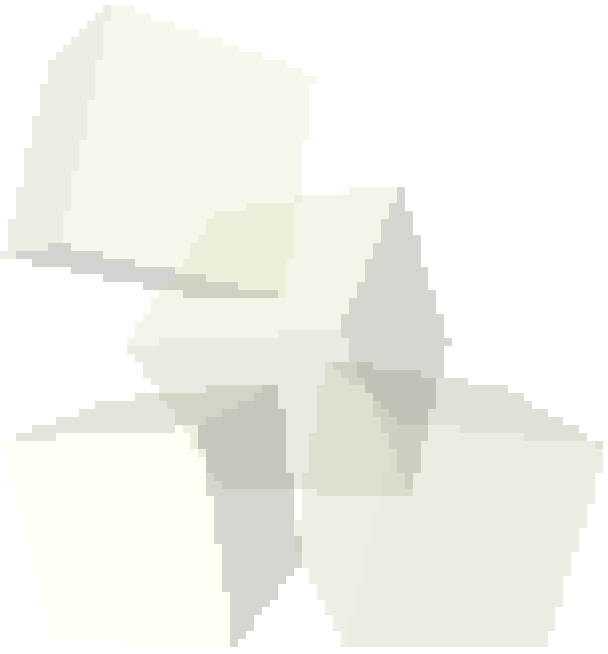
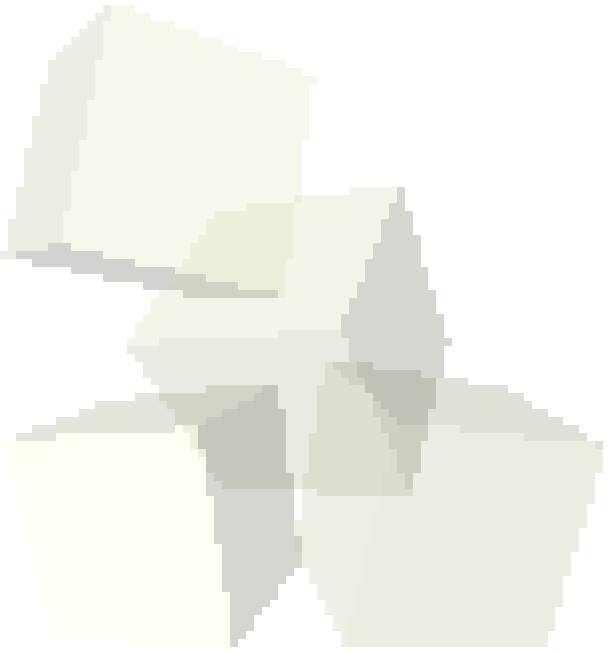
Generics, Enums, and Inner Classes

1-28-2009



Opening Discussion

- Let's look at solutions to the interclass problem.
- Minute Essay Responses
  - Scribes too slow.
  - Can you inherit a private class?
- Do you have any questions about the reading?
- Do you have any questions about the assignment?

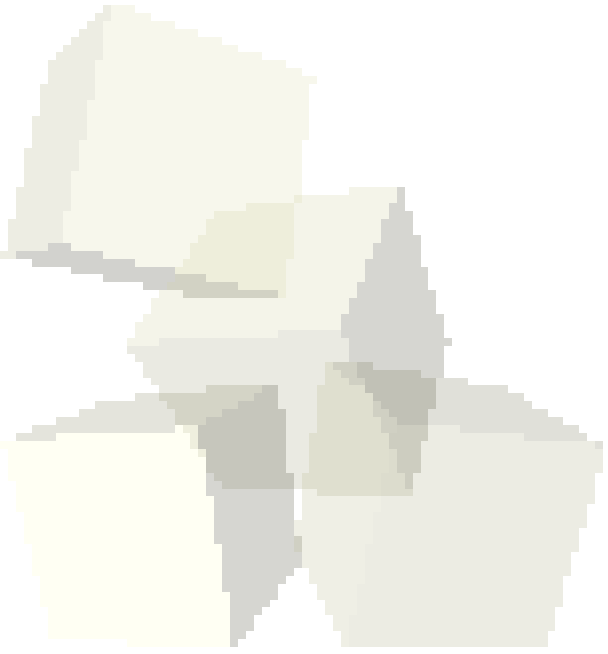


Abstract Keyword

- You can declare a method in a class that doesn't have an implementation. This method must be labeled as abstract.
- Any class that contains abstract methods must also be labeled as abstract.
- You use abstract methods when a superclass doesn't have a good default implementation so all subclasses should override it and give their implementations.

Inclusion Polymorphism/Project

- Inclusion polymorphism is what allows my code to work with what you are going to be writing.
- You are going to create subtypes of the types I have defined. My code works with the supertypes and through inclusion polymorphism it will work with your subtypes as well.

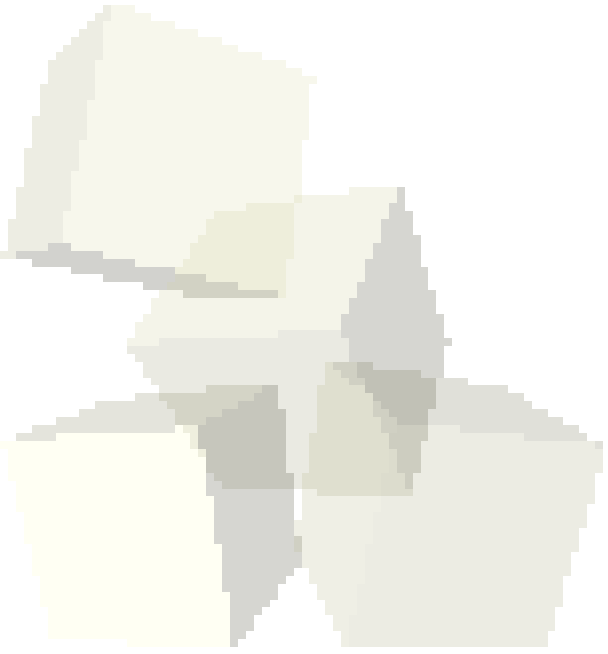


Restrictions in Java

- Java places some restrictions on inheritance to simplify the language. The main restriction is that you can only extend one class. Doing otherwise, multiple inheritance, tends to make things very complex.
- There are times when you want to have a class be a subtype of two different types though. To allow this Java has a construct called an interface. Interfaces have no data (they can have static data) and all methods in them are abstract. They only define what you can do with them, not how to do it. You can implement as many interfaces as you want.

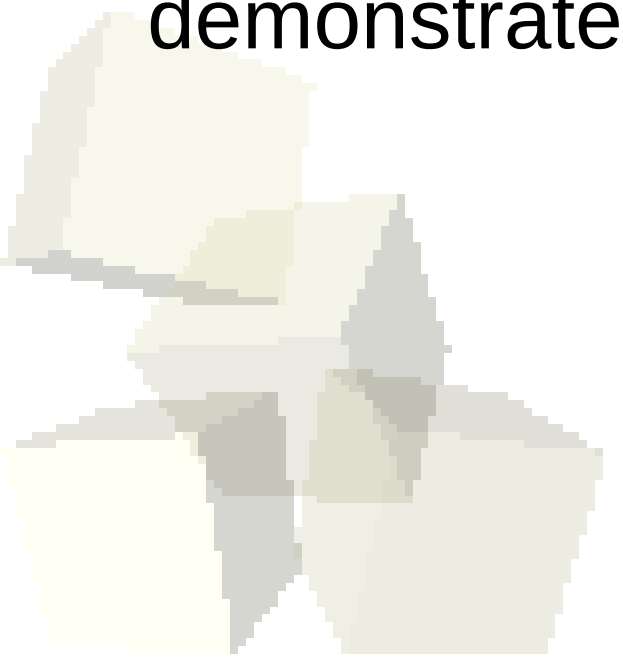
Interfaces Continued

- Java allows multiple inheritance from interfaces because they can never create ambiguity.
- Implementing an interface only provides subtyping, not code reuse.
- Subtypes of interfaces need to implement all of the methods of that interface or they will be abstract.



Inheritance Example

- Let's run through a new example of using inheritance.
- This example is pulled from functional languages. It want to abstract the concept of a function.
- Let's create an interface called MathFunction that can be used to map numbers to other numbers. Then we can write a few subclasses of it and demonstrate the use of these in our code.

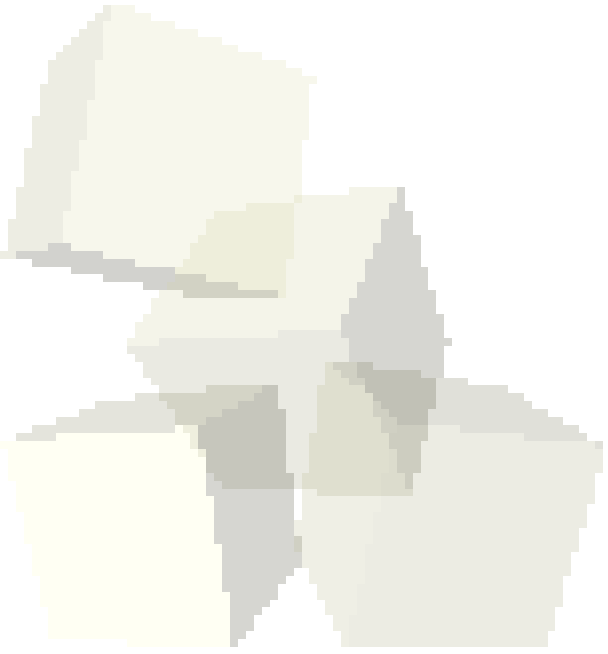


Inner Classes

- You can embed classes inside of code in Java. Not all classes have to be at the global scope. Classes that aren't at the global scope are called inner classes.
- There are three types of inner classes
  - Normal – Sits at the class scope. Looks normal.
  - Local – Sits in a method. Looks normal. Rarely used.
  - Anonymous – Sits in a method and has a syntax that doesn't name the class. Class must inherit from another class/interface.
- Each type can either be static or not. Things should be static by default. Non-static knows about the object that created it.

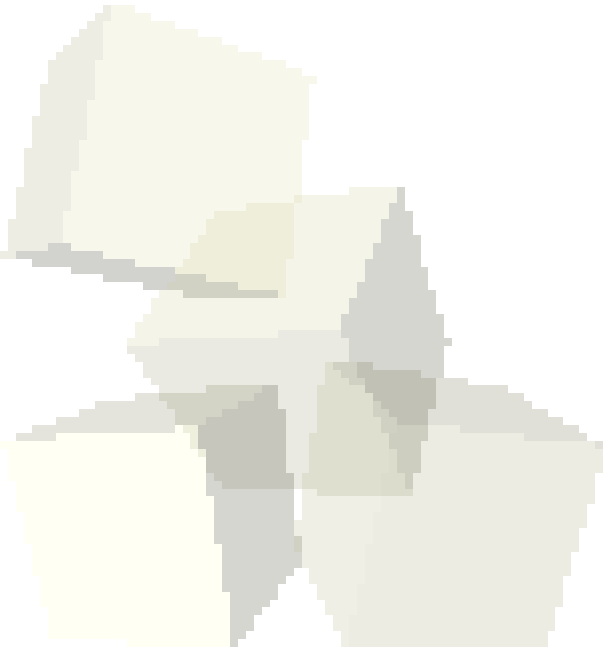
More Function Fun

- It turns out that our little MathFunction interface makes a perfect supertype for creating inner classes.
- We can demonstrate this by thinking of how we can create functions from other functions.
- Let's come up with a few of those and write code to do them.



More General Functions

- The MathFunctions can be fun, but they are a bit limited because they can only work with doubles.
- How would we write code that can do more general mapping? I want code that basically works with ANY type.
- Let's create an interface called GeneralFunction to represent this.



Java 5.0 Features

- Java 5.0 adds quite a few features to the Java language that make it easier to write real code in Java. Most of class today we will focus on two big ones, generics and enums. There are a few others worth mentioning.
- Autoboxing – primitive types are not objects, but sometimes you need to treat them that way. This automates the process.
- Varargs – Java now has syntax for passing a variable number of arguments.
- Foreach loop – A handy shortcut for when you want to do something with all elements of a container.

Generics

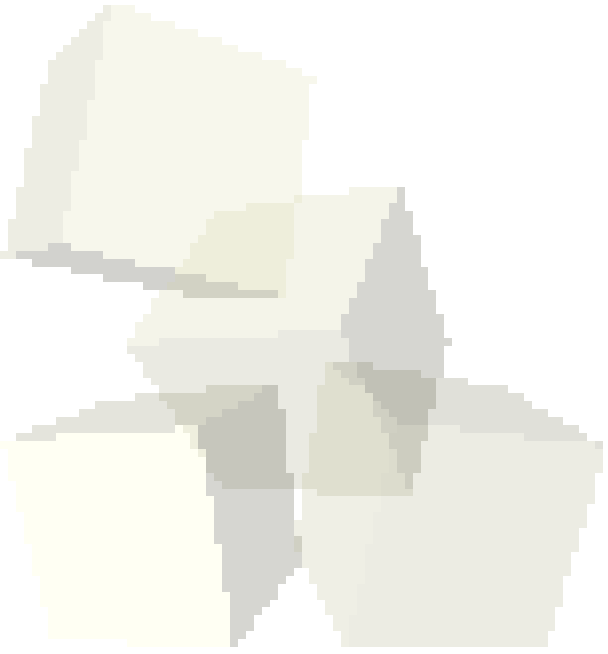
- The most significant feature added to Java 5.0 was that of generics. Generics provide a form of parametric polymorphism, typically for code that can take any type, but might be limited to a specific type for one instance.
- The most common use of this is for containers. Container classes typically should be able to hold anything, but any one container is generally intended to hold only one type.
- In practice, generics give you extra type safety and prevent you from doing a lot of type casts.

Making GeneralFunction Typesafe

- The problem with using the Object type for general polymorphism is that it many different type checks have to be done at runtime and you lose static type safety.
- Using generics we can take our general function interface and make it static typesafe.
- How can you combine these more general types of functions?

Enums

- C had enums. What were they supposed to do? What was the problem with them?
- Java includes enums as well. They serve the same goals, but lack the pitfalls.
- Java enum syntax can get quite complex, but the basic form is simple and very similar to C.



Code

- Generics play a big role in the project and appear in the first assignment so I want to show you a bit of the syntax.
- I'm going to write some code that uses generics so show the syntax. I'd rather you not try to follow along with this coding at the end of class and instead just try to pay close attention to what I'm doing, the logic behind it, and the syntax required to make it work. You can see the code on the web or even pull it down so you can play with it later.

Minute Essay

- What is the purpose of generics?
- Next class you will take the first quiz.
- Interclass Problem – Take the general function code that was written in class today and extend it. Write three different new functions. At least one must be an anonymous inner class. Prove they all work with test code.