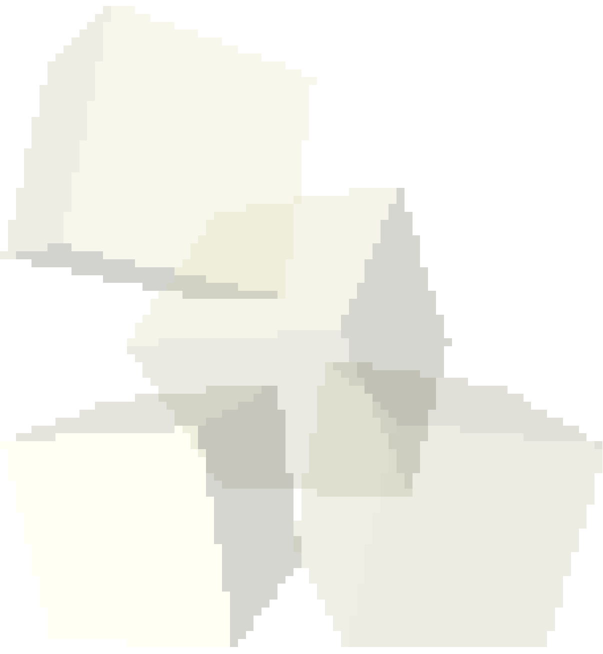
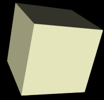


Concurrency and Files

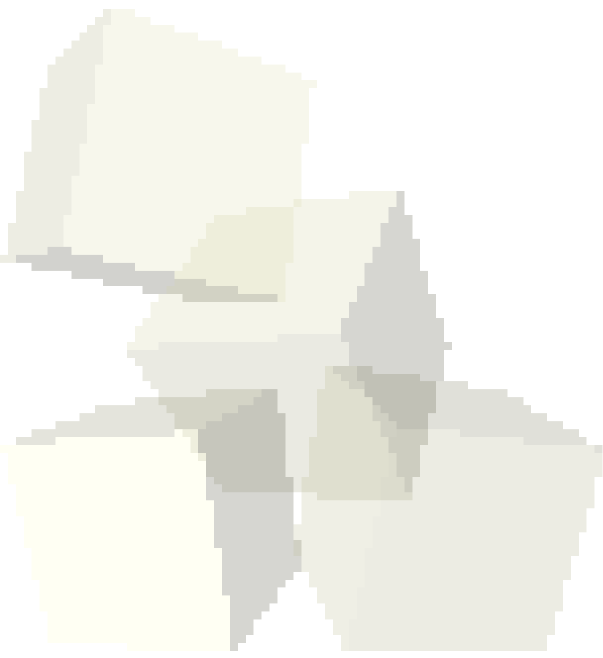
4-15-2010

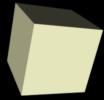




Opening Discussion

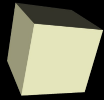
- Let's look at solutions to the interclass problem.
- Do you have any questions about the assignment?
- Let's quickly go over remove on the tree.





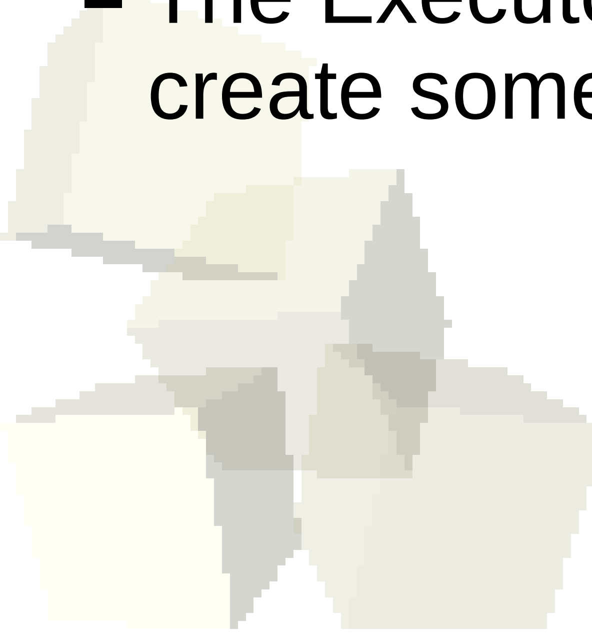
java.util.concurrent

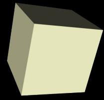
- While support of threading was built into the original version of Java, some common threading tasks still aren't easy.
- There are some things that you find yourself doing rather frequently with threaded code and there are some limitations with the basic libraries that make it difficult to use at times.
- The java.util.concurrent library was added to make common parallel tasks easier. It is built around a few key concepts. Let's go look at the javadocs for this library.



Executor Interface

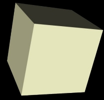
- The heart of the concurrent library is the Executor interface. It provides a method for running a Runnable object, but tells you nothing about how it will be run.
- The commonly used subtype of Executor is ExecutorService that supports the Callable<T> interface and Future<T> objects.
- The Executors utility class gives you a way to create some commonly used ExecutorServices.





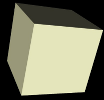
Blocking Queues

- One handy data structure in parallel applications is a blocking queue. This is a queue with a fixed number of slots in it.
- If you try to dequeue when the queue is empty, the thread will block until another thread adds something.
- If you try to enqueue when the queue is full, the thread will block until another thread removes something.



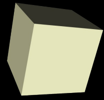
Coordinating Threads

- The concurrent library also provides a number of different classes to help with coordinating the activities of threads.
  - Semaphores – holds a number of “permits” that can be given out to threads.
  - CountDownLatch – stops threads until they have all hit the latch. Only works once.
  - Exchanger – two threads swap information at a particular point.
  - CyclicBarrier – like the CountDownLatch but works multiple times.



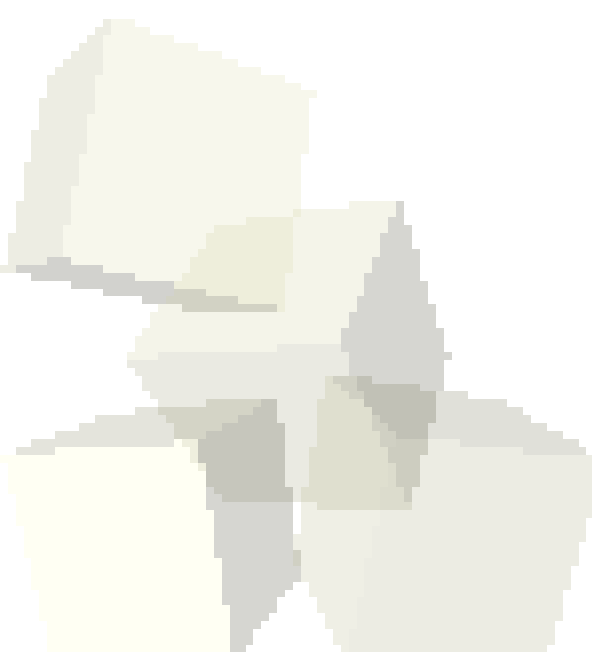
Atomic and Locks

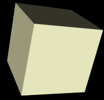
- There are two subpackages for java.util.concurrent: atomic and locks.
- The java.util.concurrent.atomic package provides data types that have atomic access methods. These methods can't be interrupted by other threads so they can be done in a thread safe way.
- In java.util.concurrent.locks you will find classes for locks that can be used in your program. Locking is basically what the synchronized keyword does, but that can't be shared across objects or methods.



Multithreading Code

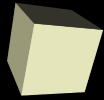
- Let's go ahead and and write something multithreaded. Instead of using the standard thread library I want us to use the facilities in java.util.concurrent.





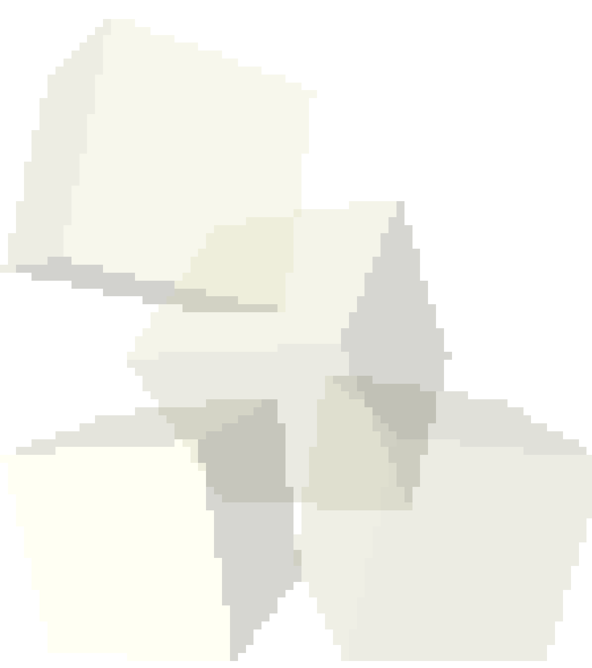
I/O Streams

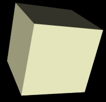
- The basic input and output streams that we use for I/O in Java are part of the java.io package.
- The package uses significant inheritance with the hierarchies rooted in the InputStream, OutputStream, Reader, and Writer abstract classes. The first two provide I/O based on bytes while the other two use characters.
- These base classes have very little functionality themselves and being abstract they can't even be instantiated.



File Streams

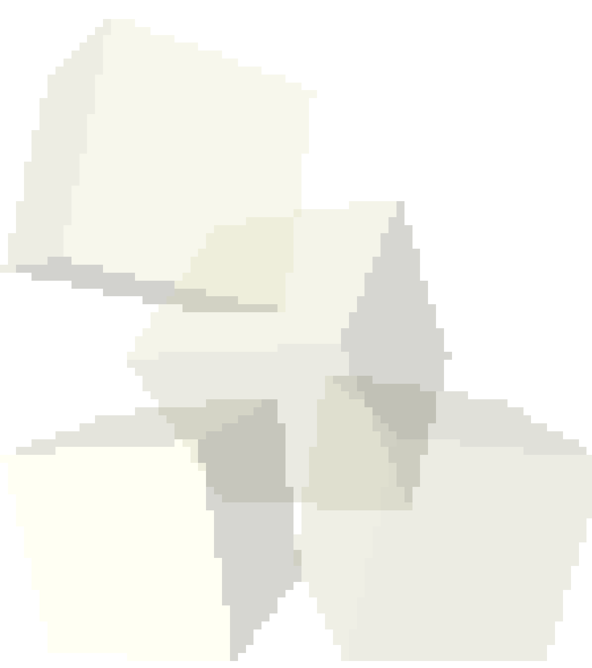
- In order to use streaming you have to be able to instantiate something. One set of classes that you can instantiate is the set of file streams.
- These classes are FileInputStream, FileOutputStream, FileReader, and FileWriter.
- Let's go look at these really quick.

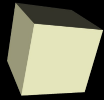




The File Class

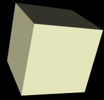
- java.io.File is a really handy class. It represents a file or directory, but it has many operations that make dealing with files across platforms easy.





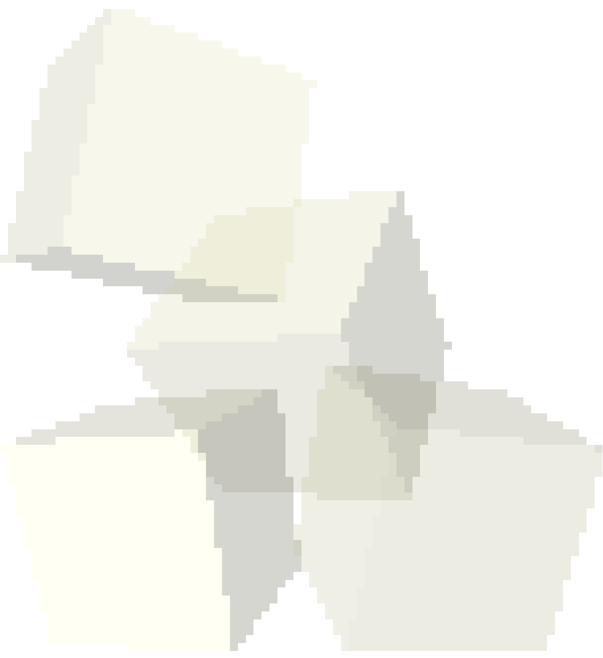
Wrapping Streams

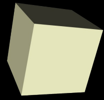
- The file stream classes still don't do much, they just do what their base class does except they are actually attached to a file.
- Being able to just read or write bytes is technically sufficient for any task, but you wouldn't want to write much code that way.
- We gain functionality by “wrapping” stream objects around one another. This is a design pattern called the Decorator.
- Example decorations include buffering, functionality for binary I/O (DataInputStream/DataOutputStream), or formatted printing (PrintWriter).



Coding Streams

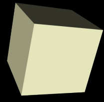
- We want to write some code that uses files and streams. A good example of this would be a simple text editor.
- We can add this functionality to our drawing program or you can write a standalone application. All it requires is a JTextArea in a GUI with save and load options.





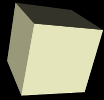
Power of Serialization

- Now we can take the next step. I want our drawing application to have the ability to save and load full drawings. What do we need to change in the code to make this happen? We basically have to take the entire object for our tree and write it out to file one element at a time.
- The task of converting an object into a stream of bytes is called serialization. In most languages it is a tough thing to do. Fortunately, Java has built in functionality to provide serialization.



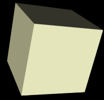
Serializing in Java

- To make it so that an object can be serialized we simply inherit from the interface Serializable. This is a “mix-in” interface that doesn't have any methods.
- The ObjectOutputStream and ObjectInputStream can be used to write and read whole objects that are Serializable. If it, or some part of it, isn't Serializable an exception will be thrown.
- Elements that you don't want written (or that can't be written) can be labeled as transient.



Challenges

- “With great power comes great responsibility.”
- Serialization is truly powerful, but you shouldn't just make everything Serializable because there are costs.
- Anything that inherits from a Serializable class/interface is itself Serializable.
- The default serialization can be expensive and potentially leads to security holes where people can find out about details of your objects that are otherwise private.



Minute Essay

- What questions do you have about streams and files? How does the use of the decorator pattern improve the flexibility of the library?
- Interclass problem – Create a program that will write an array of random doubles to a file and read it back in. Do this in two ways: using data streams and using object streams.