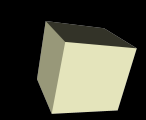
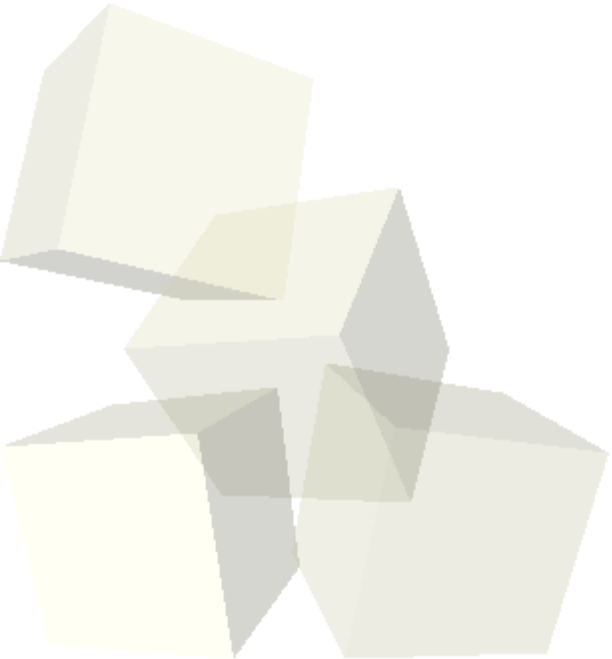


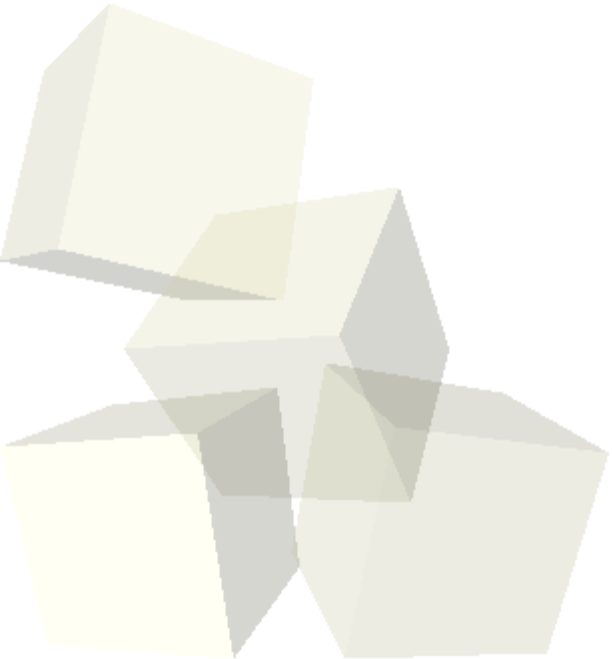
Java Basics

1/23/2007



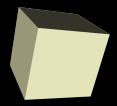
Opening Discussion

- What did we talk about last class?
- Do you have any questions about the reading?
- Do you have any questions about the assignment? Let's go over the requirements for this first design/assignment and what I'm expecting from you.



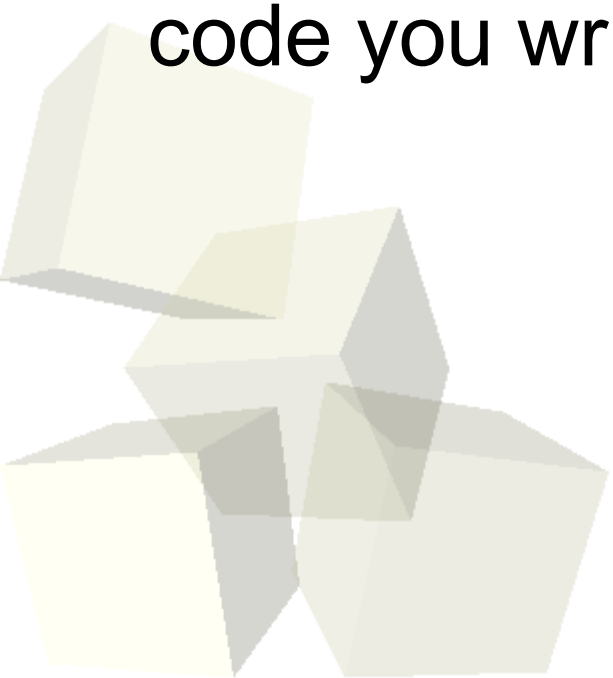
Documentation Comments

- In Java, comments that start with /** are documentation comments. These comments are used by javadoc to produce HTML documentation.
- These comments should go above all classes and methods, especially public ones. Inside the comment you start with a summary sentence then have a paragraph describing the class or method. After that can come certain “tags” that begin with @.



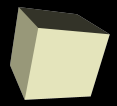
Code

- We want to continue our bank example that we worked on last time in two ways.
- First we want to add customer information. Instead of adding that straight to the account, we should create a Customer class and have the account reference it.
- Put in proper documentation comments on the code you write today.



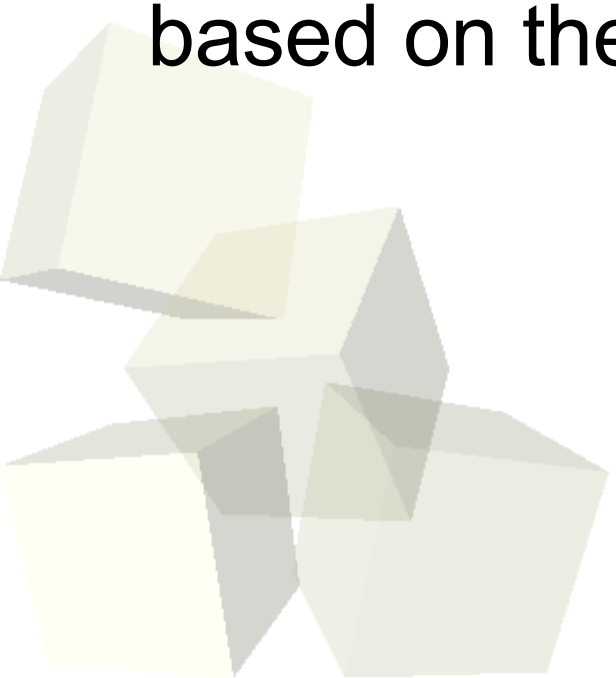
Java References vs. Pointers

- In Java when you declare an object you are really declaring a reference to an object. This is like a pointer but you can't do pointer arithmetic. To get a real object you use the new operator. New is like malloc and returns a heap object.
- All objects are gotten with new so all objects exist on the heap.
- When you call new it invokes a constructor.
- null is a universal symbol for references that don't point to anything.



Immutable Objects

- In your reading you have inevitably come across the term immutable. What does this mean?
- What are the advantages and pitfalls of immutability?
- How can you write code that takes advantage of immutability?
- The entire paradigm of functional languages is based on the idea that data is immutable.



Types and Polymorphism

- What are types in programming languages? What are some examples of types you are used to?
- Polymorphism literally means many shapes. In a programming context it means many types. Polymorphic code is code that can work with more than one input type.
- Universal polymorphism allows an infinite number of types. Ad hoc allows a finite number. Obviously the former is much more powerful and that is the type of polymorphism we will normally care about.
- Could you write polymorphic code in C?

Inheritance

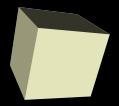
- The primary way we get polymorphism in Java is through inheritance.
- We can specify that one class inherits from another class with the extends keyword in Java.
- Many class based object oriented languages include inheritance. It is a construct in languages that models the “is-a” relationship. You should only use inheritance when this relationship applies. Even when it does apply it isn't always the the right thing to do.

Two Sides of Inheritance

- Inheritance provides two functions.
- The original motivation for inheritance, and the root of the term, is that a subclass implicitly gets a copy of everything in the class that it is inheriting from. This means it has all data and functions. It can't directly access the things that are private.
- Inheritance also provides subtyping. If class B inherits from A, then any code that uses A will work with an object of type B. This is how we get our polymorphism in Java. We write code that works with supertypes and it automatically works with subtypes. This type of polymorphism is called inclusion polymorphism.

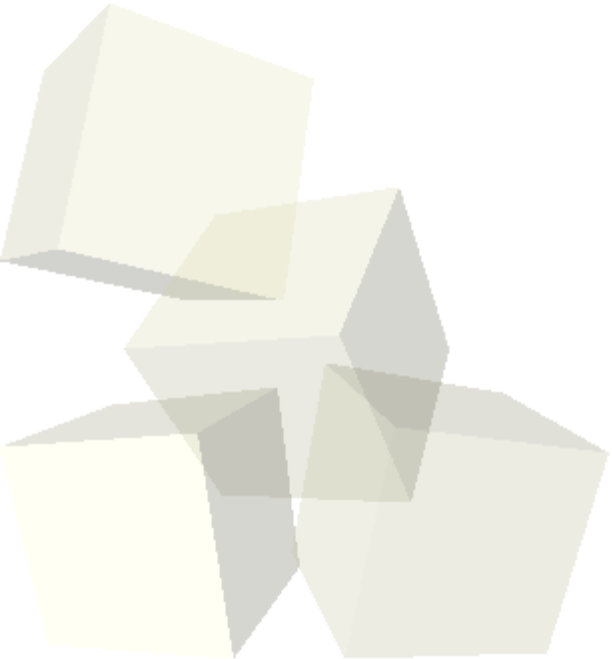
Restrictions in Java

- 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.



Code

- Now I want you to watch me construct some code that uses inheritance and polymorphism.
- The reading runs you through the classic example of a shape so I'm going to do something a bit different. I want to use the example of a simple function of one variable as our supertype and then create subtypes for specific types of functions.



Minute Essay

- How do you feel the format of a mini-lecture, followed by readings, followed by applying the knowledge is working? What could be done to improve it? Are you able to get through the readings with a reasonable comprehension of the material?
- Remember to submit the design for the first assignment, which includes your description of your game, by midnight tonight and send me an e-mail that includes your username when it is done. Point your browser to the design to check that everything worked correctly.