4/17/2008

- Let's look at solutions to the interclass problem.
- Do you have any questions about the assignment?
- Do you have any questions about the reading?
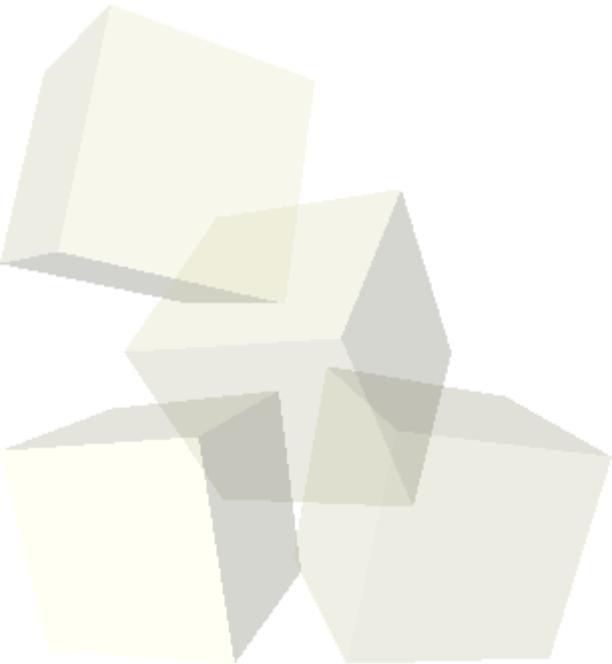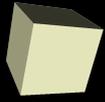
- First let's finish off Chris's code to animate the maze traversal using a thread from an executor.
- We talked about how we could multithread our mergesort. Let's go ahead and do it. Instead of using the standard thread library I want us to use the facilities in java.util.concurrent.
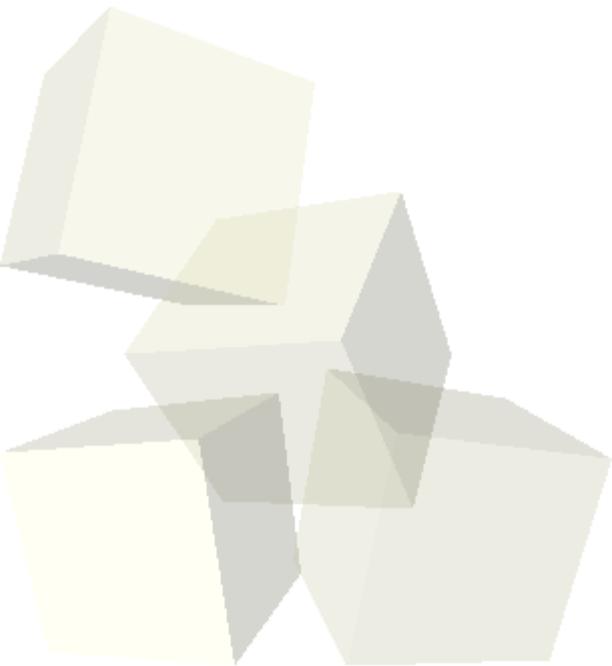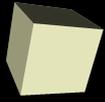
- The basic input and output streams that we use for I/O in Java are part of the java.io package.
- The package uses significant inheritance with the hierarchies rooted in the InputStream, OutputStream, Reader, and Writer abstract classes. The first two provide I/O based on bytes while the other two use characters.
- These base classes have very little functionality themselves and being abstract they can't even be instantiated.
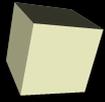
- In order to use streaming you have to be able to instantiate something. One set of classes that you can instantiate is the set of file streams.
- These classes are FileInputStream, FileOutputStream, FileReader, and FileWriter.
- Let's go look at these really quick.
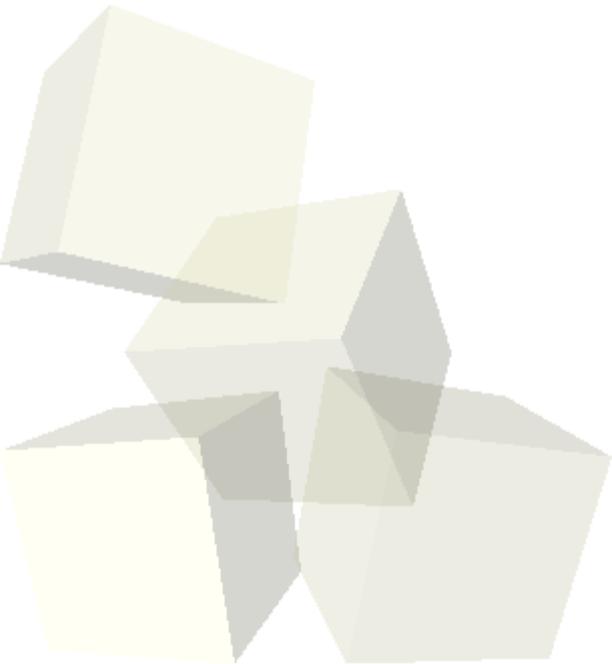
- The file stream classes still don't do much, they just do what their base class does except they are actually attached to a file.
- Being able to just read or write bytes is technically sufficient for any task, but you wouldn't want to write much code that way.
- We gain functionality by "wrapping" 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).
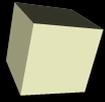
- We want to write some code that uses files and streams. A good example of this would be a simple text editor.
- We can add this functionality to our drawing program or you can write a standalone application. All it requires is a JTextArea in a GUI with save and load options.
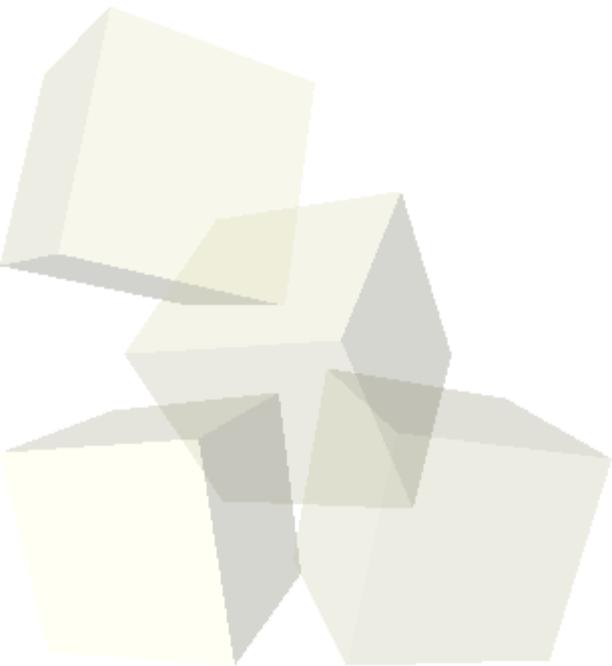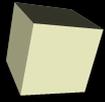
- Now we can take the next step. I want our drawing application to have the ability to save and load full drawing. What do we need to change in the code to make this happen? We basically have to take the entire object for our tree and write it out to file one element at a time.
- The task of converting 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.
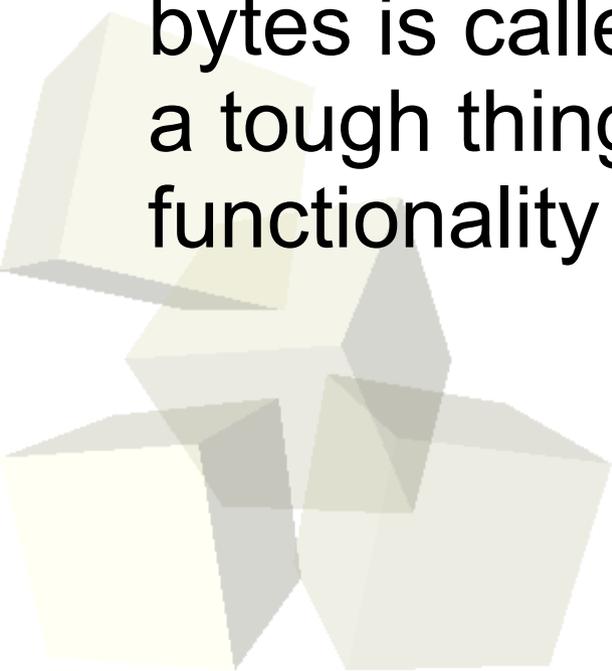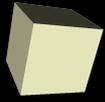
- To make it so that an object can be serialize we simply inherit from the interface Serializable. This is a "mix-in" interface that doesn't have any methods.
- The ObjectOutputStream and ObjectInputStream can be used to write and read whole objects that are Serializable. If it, or some part of it, isn't Serializable an exception will be thrown.
- Elements that you don't want written (or that can't be written) can be labeled as transient.
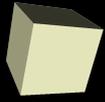
- "With great power comes great responsibility."
- Serialization is truly powerful, but you shouldn't just make everything Serializable because there are costs.
- Anything that inherits from a Serializable class/interface is itself Serializable.
- The default serialization can be expensive and potentially leads to security holes where people can find out about details of your objects that are otherwise private.
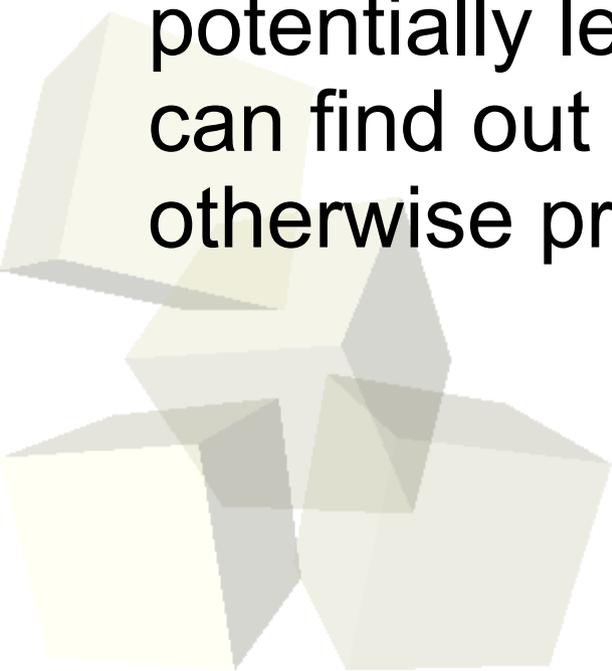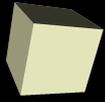
- What questions do you have about streams and files? How does the use of the decorator pattern improve the flexibility of the library?
- Interclass problem – Create a program that will write an array of random doubles to a file and read it back in. Do this in two ways: using data streams and using object streams.