More Inheritance, Polymorphism, and Java Stuff

1-27-2005

Opening Discussion

- What did we talk about last class?
- Remember that the code for assignment #1 should be submitted to me today by midnight. Does anyone have any questions about the assignment?
- Does anyone want to show a problem?
- Submitting assignments.

Minute Essay Responses

- Why does one abstract method force a whole class to be abstract?
- Code examples and code for the project. I do not show code directly related to the project in class.
- General confusion and not understanding code in Java. The method is you create objects and then invoke methods that make objects interact.

Difficulties with Inheritance

- One significant problem can be frailty. You have to think about the public interfaces of base classes very carefully because when you have many subclasses it is almost impossible to change them. Also worry about public methods that call other public methods.
- A subclass that doesn't implement everything probably shouldn't be a subclass.
- Finding the methods in superclasses in deep hierarchies. They don't show in UML.

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.

Inclusion Polymorphism

- The definition of subtyping states that when we have a subtype object, we can use it in a place where we expect a supertype object.
- This implicitly gives us polymorphism because we can write a function that works with the supertype and all subtypes should automatically work as well.

Why It Works

- The reason this happens is because the subtypes "inherit" the full public interface of the supertypes and possibly other information/code as well. The code using an object can only use the public interface which all subtypes share with the supertype.
- Inclusion polymorphism works well in Java because all objects are stored as references and all methods are virtual.

Constructors

- We have seen and written constructors, but we should be more formal about them.
- A constructor is a method with no return type that has the same name as the class. As their name implies, they are used to construct variables and some constructor is called any time new is invoked.
- They can be overloaded with different arguments.

The super Keyword

- Sometimes you want to be able to access methods or constructors from the superclass of a given class. In Java this is done with the super keyword.
  - For constructors the first line of a constructor can be super(arg1,arg2,...); to call the constructor of the superclass that takes the given argument list.
  - For other methods, using super.method(...) will call that method of the superclass.

Inner Classes

- Starting with version 1.1, Java introduced inner classes. The simplistic view of inner classes is that they are classes inside of other classes.
- The full reality is that inner classes in Java have more complexity because they have access the the outer class. Also, unless you state otherwise, inner classes keep track of the instance of the "outer class" that creates them.

Static Inner Classes

- By default, you should make your inner classes static because they will have a lower overhead.
- The instances of this inner class are associated with the class as a whole.
- They have access to all static methods and members of the outer class, even if they are private. If given an instance of the outer class they can call private methods on it.

Non-static Inner Classes

- If an inner class is not declared static, it will get a reference to the instance of the outer class in which it is created.
- This gives it access to all methods and members of that instance. The methods of the inner class can access the private data of the outer class.
- This adds some overhead, but can be very handy at times.

Anonymous Inner Classes

- Java has another type of inner class, the anonymous inner class.
- As the name implies, these classes have no names. Instead, they have to be a subtype of some class or interface. They can then be used as an instance of the supertype.
- They allow you to create a class "inline", in the code for a method.

More on Anonymous ICs

- They are static or not depending on the type of method they are created in.
- They have access to all the things the named inner classes of their type would have, plus the final variables in the method in which they are declared.
- These are used extensively for event handling in Java GUIs.

Let’s write some code

- Now let’s write some code that demonstrates all the majors syntactic points of Java that we have talked about and displays inheritance.

Minute Essay

- Next class is on string processing. This means that we are going to move away from talking about the nature of the language itself. What questions do you still have about the Java language? Do you feel comfortable trying to code assignment #2 and if not what would help?
- Quiz #1 is on Tuesday.