Java Basics

1-22-2004

Opening Discussion

- What did we talk about last class?
- What are the basic constructs in the programming languages you are familiar with? From reading or prior knowledge, how are those constructs different in Java?
- Minute essay comments

UML Class Diagrams

- UML stands for Unified Modeling Language. It is a formal graphic representation of software analysis and design. There are many types of UML diagrams, but we will mainly be looking at class diagrams.
- In this diagram classes are represented by boxes.
- Java also has things called interfaces that we will look at more a little later.

Inside a Class

- The box for each class is divided into three regions. The top one contains the class name and possibly some modifiers.
- The second region has attributes of the class. Typically these are specified with a visibility modifier followed by name:type.
- The third region holds operations (methods). They are displayed in a similar format but arguments can follow the name

Modifier Symbols

- Any of the attributes of operations can be modified with a symbol showing visibility.
  - + for public
  - # for protected
  - - for private
- Attributes can also be preceded by a '/' in some design tools to show that a given attribute is read only. Note that attributes aren't always member data.

Class Relationships

- In the diagram a subclass points to the class that it inherits from with an arrow with a closed head (more on what this means in two classes).
- An open header arrow in UML also gives us a way to denote when two classes are associated with one another is some way. Typically this implies that one class has attributes whose type is the other class.
- Associations can be labeled with a name telling what type of association it is.
- Example: Screen has grid of Blocks

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. 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 @.

Java as a Language

- In Java everything is in classes. There are no free standing functions or global variables. Also, the code for methods always goes inside the class declaration.
- All methods and members are individually given public or private specification. They can also be declared static or final. Final implies they can't be extended by inheritance. Final variables can't be changed after initialization.

Java Libraries

- One of the real powers of Java is it's libraries. These are grouped into packages. (You could specify packages for your code but we won't discuss that this semester.)
- For example, the class System is in the java.lang package. The full name of the class is java.lang.System.

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 has been added in 1.4.

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.

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.
- null is a universal symbol for references that don't point to anything.

More on Objects

- There is no operator overloading in Java. You write and call normal methods instead.
- Doing = or == with object references assigns or compares references. Think of them as pointers without the *.
- Use a copy constructor to copy and equals() for value comparison.
- No -> or * (deref) operator because things are implicitly dereferenced.

Garbage Collection

- Java has automatic garbage collection. There is no delete operator (no free).
- When you are done using an object and you have no more reachable references to it, the system can determine this and free up the memory for it.
- As a result, you can allocate objects much more freely because you don’t have to worry about freeing them.

Arrays

- Arrays in Java are actually objects. An array type is denoted by placing [] after the normal type. When you do a new you are allocating the array object. If the items in it are objects you still have to allocate the individual objects.
- This is really nice for inclusion polymorphism.
- You use them like you would in C/C++, though they do have a length member that you can access to find out how many elements are in an array.

Strings

- Like arrays, strings are a class in Java, called String. Literal strings in Java are objects of that class too.
- This class is in the java.lang package. Note that java.lang is the one package that is implicitly imported so you don’t have to use the import statement to refer to the classes in it directly.

Exceptions

- Java tries to prevent sloppy error handling by providing exceptions instead of requiring programmers to check return codes.
- try blocks surround code that might throw exceptions.
- catch blocks follow a try block and specify what type it is waiting for.
- finally catches everything and always happens.
- If an exception type can be thrown but isn't caught, it must be in the throws clause of that method. Not true for RuntimeExceptions.

Inheritance in Java

- Java allows inheritance for subtyping and reuse like C++ does.
- Java has single inheritance of classes, but it also has things called interfaces. You can have multiple inheritance of interfaces.
- An interface is like a class except that it only has method signatures. Classes that implement it have to define those methods or be abstract.

Java and the JVM

- Java code does not typically compile to executable format directly. Instead it compiles to bytecode that is stored in .class files. The bytecode was designed to be very small so that it can be easily moved across the net or put on small devices. It also allows code to run on many platforms without recompiles.
- In addition, this allows classes to be dynamically loaded.

Let's Code

- Now we will sit down and write a bit of code just to show you what it looks like. We'll do this until time comes for you to start minute essays.

Minute Essay

- Both the String class and the primitive wrapper classes in Java are immutable. This means that once created they can't be changed. How does using classes like this impact your programming? It can have both good and bad effects. What are those?