Packages

Where, oh where, is my little lost class file?

Administrivia

- Reminder: Midterm, Mar 9 (next Wed)
  - In class, closed book/notes, individual
- Office hours cancelled all next week
  - Contact me for special appointment
- I’m out of town from Mar 7 (after class) - Mar 20
  - Don’t depend on email contact w/ me then

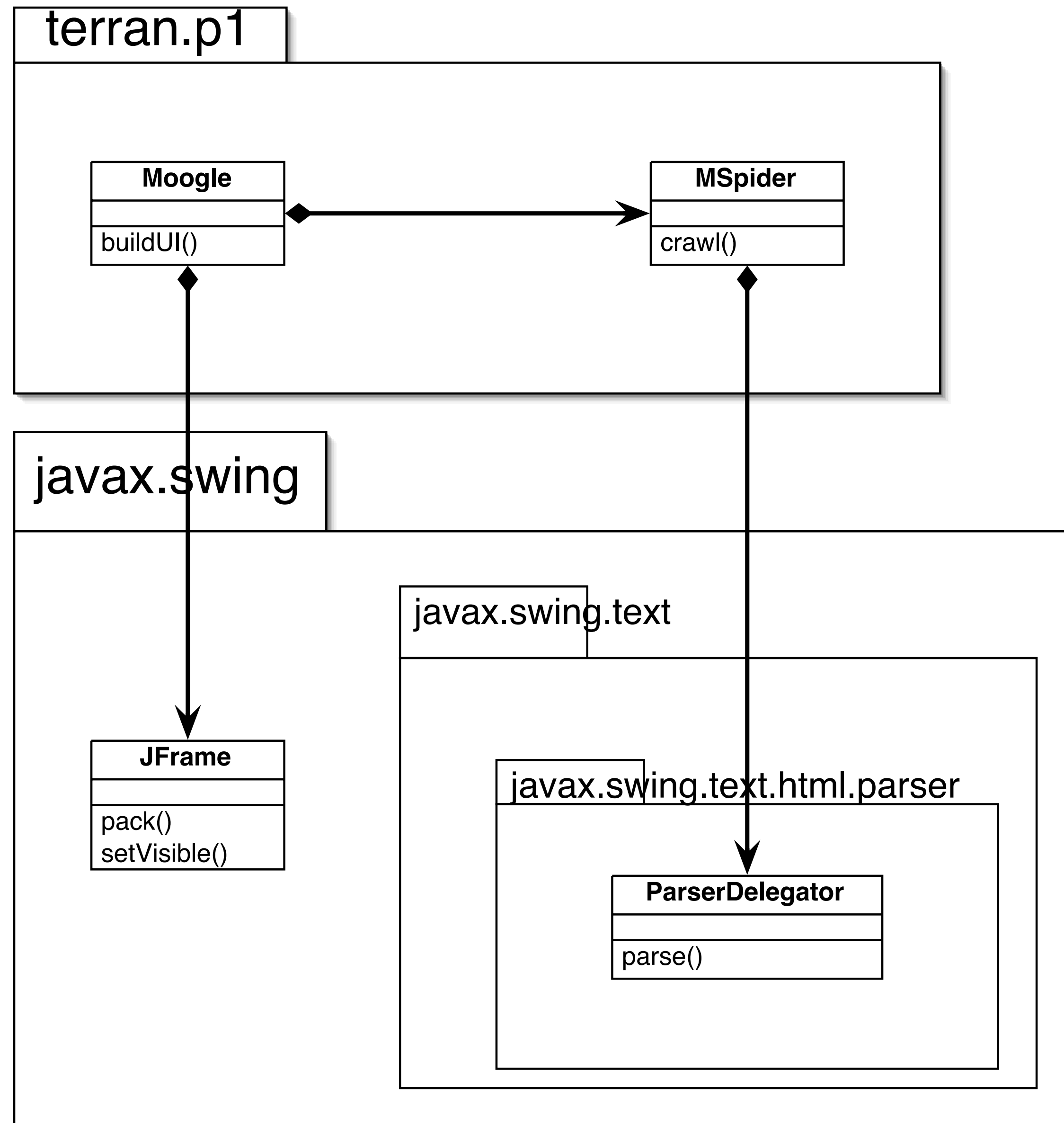
Testing Tips #1

- Divide and conquer
  - Test all sub-modules of code, as well as full system
  - Helps you isolate what is broken when something breaks
  - E.g., test REVERSE INDEX on its own, spider crawling code on its own, and combo of two
- Build test cases for each part
  - Test cases for individual modules == Unit testing
  - For overall system==Integration testing

Packages

- Every Java class exists in a "package"
- Essentially a namespace
  - Allows multiple developers to have classes w/ same base name
    - `java.util.Date` diff from `java.sql.Date`
  - "True" name of a class is `[packagename].[classname]`

The UML view



Package imports

- Could specify all classes w/ "fully qualified name"

```
java.util.Map m=new java.util.HashMap();
unm.cs351.s05.terran.p2.Parser p=
  new unm.cs351.s05.terran.p2.ZurkParser();
```

- Gets *very* tedious *very* quickly
- Java allows "import" statements
- Essentially just aliases short version to long version

```
import java.util.Map;
import java.util.HashMap;
Map m=new HashMap();
```

- Rewritten by javac during compile to
  - `java.util.Map m=new java.util.HashMap();`
- Stored that way in .class file (bytecode)

Importing like crazy

- Can also import everything in a package
  - `import java.util.*;`
- Opens up short form of everything in that package
- Convenient...
- But also sloppy
  - Clutters namespace
  - What if you want to refer to `java.util.Map` and `java.sql.Date` in same prog?
  - Run into trouble if you import `*` from both...

Nested... But not really

- Package names are "nested" by "`.`" characters
  - `javax.swing.text.html` is "inside" `javax.swing.text` is "inside" `javax.swing` ...
- Except that import `javax.swing.*` doesn't get `javax.swing.text.*`, `javax.swing.text.html.*`, etc.
- Dot notation is mostly just convenience for making very long package names

Picking your package

- Declaring that class is in a specific class is easy:

```
package my.package.name;
```

  - At top of file, before imports
- Sun suggests package name of form:
  - edu.unm.cs.[otherstuff]
  - I.e., your domain-name, run drawkcab
- At minimum: just ensure name uniqueness
  - Nobody should be likely to collide w/ your package name
- E.g., `unm.cs351.s05.lane.p2`

Finding the class

- Java class loader expects class file for `java.util.Date` to be
  - `[CLASSPATH elt]/java/util/Date.class`
- Can be another "but it's right there!" problem
- Java class loader has file system expectations built into it... (Ugh)
- Problem is that *source file* doesn't have to be in any particular place
  - `javac` will still dump it into the corresponding path
- The *top* of the package path has to be on `CLASSPATH`

Packages and access

- All members of a package can "see" all other members of a package for free
  - Don't need `import` statements
- Anything declared with no access modifier (`public`, `protected`, `private`) has *package access*
  - Everything in the package has access to it, but nothing outside the package does
  - Sort-of like "semi-global" stuff
  - Good for stuff shared among all members of a package
    - E.g., a static variable used in common by all members of a package