Even More Java

Java Packages

- What is a package?
- **Definition:** A *package* is a grouping of related types providing access protection and name space management. Note that *types* refers to classes, interfaces, enumerations, and annotation types. (Java's Definition)

Java Packages

- Access Protection means that using a package, as I showed last time, we can use different interfacing which can limit usage to within the package
- Name space management essentially means controlling scope. For example there is already a defined Vector class. There is no name conflict with your own because its in the Java.util package

Creating a Package

- Choose a name for your package
- Every source file must have `package <name>` as the first line of the file.
- Make sure you follow naming conventions on the next slide

Package Naming Conventions

- Packages have a naming convention
  - The name is lower case so it isn't confused with a type or interface
  - Often companies use the reverse of their domain name, ie com.example.orion
  - All provided packages start with java. Or javax.
  - Some cases the plain internet address cannot be used, usually we add an underscore; ie clipart-open.org -> org.clipart_open, free.fonts.int -> int_.fonts.free, poetry.7days.com -> com._7days.poetry

Using Package Members

- There are 3 ways to access a public package member from outside the package
  - Use the full qualified name, ie
    ```
    Java.util.Vector v = new Java.util.Vector()
    ```
  - Import the package member, *ie*
    ```
    import Java.util.Vector;
    Vector v = new Vector()
    ```
  - Import the package, ie
    ```
    import Java.util.*;
    Vector v = new Vector()
    ```
- The last option is usually discouraged for using a single class.

Apparent Hierarchies

- Sometimes it appears that some classes are contained in a package when in truth they aren’t; ie `import java.awt.*` will not import java.awt.color
- Therefore always look at your API

Name Ambiguities

- Should you ever import a package with an identical class name you must then qualify you classes to avoid ambiquity.
- For example, if on your last assignment you had imported java.util.Vector, every time you declared a vector you would have to qualify. `Vector v = new Vector()` is now ambiguous.

Static Import Statement

- If you want to import the static methods and fields of a class you can do this.
- For example, java.lang.Math contains a static PI field. If you wanted to import this simply type `import static java.lang.Math.PI` or as a group `import static java.lang.Math.*`
- Overusing static imports tends to make your code unreadable so use them sparingly

Managing Files

- When using packages, the class *`graphics.Rectangle`* should be in the directory *`/graphics/Rectangle`*
- Both your .java and .class files should be in this directory structure, but they don’t have to be the same one
- You can change your classpath

ClassPath

- Both the compiler and the JVM use your classpath
- The compiler will create directories based on packages for you
- You can change your classpath easily
  - To display the current CLASSPATH variable,
    - In Windows: C:\> set CLASSPATH
    - In Unix: % echo $CLASSPATH
  - To delete the current contents of the CLASSPATH variable
    - In Windows: C:\> set CLASSPATH=
    - In Unix: % unset CLASSPATH; export CLASSPATH
  - To set the CLASSPATH variable,
    - In Windows: C:\> set CLASSPATH=C:\users\george\java\classes
    - In Unix: % CLASSPATH=/home/george/java/classes; export CLASSPATH

Class Naming Conventions

- Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words- avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).

Variable Naming Conventions

- Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. ie., `float length, int height`
- Some conventions say all private variables should start with an underscore, others don't. ie., `private float _lentgth; private int _height`

Other Naming Conventions

- Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized. ie., *`run(), runFast(), getBackground()`*
- Constants should be declared in all capitals seperated by underscores. ie., *`static final int MIN_WIDTH = 4; static final int MAX_WIDTH = 999;`*