Objects and Classes

1/22/2008

Opening Discussion

- Do you have any questions about the reading?
- Have you thought of any ideas for your games?

Old Games

- Let's look at some of the games that students have written in the past to give you some ideas of what you can create.

Objects and Classes

- From your reading and the last class you should have a general idea of what object orientation is.
- What is encapsulation?
- What is the distinction between an object and a class?
- What are the different visibility levels you can use in Java?
- How does visibility of members impact encapsulation?

Writing Java

- In Java all code goes inside of classes. Methods in a class have innate access to the data in that class.
- Java is not 100% object-oriented. Primitive types in Java (int, double, etc.) are not objects, they have no members. This decision was made for efficiency.
- Public members of a class define a “public interface”. These are the things that are known and accessible outside the class. They are hard to change without breaking other code. Data should never be public.

ACMEBank

- Now let's do an example. This is a rather standard example of a bank account. We will enhance it a bit to help demonstrate some features of Java.
- This example lets us see a scope for variables that didn't exist in C. The member data exist in each object and all methods invoked on that object can access it. We see that with the balance on the account.

Primitive Types in Java

- Java is not purely object-oriented because it does have primitive types. These types are boolean, char, byte, short, int, long, float, and double.
- Note that booleans and chars are NOT ints in Java (though you can cast chars to ints). This is significant because the statement **`if(v=3)`** does not compile. This helps cut down on bugs but might seem restrictive in some cases.

Primitives as Classes

- When you need to represent a primitive type as a class there are some classes in java.lang that can help.
- They are classes like Integer and Double that are basically wrapper classes.
- They do have some nice functionality in static methods as well like Integer.parseInt(String s).
- These classes are immutable.
- Autoboxing, adding in Java 5, will automate the use of wrappers, but you still need to understand what is happening.

Java References vs. Pointers

- In Java, when you declare an object variable 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 at some level (even if you don't call new yourself) so all objects exist on the heap.
- null is a universal symbol for references that don’t point to anything.

this and Using Members

- When you are writing a method of a class, it has direct access to the member data and methods of that class. You don’t have to use the ‘.’ notation.
- To be explicit, you can use the ‘this’ keyword which implies the object that the method was invoked on.

static

- The term static in the C-family languages implies something like “there is only one”. This is true in Java as well.
- A static member or method is associated with the class itself, not with an object/instance of that class.
- They can be reached or invoked without having an object of that class too.
- In our blueprint analogy, a static member is something written on the blueprint or associated with the factory, not something that is carried with every object made from the blueprint.

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 @.

No Preprocessor Directives

- You import so you don’t have to type in full package names. This looks similar to #include in use, but it is quite different.
- No #define in Java. For constants use static final variables. For macros just use functions.
- There is also no conditional compilation in Java so #ifdef, #ifndef, etc. don’t exit. Assert was added in 1.4 but we won't be using it.

Minute Essay

- What are you thinking of doing for your game?
- Name one way that you can think of that object-orientation can help you with programming.
- Keep reading through “From C to Java”.
- Interclass Problem – Write a Java program that uses a loop and an if statement. The whole thing can be put inside a single main method for this.