CMSC131

Java Packages

Java Packages

- There are different uses of the Java **`package`** system, some for organization, some for access protection, some which involve both.
- In most of the projects this semester we used packages to organize things.
- There can be sub-packages contained within a package.
- We can use **`import`** statements to gain access to public classes and interfaces in packages or can use fully qualified names to access specific things.

Why packages?

Some advantages of packages are:

- Two classes in different packages can have the same name without being a direct conflict and we could even have both used within the same project via qualified naming.
- Classes can be designed so that some of the classes in the same package can be accessed by others in the package but **not** by outsiders.

NOTE: There is a "default package" even if we don't define our own package.

Access / Visibility Summary

Modifier	Within Class	Within Package	Subclass (more to come)	Outsiders
`public`	YES	YES	YES	YES
`protected` (more to come)	YES	YES	YES	NO
`default` (no modifier)	YES	YES	NO	NO
`private`	YES	NO	NO	NO

The `import` statement

They go at the top of the **.java** files to tell Java "where" to look for classes and interfaces referred to in your code.

Allows us to avoid needing to use naming such as **`java.util.Date`** but rather be able to use **`Date`** by using import **`java.util.*;`**

No code is actually brought into the **.java** file with the import statement. This differs greatly from the C++ **`include`** statement you will see later.

The **package** statement

Your **.java** files will be in a folder named for the package, and each will have a **package** statement at the top of the form:

```
package packagename;
```

Sub-Packages

- A package might have a variety of classes within it but also contain sub-packages.
- These sub-packages might feel "right" to organize to be within the package but have their own self-contained purpose, thus a sub-package.
- For example, the **`java.util`** package is something that we've been using. By importing **`java.util.*`** we get direct access to the classes at that level.
- However, we do not get this access for the **`java.util.concurrent.*`** classes unless we import that as well.

.jar files

- Related to packages, once development is done, you might provide a package to someone by creating a **.jar** file containing the entire package's directory for easy portability.
- The **.jar** file needs to be placed somewhere that is listed within your CLASSPATH.
- There are other uses for **.jar** files. For example, you can create a "standalone" "executable" of your Java program.
- Structurally, a **.jar** file is essentially a **.zip** format archive file.