```
/*

   Porter stemmer in Java. The original paper is in

       Porter, 1980, An algorithm for suffix stripping, Program, Vol. 14,
       no. 3, pp 130-137,

   See also http://www.tartarus.org/~martin/PorterStemmer

   History:

   Release 1

   Bug 1 (reported by Gonzalo Parra 16/10/99) fixed as marked below.
   The words 'aed', 'eed', 'oed' leave k at 'a' for step 3, and b[k-1]
   is then out outside the bounds of b.

   Release 2

   Similarly,

   Bug 2 (reported by Steve Dyrdahl 22/2/00) fixed as marked below.
   'ion' by itself leaves j = -1 in the test for 'ion' in step 5, and
   b[j] is then outside the bounds of b.

   Release 3

   Considerably revised 4/9/00 in the light of many helpful suggestions
   from Brian Goetz of Quiotix Corporation (brian@quiotix.com).

   Release 4

*/

import java.io.*;

/**
  * Stemmer, implementing the Porter Stemming Algorithm
  *
  * The Stemmer class transforms a word into its root form.  The input
  * word can be provided a character at time (by calling add()), or at once
  * by calling one of the various stem(something) methods.
  */

class Stemmer
{  private char[] b;
   private int i,     /* offset into b */
               i_end, /* offset to end of stemmed word */
               j, k;
   private static final int INC = 50;
                     /* unit of size whereby b is increased */
   public Stemmer()
```

```
{  b = new char[INC];
   i = 0;
   i_end = 0;
}

/**
 * Add a character to the word being stemmed.  When you are finished
 * adding characters, you can call stem(void) to stem the word.
 */

public void add(char ch)
{  if (i == b.length)
   {  char[] new_b = new char[i+INC];
      for (int c = 0; c < i; c++) new_b[c] = b[c];
      b = new_b;
   }
   b[i++] = ch;
}


/** Adds wLen characters to the word being stemmed contained in a portion
 * of a char[] array. This is like repeated calls of add(char ch), but
 * faster.
 */

public void add(char[] w, int wLen)
{  if (i+wLen >= b.length)
   {  char[] new_b = new char[i+wLen+INC];
      for (int c = 0; c < i; c++) new_b[c] = b[c];
      b = new_b;
   }
   for (int c = 0; c < wLen; c++) b[i++] = w[c];
}

/**
 * After a word has been stemmed, it can be retrieved by toString(),
 * or a reference to the internal buffer can be retrieved by getResultBuffer
 * and getResultLength (which is generally more efficient.)
 */
public String toString() { return new String(b,0,i_end); }

/**
 * Returns the length of the word resulting from the stemming process.
 */
public int getResultLength() { return i_end; }

/**
 * Returns a reference to a character buffer containing the results of
 * the stemming process.  You also need to consult getResultLength()
 * to determine the length of the result.
 */
public char[] getResultBuffer() { return b; }

/* cons(i) is true <=> b[i] is a consonant. */
```

```
private final boolean cons(int i)
{  switch (b[i])
   {  case 'a': case 'e': case 'i': case 'o': case 'u': return false;
      case 'y': return (i==0) ? true : !cons(i-1);
      default: return true;
   }
}

/* m() measures the number of consonant sequences between 0 and j. if c is
   a consonant sequence and v a vowel sequence, and <..> indicates arbitrary
   presence,

      <c><v>       gives 0
      <c>vc<v>     gives 1
      <c>vcvc<v>   gives 2
      <c>vcvcvc<v> gives 3
      ....
*/

private final int m()
{  int n = 0;
   int i = 0;
   while(true)
   {  if (i > j) return n;
      if (! cons(i)) break; i++;
   }
   i++;
   while(true)
   {  while(true)
      {  if (i > j) return n;
            if (cons(i)) break;
            i++;
      }
      i++;
      n++;
      while(true)
      {  if (i > j) return n;
         if (! cons(i)) break;
         i++;
      }
      i++;
    }
}

/* vowelinstem() is true <=> 0,...j contains a vowel */

private final boolean vowelinstem()
{  int i; for (i = 0; i <= j; i++) if (! cons(i)) return true;
   return false;
}

/* doublec(j) is true <=> j,(j-1) contain a double consonant. */
```

```
private final boolean doublec(int j)
{  if (j < 1) return false;
   if (b[j] != b[j-1]) return false;
   return cons(j);
}

/* cvc(i) is true <=> i-2,i-1,i has the form consonant - vowel - consonant
   and also if the second c is not w,x or y. this is used when trying to
   restore an e at the end of a short word. e.g.

      cav(e), lov(e), hop(e), crim(e), but
      snow, box, tray.

*/

private final boolean cvc(int i)
{  if (i < 2 || !cons(i) || cons(i-1) || !cons(i-2)) return false;
   {  int ch = b[i];
      if (ch == 'w' || ch == 'x' || ch == 'y') return false;
   }
   return true;
}

private final boolean ends(String s)
{  int l = s.length();
   int o = k-l+1;
   if (o < 0) return false;
   for (int i = 0; i < l; i++) if (b[o+i] != s.charAt(i)) return false;
   j = k-l;
   return true;
}

/* setto(s) sets (j+1),...k to the characters in the string s, readjusting
   k. */

private final void setto(String s)
{  int l = s.length();
   int o = j+1;
   for (int i = 0; i < l; i++) b[o+i] = s.charAt(i);
   k = j+l;
}

/* r(s) is used further down. */

private final void r(String s) { if (m() > 0) setto(s); }

/* step1() gets rid of plurals and -ed or -ing. e.g.

       caresses  ->  caress
       ponies    ->  poni
       ties      ->  ti
       caress    ->  caress
       cats      ->  cat
```

```
        feed      ->  feed
        agreed    ->  agree
        disabled  ->  disable

        matting   ->  mat
        mating    ->  mate
        meeting   ->  meet
        milling   ->  mill
        messing   ->  mess

        meetings  ->  meet

    */

    private final void step1()
    {  if (b[k] == 's')
       {  if (ends("sses")) k -= 2; else
          if (ends("ies")) setto("i"); else
          if (b[k-1] != 's') k--;
       }
       if (ends("eed")) { if (m() > 0) k--; } else
       if ((ends("ed") || ends("ing")) && vowelinstem())
       {  k = j;
          if (ends("at")) setto("ate"); else
          if (ends("bl")) setto("ble"); else
          if (ends("iz")) setto("ize"); else
          if (doublec(k))
          {  k--;
             {  int ch = b[k];
                if (ch == 'l' || ch == 's' || ch == 'z') k++;
             }
          }
          else if (m() == 1 && cvc(k)) setto("e");
      }
    }

    /* step2() turns terminal y to i when there is another vowel in the stem. */

    private final void step2() { if (ends("y") && vowelinstem()) b[k] = 'i'; }

    /* step3() maps double suffices to single ones. so -ization ( = -ize plus
       -ation) maps to -ize etc. note that the string before the suffix must give
       m() > 0. */

    private final void step3() { if (k == 0) return; /* For Bug 1 */ switch (b[k-1])
    {
        case 'a': if (ends("ational")) { r("ate"); break; }
                  if (ends("tional")) { r("tion"); break; }
                  break;
        case 'c': if (ends("enci")) { r("ence"); break; }
                  if (ends("anci")) { r("ance"); break; }
                  break;
        case 'e': if (ends("izer")) { r("ize"); break; }
                  break;
```

```
   case 'l': if (ends("bli")) { r("ble"); break; }
             if (ends("alli")) { r("al"); break; }
             if (ends("entli")) { r("ent"); break; }
             if (ends("eli")) { r("e"); break; }
             if (ends("ousli")) { r("ous"); break; }
             break;
   case 'o': if (ends("ization")) { r("ize"); break; }
             if (ends("ation")) { r("ate"); break; }
             if (ends("ator")) { r("ate"); break; }
             break;
   case 's': if (ends("alism")) { r("al"); break; }
             if (ends("iveness")) { r("ive"); break; }
             if (ends("fulness")) { r("ful"); break; }
             if (ends("ousness")) { r("ous"); break; }
             break;
   case 't': if (ends("aliti")) { r("al"); break; }
             if (ends("iviti")) { r("ive"); break; }
             if (ends("biliti")) { r("ble"); break; }
             break;
   case 'g': if (ends("logi")) { r("log"); break; }
} }

/* step4() deals with -ic-, -full, -ness etc. similar strategy to step3. */

private final void step4() { switch (b[k])
{
   case 'e': if (ends("icate")) { r("ic"); break; }
             if (ends("ative")) { r(""); break; }
             if (ends("alize")) { r("al"); break; }
             break;
   case 'i': if (ends("iciti")) { r("ic"); break; }
             break;
   case 'l': if (ends("ical")) { r("ic"); break; }
             if (ends("ful")) { r(""); break; }
             break;
   case 's': if (ends("ness")) { r(""); break; }
             break;
} }

/* step5() takes off -ant, -ence etc., in context <c>vcvc<v>. */

private final void step5()
{   if (k == 0) return; /* for Bug 1 */ switch (b[k-1])
   {  case 'a': if (ends("al")) break; return;
      case 'c': if (ends("ance")) break;
                if (ends("ence")) break; return;
      case 'e': if (ends("er")) break; return;
      case 'i': if (ends("ic")) break; return;
      case 'l': if (ends("able")) break;
                if (ends("ible")) break; return;
      case 'n': if (ends("ant")) break;
                if (ends("ement")) break;
                if (ends("ment")) break;
                /* element etc. not stripped before the m */
                if (ends("ent")) break; return;
      case 'o': if (ends("ion") && j >= 0 && (b[j] == 's' || b[j] == 't')) break;
                        /* j >= 0 fixes Bug 2 */
                if (ends("ou")) break; return;
                /* takes care of -ous */
      case 's': if (ends("ism")) break; return;
      case 't': if (ends("ate")) break;
                if (ends("iti")) break; return;
      case 'u': if (ends("ous")) break; return;
      case 'v': if (ends("ive")) break; return;
      case 'z': if (ends("ize")) break; return;
      default: return;
   }
   if (m() > 1) k = j;
}

/* step6() removes a final -e if m() > 1. */

private final void step6()
{  j = k;
   if (b[k] == 'e')
   {  int a = m();
      if (a > 1 || a == 1 && !cvc(k-1)) k--;
   }
   if (b[k] == 'l' && doublec(k) && m() > 1) k--;
}

/** Stem the word placed into the Stemmer buffer through calls to add().
 * Returns true if the stemming process resulted in a word different
 * from the input.  You can retrieve the result with
 * getResultLength()/getResultBuffer() or toString().
 */
public void stem()
{  k = i - 1;
   if (k > 1) { step1(); step2(); step3(); step4(); step5(); step6(); }
   i_end = k+1; i = 0;
}

/** Test program for demonstrating the Stemmer.  It reads text from a
 * a list of files, stems each word, and writes the result to standard
 * output. Note that the word stemmed is expected to be in lower case:
 * forcing lower case must be done outside the Stemmer class.
 * Usage: Stemmer file-name file-name ...
 */
public static void main(String[] args)
{
   char[] w = new char[501];
   Stemmer s = new Stemmer();
   for (int i = 0; i < args.length; i++)
   try
   {
      FileInputStream in = new FileInputStream(args[i]);

      try
```

```
         { while(true)

           {  int ch = in.read();
              if (Character.isLetter((char) ch))
              {
                 int j = 0;
                 while(true)
                 {  ch = Character.toLowerCase((char) ch);
                    w[j] = (char) ch;
                    if (j < 500) j++;
                    ch = in.read();
                    if (!Character.isLetter((char) ch))
                    {
                       /* to test add(char ch) */
                       for (int c = 0; c < j; c++) s.add(w[c]);

                       /* or, to test add(char[] w, int j) */
                       /* s.add(w, j); */

                       s.stem();
                       {  String u;

                          /* and now, to test toString() : */
                          u = s.toString();

                          /* to test getResultBuffer(), getResultLength() : */
                          /* u = new String(s.getResultBuffer(), 0, s.getResultLength()); */

                          System.out.print(u);
                       }
                       break;
                    }
                 }
              }
              if (ch < 0) break;
              System.out.print((char)ch);
           }
         }
         catch (IOException e)
         {  System.out.println("error reading " + args[i]);
            break;
         }
      }
      catch (FileNotFoundException e)
      {  System.out.println("file " + args[i] + " not found");
         break;
      }
   }
}
```