```
 1  /*----------------------------------------------------------------------
 2  This file has been modified for use in CISC 323.  Some code has
 3  been removed to reduce the length of the inspection task.  Defects
 4  may have been added.  Please do not assume all defects found are
 5  the fault of the original author.  The choice of this class for an
 6  inspection exercise does not imply any value judgement about the
 7  quality of the original code.  It was chosen simply because it is
 8  real-world code and freely available under the GNU license.
 9  ----------------------------------------------------------------------*/
10
11  // $Id: Matcher.java,v 1.2 2001/12/07 11:41:24 ramsdell Exp $
12  // A string matcher.
13
14  /*
15   * Copyright 1997 by John D. Ramsdell
16   *
17   * This program is free software; you can redistribute it and/or
18   * modify it under the terms of the GNU Lesser General Public License
19   * as published by the Free Software Foundation; either version 2
20   * of the License, or (at your option) any later version.
21   *
22   * This program is distributed in the hope that it will be useful,
23   * but WITHOUT ANY WARRANTY; without even the implied warranty of
24   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
25   * GNU Lesser General Public License for more details.
26   *
27   * You should have received a copy of the GNU Lesser General Public License
28   * along with this program; if not, write to the Free Software
29   * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
30   */
31
32  /**
33   * A string matcher.
34   * This class implements pattern matches of the form:
35   * pattern = prefix + "%" + suffix.  If the pattern does contain '%',
36   * the percent sign is added to the beginning of the pattern.
37   * A pattern with more than one percent sign is erroneous.
38   * @version November 1997
39   * @author John D. Ramsdell
40   */
41
42  final class Matcher
43  {
44    // The wild card for string patterns.
45    private final static char wildCard = '%';
46
47    private String prefix = "";
48    private String suffix = null;
49    private int prefixLength;
50    private int suffixLength;
51
52    /**
53     * Create a matcher associated with a given pattern.
54     * When given a bad pattern, a matcher that matches nothing is created.
55     */
56    Matcher(String pattern) {
57      if (isPatternOkay(pattern)) {
58        suffix = pattern;
59        prefixLength = pattern.indexOf(wildCard);
60        if (prefixLength >= 0) {
61          prefix = pattern.substring(0, prefixLength);
62          suffix = pattern.substring(prefixLength + 1);
63        }
```

```
64          else
65            prefixLength = 0;
66          suffixLength = suffix.length();
67        }
68      }
69
70      /**
71       * Match a string against the pattern.
72       * @return null if there is no match, otherwise a substitution
73       *         that makes the pattern equal to the string
74       */
75      String match(String string) {
76        if (suffix != null
77            && string.length() >= prefixLength + suffixLength
78            && string.startsWith(prefix)
79            && string.endsWith(suffix)) {
80          string = string.substring(0, string.length() - suffixLength);
81          string = string.substring(prefixLength);
82          return string;
83        }
84        else
85          return null;
86      }
87
88      /**
89       * Substitute the wild cards in the replacement with the match.
90       */
91      static String subst(String match, String replacement) {
92        if (match == null)
93          return replacement;
94        int i = replacement.indexOf(wildCard);
95        if (i < 0)
96          return replacement;
97        StringBuffer sb = new StringBuffer();
98        do {
99          sb.append(replacement.substring(0, i));
100         sb.append(match);
101         replacement = replacement.substring(i + 1);
102         i = replacement.indexOf(wildCard);
103       }
104       while (i >= 0);
105       sb.append(replacement);
106       return sb.toString();
107     }
108
109     /**
110      * Test if a string has at least one wild card.
111      */
112     static boolean isPattern(String string) {
113       return string.indexOf(wildCard) >= 0;
114     }
115
116     /**
117      * Test if a pattern has at most one wild card.
118      */
119     static boolean isPatternOkay(String pattern) {
120       int wildCards = 0;
121       for (int i = 0; i < pattern.length(); i++)
122         if (pattern.charAt(i) == wildCard)
123           wildCards++;
124         return wildCards <= 1;
125     }
126   }
```