Inheritance in Java

9-7-2004

Opening Discussion

- What did we talk about last class?
- How many of you have looked in the book at the chapters I have asked you to read? How are you coming on your assignments? Remember that designs are due by midnight tonight and code is due on Thursday.
- Read project description.

Immutability

- Immutability can be very significant in how it impacts your programming. In fact, the entire paradigm of functional programming is based on immutability.
- Immutable objects are good because they can prevent bugs, especially in Java where all object variables are references. They are bad because they can produce extra allocations though the safety can reduce allocation.
- This isn't the same as const in C/C++, nor is final the same as const.

Passing Variables

- In C you could choose to pass by value or reference. In Java everything is basically passed by value, but all the objects are reference variables so you are passing a reference for them.
- You can't have multiple primitive returns. Instead, you should return a class that stores the multiple values you want to return.

Inheritance

- The Java model of programming makes extensive use of Inheritance, more than any other language I know of.
- Normal inheritance plays two roles in programming.
  - When class B inherits from class A, it "reuses" all the non-private methods and members of class A.
  - B also becomes a subtype of A.

Inheritance Hierarchies

- The standard way of drawing out inheritance is through a tree-like hierarchy.
- In UML the arrows point from the subclass to the superclass. This is because the superclass doesn’t generally know of all of its subclasses but the subclasses know of the superclass.

Inheritance for Code Reuse

- The first side effect of inheritance is gaining "copies of" non-private members.
- This means that if A had a public method foo() then B will also have a public method foo().
- In the assignment I mentioned that MainFrame inherits from javax.swing.Jframe and gets the show() method from it.

Virtual Functions

- One of the powers of Java is that you don't always have to use the methods defined by the superclass. You can override them in the subclass.
- Methods that can be overridden are called virtual methods. By default all methods in Java are virtual.
- A method invocation uses the definition "closest" to the actual class.

Final Keyword

- If you have a method that you don't want to ever be overridden, you can declare it as final.
- You can also declare an entire class to be final in which case no subclasses can ever be written to inherit from it.
- The final keyword is greatly underused in Java. It requires thought, but should be used more.

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 functions when a superclass doesn't have a good default implementation so all subclasses should override it and give their implementations

Inheritance for Subtyping

- Inheritance also provides subtyping. This is in part because the subclass has all the public methods and members of the superclass.
- Formally, when we say that B is a subtype of A, what we are saying is that any place in the code where an A is expected, a B can be used, or a B can always take the place of an A.

Inclusion Polymorphism

- This ability to substitute subtypes in place of supertypes is what leads to inclusion polymorphism.
- Inclusion polymorphism is a form of "universal polymorphism" because there are an infinite number of possible subclasses for any given class (assuming it isn't final).

Inclusion Polymorphism in the 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.

Single Inheritance of Classes

- Java only allows single inheritance of classes. That is to say that a class can only inherit from one superclass.
- This greatly simplifies code by reducing ambiguity. C++ has multiple inheritance which causes one to frequently need to specify which superclass of a given class a method should be called through.

Interfaces

- Of course, C++ has multiple inheritance for a reason, there are times when you want one type to be a subtype of several supertypes.
- To deal with this Java has interfaces. An interface is much like a class, but contains only method signatures (all methods are abstract). They have no implementations and no member data except constants.

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.

The Project

- Now that you know a bit more about Java, let's look real quick at the project framework and some more specifics about what you are going to be doing during the semester.

Let's Write Code

- Now we will use the rest of the time to write some code in Together that demonstrates a bit more about Java and inheritance as well as Together and what you can do in it. We'll also talk about the running example for the semester.

Minute Essay

- Inheritance is a very powerful tool, but it does have pitfalls. Can you think of what some of the problems might be with using inheritance? If not don't worry, they can be subtle so then write any questions you have about inheritance or how it is done in Java.
- Remember that your assignment #1 design is due today and the code is due on Thursday.