CS193j, Stanford
Summer, 2002-03

Handout #29
Manu Kumar

Advanced Java

A tour of a few advanced Java features added in Java 1.4...

Regular Expressions

Added in Java 1.4
http://developer.java.sun.com/developer/technicalArticles/releases/1.4regex/

Pattern

Represents a regular expression pattern
Supports Perl-style regular expressions: \w+ \s [^a-z0-9]* etc. etc.
Need to use double \\ in strings to get a single \: e.g. "\\s" translates to \s sent to the regular expression engine.

Matcher

Create a matcher out of a pattern and some text
The matcher can search for the pattern in the text
find() searches for an occurrence of the pattern, searching from a point after the previous find()
group() returns the matched text from the previous find
Matcher supports many other ways to look and iterate with the pattern on the text.

```
import java.util.regex.*;
...
// Extract email addrs from text
// email addr is @ surrounded by \w.-_
// We need to use double \\ in the " string, to put a single \ in the pattern
// (\w represents a 'word' character: a-zA-Z and _
String text = "blah blah, nick@cs, binky binky foo@bar.com; spam_me@foo.edu ";
String re = "[\\w\\.\\-]+\\@[\\w\\.\\-]+";
Pattern pattern = Pattern.compile(re);

// Create a matcher on the string
Matcher matcher = pattern.matcher(text);

// find() will iterate through matches in the text
while (matcher.find()) {
   // group() returns the most recently matched text
   System.out.println("email:" + matcher.group() );
}
/*
Output
 email:nick@cs
 email:foo@bar.com
 email:spam_me@foo.edu
*/
```

pattern.split()

Use a pattern to split up a string -- extract the strings separated by the pattern (like the split function in Perl)

```
      // Use split() to extract parts of a string
      String text2 = "Hello, what's with all the punctuation, and stuff here;   I 
want just the words.";
      // The pattern matches one or more adjacent whitespace or punctuation chars
      Pattern splitter = Pattern.compile("[\\s,.;]+");

      // Split() uses the pattern as a separator, returns all the other strings:
      // "Hello" "what's" "with" ...
      String[] words = splitter.split(text2);
      for (int i=0; i<words.length; i++) {
         System.out.println(words[i]);
      }
```

Static Pattern.match() convenient method

Static convenience method
Builds a pattern and matcher, and runs the matches() method against the given text.
Returns true if the _entire text_ matches the pattern.
Less efficient, since the pattern and matcher are built, used once, and discarded.
Handy for simple little cases where the client doesn't want the bother of creating the pattern and matcher objects. Good client oriented design -- the simple, common case is very easy to code.

```
boolean found = Pattern.matches("\\w+\\s\\w+", "hello there");
System.out.println(found); // true
```

Assert

Added in Java 1.4
Off by default at runtime
Assert code is essentially deleted at runtime by the JVM
Therefore, never include code that must execute in an assert (example below)
Turn on and off with the -ea/-da command line switches to java

```
class Assert {
   /*
    Use assertions to sprinkle code with tests of what should be true.
    The assertion throws an AssertionError exception if the test is false.
    -help document what you think is going on, say, at the top
      of each loop iteration
    -help find bugs more quickly in this code, and in client code that
      calls this code.

    The assert code may be deleted by the JVM at runtime, so do not put
    code that must execute in the assert. Assert should do read-only tests.

    To compile with asserts, use the '-source 1.4' to javac. If you compile this
    way, the code will only work on a 1.4 or later JVM.
    By default at runtime, assert is not enabled -- they are NOPs
    Turn asserts on with the -enableassertions switch to the 'java' command
    (-ea is the shorthand)
    java -ea // turns on asserts for the whole program
    java -ea MyClass // turns on asserts for just that class
    java -ea -da MyClass // turn on asserts, but turn them off for MyClass
   */
   public static void main(String[] args) {

      int len=1;

      // assert a condition that should be true
      assert len<100;

      // Can include a : <string> after the assert that goes in the error printout
      assert len<1 : "len=" + len;

      /*
      Output:
      Exception in thread "main" java.lang.AssertionError: len=1
        at Assert.main(Advanced.java:80)
        */

        // Suppose your code calls foo(), and it returns 0 on success

        // Never do this:
        //   assert foo()==0;
        // Do it this way, so it still works if the assert is disabled
        //   int result = foo();
        //   assert result==0;
   }

}
```