

```

1 // foj/8/terminal/Terminal.java
2 // (and terminal/Terminal.java)
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5
6 import java.io.*;
7
8 /**
9  * Terminal provides a user-friendly interface to the standard System
10 * input and output streams (in, out, and err).
11 * <p>
12 * A Terminal is an object. In general, one is expected to instantiate
13 * just one Terminal. Although one might instantiate several, all will
14 * share the same System streams.
15 * <p>
16 * A Terminal may either explicitly echo input, or not. Echoing input
17 * is useful, for example, when testing with I/O redirection.
18 * <p>
19 * Inspired by Cay Horstmann's Console Class.
20 */
21
22 public class Terminal
23 {
24     private boolean echo = false;
25     private static BufferedReader in =
26         new BufferedReader(new FileReader(FileDescriptor.in));
27
28
29     // Print a prompt to the console without a newline.
30
31     private void printPrompt( String prompt )
32     {
33         print( prompt );
34         System.out.flush();
35     }
36
37     /**
38      * Construct a Terminal that doesn't echo input.
39      */
40
41     public Terminal()
42     {
43         this( false );
44     }
45
46     /**
47      * Construct a Terminal.
48      *
49      * @param echo whether or not input should be echoed.
50      */
51
52     public Terminal( boolean echo )
53     {
54         this.echo = echo;
55     }
56

```

```

57
58     /**
59      * Read a line (terminated by a newline) from the Terminal.
60      * @param prompt output string to prompt for input.
61      * @return the string (without the newline character),
62      *         * null if eof.
63      */
64
65     public String readline( String prompt )
66     {
67         printPrompt(prompt);
68         try {
69             String line = in.readLine();
70             if (echo) {
71                 println(line);
72             }
73             return line;
74         }
75         catch (IOException e) {
76             return null;
77         }
78     }
79
80     /**
81      * Read a line (terminated by a newline) from the Terminal.
82      *
83      * @return the string (without the newline character).
84      */
85
86     public String readline()
87     {
88         return readline( "" );
89     }
90
91     // Read a line from the Terminal. An end of file,
92     // indicated by a null, raises a runtime exception.
93     // Used only internally.
94
95     private String readNonNullLine()
96     {
97         return readNonNullLine( "" );
98     }
99
100    // Read a line from the Terminal. An end of file,
101    // indicated by a null, raises a runtime exception.
102    // Used only internally.
103
104    private String readNonNullLine( String prompt )
105    {
106        String line = readline( prompt );
107        if (line == null) {
108            throw new RuntimeException( "End of file encountered. " );
109        }
110        return line;
111    }
112

```

```

113  /**
114  * Read a word from the Terminal.
115  * If an empty line is entered, try again.
116  * Words are terminated by whitespace.
117  * Leading whitespace is trimmed; the rest of the line
118  * is disposed of.
119  *
120  * @param prompt output string to prompt for input.
121  * @return the word read.
122  */
123  public String readWord( String prompt )
124  {
125      String line = readNonNullLine( prompt );
126      if (line.length() == 0) {
127          println( "Empty line. Please try again." );
128          return readWord( "" );
129      }
130      line = line.trim();
131      for ( int i = 0; i < line.length(); i++ ) {
132          if ( Character.isWhitespace( line.charAt(i) ) ) {
133              return line.substring( 0, i );
134          }
135      }
136      return line;
137  }
138  /**
139  * Read a word from the Terminal.
140  * If an empty line is entered, try again.
141  * Words are terminated by whitespace.
142  * Leading whitespace is trimmed; the rest of the line
143  * is disposed of.
144  *
145  * @return the word read.
146  */
147  public String readWord()
148  {
149      return readWord( "" );
150  }
151  /**
152  * Read a word from the Terminal.
153  * If an empty line is entered, throw an exception.
154  * Words are terminated by whitespace.
155  * Leading whitespace is trimmed; the rest of the line
156  * is disposed of.
157  *
158  * @param prompt output string to prompt for input.
159  * @return the word read.
160  * @throws RuntimeException if it reads an empty line.
161  */
162  public String readWordOnce( String prompt )
163  {
164      String line = readNonNullLine( prompt );
165      if (line.length() == 0) {
166          println( "No character on line. Please try again." );
167          return readChar( "" );
168      }

```

```

169      return line;
170  }
171  /**
172  * Read a character from the Terminal.
173  * If an empty line is entered, throw an exception.
174  * Words are terminated by whitespace.
175  * Leading whitespace is trimmed; the rest of the line
176  * is disposed of.
177  *
178  * @param prompt output string to prompt for input.
179  * @return the character read.
180  */
181  public char readChar( String prompt )
182  {
183      String line = readNonNullLine( prompt );
184      if (line.length() == 0) {
185          println( "No character on line. Please try again." );
186          return readChar( "" );
187      }
188      line = line.trim();
189      for ( int i = 0; i < line.length(); i++ ) {
190          if ( Character.isWhitespace( line.charAt(i) ) ) {
191              return line.charAt( i );
192          }
193      }
194      return line.charAt( 0 );
195  }
196  /**
197  * Read a character from the Terminal.
198  * If an empty line is entered, throw an exception.
199  * Words are terminated by whitespace.
200  * Leading whitespace is trimmed; the rest of the line
201  * is disposed of.
202  *
203  * @param prompt output string to prompt for input.
204  * @return the character read.
205  */
206  public char readCharOnce( String prompt )
207  {
208      String line = readNonNullLine( prompt );
209      if (line.length() == 0) {
210          println( "No character on line. Please try again." );
211          return readChar( "" );
212      }
213      line = line.trim();
214      for ( int i = 0; i < line.length(); i++ ) {
215          if ( Character.isWhitespace( line.charAt(i) ) ) {
216              return line.charAt( i );
217          }
218      }
219      return line.charAt( 0 );
220  }
221  /**
222  * Read a character from the Terminal.
223  * If an empty line is entered, throw an exception.
224  * Words are terminated by whitespace.
225  * Leading whitespace is trimmed; the rest of the line
226  * is disposed of.
227  *
228  * @param prompt output string to prompt for input.
229  */
230  public char readCharOnce()
231  {
232      return readCharOnce( "" );
233  }
234  /**
235  * Read a character from the Terminal.
236  * If an empty line is entered, throw an exception.
237  * Words are terminated by whitespace.
238  * Leading whitespace is trimmed; the rest of the line
239  * is disposed of.
240  *
241  * @return the character read.
242  */
243  public char readChar()
244  {
245      return readChar( "" );
246  }

```

```

225 * @return the character read.
226 *
227 * @throws RuntimeException if it reads an empty line.
228 */
229
230 public char readCharOnce( String prompt )
231 {
232     String line = readNonNullLine(prompt);
233     if (line.length() == 0) {
234         throw new RuntimeException("Empty line encountered.");
235     }
236     return line.charAt(0);
237 }
238
239 /**
240 * Read a character from the Terminal.
241 * Prompt again when an empty line is read.
242 *
243 * @param prompt output string to prompt for input.
244 *
245 * @return the character read.
246 */
247
248 public char readChar()
249 {
250     return readChar("");
251 }
252
253 /**
254 * Read a character from the Terminal.
255 * Throw an exception if an empty line is read.
256 *
257 * @return the character read.
258 *
259 * @throws RuntimeException if it reads an empty line.
260 */
261
262 public char readCharOnce()
263 {
264     return readCharOnce("");
265 }
266
267 /**
268 * Read "yes" or "no" from the Terminal.
269 * If an empty line or improper character is read,
270 * try again.
271 * Look only at first character and accept any case.
272 *
273 * @param prompt output string to prompt for input.
274 * @return true if yes, false if no.
275 */
276
277 public boolean readYesOrNo( String prompt )
278 {
279     printPrompt( prompt );
280     while ( true ) {

```

```

281         char answer = readChar( " (y or n): " );
282         if ( answer == 'y' || answer == 'Y' ) {
283             return true;
284         }
285         else if ( answer == 'n' || answer == 'N' ) {
286             return false;
287         }
288         else {
289             printPrompt( "oops!" );
290         }
291     }
292 }
293
294 /**
295 * Read "yes" or "no" from the Terminal.
296 * If an empty line or improper character is read,
297 * throw an exception.
298 * Look only at first character and accept any case.
299 *
300 * @param prompt output string to prompt for input.
301 * @return true if yes, false if no.
302 *
303 * @throws RuntimeException on improper input.
304 */
305
306 public boolean readYesOrNoOnce( String prompt )
307 {
308     printPrompt( prompt );
309     while ( true ) {
310         char answer = readCharOnce( " (y or n): " );
311         if ( answer == 'y' || answer == 'Y' ) {
312             return true;
313         }
314         else if ( answer == 'n' || answer == 'N' ) {
315             return false;
316         }
317         else {
318             throw new RuntimeException( "Must be y or n." );
319         }
320     }
321 }
322
323 /**
324 * Read "yes" or "no" from the Terminal.
325 * If an empty line or improper character is read,
326 * try again. No prompting is done.
327 * Look only at first character and accept any case.
328 *
329 * @return true if yes, false if no.
330 */
331
332 public boolean readYesOrNo()
333 {
334     while ( true ) {
335         char answer = readChar();
336         if ( answer == 'y' || answer == 'Y' ) {

```

```

337         return true;
338     }
339     else if ( answer == 'n' || answer == 'N' ) {
340         return false;
341     }
342 }
343 }
344 }
345 /**
346  * Read "yes" or "no" from the Terminal.
347  * If an empty line or improper character is read,
348  * throw an exception.
349  * Look only at first character and accept any case.
350  *
351  * @return true if yes, false if no.
352  *
353  * @throws RuntimeException on improper input.
354  */
355
356 public boolean readYesOrNoOnce()
357 {
358     char answer = readCharOnce( " (y or n): " );
359     if ( answer == 'y' || answer == 'Y' ) {
360         return true;
361     }
362     else if ( answer == 'n' || answer == 'N' ) {
363         return false;
364     }
365     else {
366         throw new RuntimeException( "Must be y or n." );
367     }
368 }
369
370 /**
371  * Read an integer, terminated by a new line, from the Terminal.
372  * If a NumberFormatException is encountered, try again.
373  *
374  * @param prompt output string to prompt for input.
375  * @return the input value as an int.
376  */
377
378 public int readInt( String prompt )
379 {
380     while( true ) {
381         try {
382             return Integer.
383                 parseInt(readNonNullLine( prompt ).trim());
384         }
385         catch (NumberFormatException e) {
386             println( "Not an integer. Please try again." );
387         }
388     }
389 }
390
391 /**
392  * Read an integer, terminated by a new line, from the Terminal.

```

```

393
394 *
395 * @param prompt output string to prompt for input.
396 * @return the input value as an int.
397 *
398 * @throws NumberFormatException for a badly formed integer.
399 */
400
401 public int readIntOnce( String prompt )
402 {
403     throws NumberFormatException
404     return Integer.parseInt(readNonNullLine( prompt ).trim());
405 }
406
407 /**
408  * Read an integer, terminated by a new line, from the Terminal.
409  * If a NumberFormatException is encountered, try again.
410  *
411  * @return the input value as an int.
412  */
413
414 public int readInt()
415 {
416     return readInt("");
417 }
418
419 /**
420  * Read an integer, terminated by a new line, from the Terminal.
421  *
422  * @return the input value as an int.
423  *
424  * @throws NumberFormatException for a badly formed integer.
425  */
426
427 public int readIntOnce()
428 {
429     throws NumberFormatException
430     return readIntOnce("");
431 }
432
433 /**
434  * Read a double-precision floating point number,
435  * terminated by a newline, from the Terminal.
436  * If a NumberFormatException is encountered, try again.
437  *
438  * @param prompt output string to prompt for input.
439  * @return the input value as a double.
440  */
441
442 public double readDouble( String prompt )
443 {
444     while( true ) {
445         try {
446             return Double.
447                 parseDouble(readNonNullLine( prompt ).trim());
448         }
449         catch (NumberFormatException e) {

```

```

449         println("Not a floating point number. Please try again.");
450     }
451 }
452 }
453 }
454 /**
455  * Read a double-precision floating point number,
456  * terminated by a newline, from the Terminal.
457  *
458  * @param prompt output string to prompt for input.
459  * @return the input value as a double.
460  *
461  * @throws NumberFormatException for a badly formed number.
462  */
463 public double readDoubleOnce( String prompt )
464     throws NumberFormatException
465 {
466     return Double.parseDouble(readNonNullLine( prompt ).trim());
467 }
468
469 /**
470  * Read a double-precision floating point number,
471  * terminated by a newline, from the Terminal.
472  *
473  * If a NumberFormatException is encountered, try again.
474  *
475  * @return the input value as a double.
476  */
477 public double readDouble()
478 {
479     return readDouble("");
480 }
481
482 /**
483  * Read a double-precision floating point number,
484  * terminated by a newline, from the Terminal.
485  *
486  * @return the input value as a double.
487  *
488  * @throws NumberFormatException for a badly formed number.
489  */
490 public double readDoubleOnce()
491     throws NumberFormatException
492 {
493     return readDouble("");
494 }
495
496 /**
497  * Print a Boolean value
498  * (<code>true</code> or <code>false</code>)
499  * to standard output (without a newline).
500  *
501  * @param b Boolean to print.
502  */
503
504

```

```

505 public void print( boolean b )
506 {
507     System.out.print( b );
508 }
509
510 /**
511  * Print character to standard output (without a newline).
512  *
513  * @param ch character to print.
514  */
515 public void print( char ch )
516 {
517     System.out.print( ch );
518 }
519
520 /**
521  * Print character array to standard output (without a newline).
522  *
523  * @param s character array to print.
524  */
525 public void print( char[] s )
526 {
527     System.out.print( s );
528 }
529
530 /**
531  * Print a double-precision floating point number to standard
532  * output (without a newline).
533  *
534  * @param val number to print.
535  */
536 public void print( double val )
537 {
538     System.out.print( val );
539 }
540
541 /**
542  * Print a floating point number to standard output
543  * (without a newline).
544  *
545  * @param val number to print.
546  */
547 public void print( float val )
548 {
549     System.out.print( val );
550 }
551
552 /**
553  * Print integer to standard output (without a newline).
554  *
555  * @param val integer to print.
556  */
557
558
559
560

```

```

561 */
562 public void print( int val )
563 {
564     System.out.print( val );
565 }
566
567 /**
568  * Print a long integer to standard output (without a newline).
569  *
570  * @param val integer to print.
571  */
572
573 public void print( long val )
574 {
575     System.out.print( val );
576 }
577
578 /**
579  * Print Object to standard output (without a newline).
580  *
581  * @param val Object to print.
582  */
583
584 public void print( Object val )
585 {
586     System.out.print( val.toString() );
587 }
588
589 /**
590  * Print string to standard output (without a newline).
591  *
592  * @param str String to print.
593  */
594
595 public void print( String str )
596 {
597     System.out.print( str );
598 }
599
600 /**
601  * Print a newline to standard output,
602  * terminating the current line.
603  */
604
605 public void println()
606 {
607     System.out.println();
608 }
609
610 /**
611  * Print a Boolean value
612  * (<code>true</code> or <code>false</code>)
613  * to standard output, followed by a newline.
614  * @param b Boolean to print.
615  */
616

```

```

617
618     public void println( boolean b )
619     {
620         System.out.println( b );
621     }
622
623 /**
624  * Print character to standard output, followed by a newline.
625  *
626  * @param ch character to print.
627  */
628
629 public void println( char ch )
630 {
631     System.out.println( ch );
632 }
633
634 /**
635  * Print a character array to standard output,
636  * followed by a newline.
637  *
638  * @param s character array to print.
639  */
640
641 public void println( char[] s )
642 {
643     System.out.println( s );
644 }
645
646 /**
647  * Print floating point number to standard output,
648  * followed by a newline.
649  *
650  * @param val number to print.
651  */
652
653 public void println( float val )
654 {
655     System.out.println( val );
656 }
657
658 /**
659  * Print a double-precision floating point number to standard
660  * output, followed by a newline.
661  *
662  * @param val number to print.
663  */
664
665 public void println( double val )
666 {
667     System.out.println( val );
668 }
669
670 /**
671  * Print integer to standard output, followed by a newline.
672

```

```

673  * @param val Integer to print.
674  */
675
676  public void println( int val )
677  {
678      System.out.println( val );
679  }
680
681  /**
682   * Print a long integer to standard output,
683   * followed by a newline.
684   *
685   * @param val Long integer to print.
686   */
687
688  public void println( long val )
689  {
690      System.out.println( val );
691  }
692
693  /**
694   * Print Object to standard output, followed by a newline.
695   *
696   * @param val Object to print
697   */
698
699  public void println( Object val )
700  {
701      System.out.println( val.toString() );
702  }
703
704  /**
705   * Print string to standard output, followed by a newline.
706   *
707   * @param str String to print
708   */
709
710  public void println( String str )
711  {
712      System.out.println( str );
713  }
714
715  /**
716   * Print a Boolean value
717   * (<code>true</code> or <code>false</code>)
718   * to standard err (without a newline).
719   *
720   * @param b Boolean to print.
721   */
722
723  public void errPrint( boolean b )
724  {
725      System.err.print( b );
726  }
727
728  /**

```

```

729  * Print character to standard err (without a newline).
730  *
731  * @param ch character to print.
732  */
733
734  public void errPrint( char ch )
735  {
736      System.err.print( ch );
737  }
738
739  /**
740   * Print character array to standard err (without a newline).
741   *
742   * @param s character array to print.
743   */
744
745  public void errPrint( char[] s )
746  {
747      System.err.print( s );
748  }
749
750  /**
751   * Print a double-precision floating point number to standard
752   * err (without a newline).
753   *
754   * @param val number to print.
755   */
756
757  public void errPrint( double val )
758  {
759      System.err.print( val );
760  }
761
762  /**
763   * Print a floating point number to standard err
764   * (without a newline).
765   *
766   * @param val number to print.
767   */
768
769  public void errPrint( float val )
770  {
771      System.err.print( val );
772  }
773
774  /**
775   * Print integer to standard err (without a newline).
776   *
777   * @param val integer to print.
778   */
779
780  public void errPrint( int val )
781  {
782      System.err.print( val );
783  }
784

```

```

785      /**
786       * Print a long integer to standard err (without a newline).
787       *
788       * @param val integer to print.
789       */
790
791     public void errPrint( long val )
792     {
793         System.err.print( val );
794     }
795
796     /**
797     * Print Object to standard err (without a newline).
798     *
799     * @param val Object to print.
800     */
801
802     public void errPrint( Object val )
803     {
804         System.err.print( val.toString() );
805     }
806
807     /**
808     * Print string to standard err (without a newline).
809     *
810     * @param str String to print.
811     */
812
813     public void errPrint( String str )
814     {
815         System.err.print( str );
816     }
817
818     /**
819     * Print a newline to standard err,
820     * terminating the current line.
821     */
822
823     public void errPrintln()
824     {
825         System.err.println();
826     }
827
828     /**
829     * Print a Boolean value
830     * (<code>true</code> or <code>false</code>)
831     * to standard err, followed by a newline.
832     */
833     * @param b Boolean to print.
834     */
835
836     public void errPrintln( boolean b )
837     {
838         System.err.println( b );
839     }
840

```

```

841      /**
842       * Print character to standard err, followed by a newline.
843       *
844       * @param ch character to print.
845       */
846
847     public void errPrintln( char ch )
848     {
849         System.err.println( ch );
850     }
851
852     /**
853     * Print a character array to standard err,
854     * followed by a newline.
855     *
856     * @param s character array to print.
857     */
858
859     public void errPrintln( char[] s )
860     {
861         System.err.println( s );
862     }
863
864     /**
865     * Print floating point number to standard err,
866     * followed by a newline.
867     *
868     * @param val number to print.
869     */
870
871     public void errPrintln( float val )
872     {
873         System.err.println( val );
874     }
875
876     /**
877     * Print a double-precision floating point number to
878     * standard err, followed by a newline.
879     *
880     * @param val number to print.
881     */
882
883     public void errPrintln( double val )
884     {
885         System.err.println( val );
886     }
887
888     /**
889     * Print integer to standard err, followed by a newline.
890     *
891     * @param val integer to print.
892     */
893
894     public void errPrintln( int val )
895     {
896         System.err.println( val );
897     }
898

```



```

897     }
898     /**
899     * Print a long integer to standard err, followed by a newline.
900     */
901     * @param val long integer to print.
902     */
903     public void errPrintln( long val )
904     {
905         System.err.println( val );
906     }
907     /**
908     * Print Object to standard err, followed by a newline.
909     */
910     * @param val Object to print
911     */
912     public void errPrintln( Object val )
913     {
914         System.err.println( val.toString() );
915     }
916     /**
917     * Print string to standard err, followed by a newline.
918     */
919     * @param str String to print
920     */
921     public void errPrintln( String str )
922     {
923         System.err.println( str );
924     }
925     /**
926     * Unit test for Terminal.
927     */
928     * @param args command line arguments:
929     * <pre>
930     * -e echo all input.
931     * </pre>
932     */
933     public static void main( String[] args )
934     {
935         Terminal t =
936             new Terminal( args.length == 1 && args[0].equals("-e") );
937         String line = t.readLine( "line:" );
938         String word = t.readWord( "word:" );
939         char c = t.readChar( "char:" );
940         boolean yn = t.readYesOrNo( "yorn:" );
941         double d = t.readDouble( "double:" );
942         int i = t.readInt( "int:" );
943
944
945
946
947
948
949
950
951
952

```

```

953         t.print( " line:[ ]? "); t.print( line ); t.print( " ]");
954         t.print( " line:[ ]? "); t.println( line ); t.print( " ]");
955         t.print( " word:[ ]? "); t.print( word ); t.print( " ]");
956         t.print( " word:[ ]? "); t.println( word ); t.print( " ]");
957         t.print( " char:[ ]? "); t.print( c ); t.print( " ]");
958         t.print( " char:[ ]? "); t.println( c ); t.print( " ]");
959         t.print( " yorn:[ ]? "); t.print( yn ); t.print( " ]");
960         t.print( " yorn:[ ]? "); t.println( yn ); t.print( " ]");
961         t.print( " doub:[ ]? "); t.print( d ); t.print( " ]");
962         t.print( " doub:[ ]? "); t.println( d ); t.print( " ]");
963         t.print( " int:[ ]? "); t.print( i ); t.print( " ]");
964         t.print( " int:[ ]? "); t.println( i ); t.print( " ]");
965         t.print( " line:[ ]? "); t.println( line ); t.print( " ]");
966         t.print( " line:[ ]? "); t.println( line ); t.print( " ]");
967         t.print( " word:[ ]? "); t.println( word ); t.print( " ]");
968         t.print( " word:[ ]? "); t.println( word ); t.print( " ]");
969         t.print( " char:[ ]? "); t.println( c ); t.print( " ]");
970         t.print( " char:[ ]? "); t.println( c ); t.print( " ]");
971         t.print( " yorn:[ ]? "); t.println( yn ); t.print( " ]");
972         t.print( " yorn:[ ]? "); t.println( yn ); t.print( " ]");
973         t.print( " doub:[ ]? "); t.println( d ); t.print( " ]");
974         t.print( " doub:[ ]? "); t.println( d ); t.print( " ]");
975         t.print( " int:[ ]? "); t.println( i ); t.print( " ]");
976         t.print( " int:[ ]? "); t.println( i ); t.print( " ]");
977         t.print( " ");
978         t.print( " ");
979         t.print( " ");
980         t.print( " ");
981         t.print( " ");
982         t.print( " ");
983         t.print( " ");
984         t.print( " ");
985         t.print( " ");
986         t.print( " ");
987         t.print( " ");
988         t.print( " ");
989         t.print( " ");

```