```java
// joi/8/terminal/Terminal.java
// (and terminal/Terminal.java)
//
// Copyright 2003 Bill Campbell and Ethan Bolker

import java.io.*;

/**
 * Terminal provides a user-friendly interface to the standard System
 * input and output streams (in, out, and err).
 * <p>
 * A Terminal is an object.  In general, one is expected to instantiate
 * just one Terminal.  Although one might instantiate several, all will
 * share the same System streams.
 * <p>
 * A Terminal may either explicitly echo input, or not.  Echoing input
 * is useful, for example, when testing with I/O redirection.
 * <p>
 * Inspired by Cay Horstmann’s Console Class.
 */

public class Terminal
{
    private boolean echo = false;
    private static BufferedReader in =
        new BufferedReader(new FileReader(FileDescriptor.in));


    // Print a prompt to the console without a newline.

    private void printPrompt( String prompt )
    {
        print( prompt );
        System.out.flush();
    }

    /**
     * Construct a Terminal that doesn’t echo input.
     */

    public Terminal()
    {
        this( false );
    }

    /**
     * Construct a Terminal.
     *
     * @param echo whether or not input should be echoed.
     */

    public Terminal( boolean echo )
    {
        this.echo = echo;
    }

```

```java
    /**
     * Read a line (terminated by a newline) from the Terminal.
     * @param prompt output string to prompt for input.
     *
     * @return the string (without the newline character),
     *  null if eof.
     */

    public String readLine( String prompt )
    {
        printPrompt(prompt);
        try {
            String line = in.readLine();
            if (echo) {
                println(line);
            }
            return line;
        }
        catch (IOException e) {
            return null;
        }
    }

    /**
     * Read a line (terminated by a newline) from the Terminal.
     *
     * @return the string (without the newline character).
     */

    public String readLine()
    {
        return readLine( "" );
    }

    // Read a line from the Terminal.  An end of file,
    // indicated by a null, raises a runtime exception.
    // Used only internally.

    private String readNonNullLine()
    {
        return readNonNullLine( "" );
    }

    // Read a line from the Terminal.  An end of file,
    // indicated by a null, raises a runtime exception.
    // Used only internally.

    private String readNonNullLine( String prompt )
    {
        String line = readLine( prompt );
        if (line == null ) {
            throw new RuntimeException( "End of File encountered." );
        }
        return line;
    }

    /**
     * Read a word from the Terminal.
     * If an empty line is entered, try again.
     * Words are terminated by whitespace.
     * Leading whitespace is trimmed; the rest of the line
     * is disposed of.
     *
     * @param prompt output string to prompt for input.
     *
     * @return the word read.
     */

    public String readWord( String prompt )
    {
        String line = readNonNullLine( prompt );
        if (line.length() == 0) {
            println( "Empty line.  Please try again." );
            return readWord("");
        }
        line = line.trim();
        for ( int i = 0; i < line.length(); i++ ) {
            if ( Character.isWhitespace( line.charAt(i) ) ) {
                return line.substring( 0, i );
            }
        }
        return line;
    }

    /**
     * Read a word from the Terminal.
     * If an empty line is entered, try again.
     * Words are terminated by whitespace.
     * Leading whitespace is trimmed; the rest of the line
     * is disposed of.
     *
     * @return the word read.
     */

    public String readWord()
    {
        return readWord( "" );
    }

    /**
     * Read a word from the Terminal.
     * If an empty line is entered, throw an exception.
     * Words are terminated by whitespace.
     * Leading whitespace is trimmed; the rest of the line
     * is disposed of.
     *
     * @param prompt output string to prompt for input.
     * @return the word read.
     *
     * @throws RuntimeException if it reads an empty line.
     */

```

```
    public String readWordOnce( String prompt )
    {
        String line = readNonNullLine( prompt );
        if (line.length() == 0) {
            throw new RuntimeException("Empty line encountered.");
        }
        line = line.trim();
        for ( int i = 0; i < line.length(); i++ ) {
            if ( Character.isWhitespace( line.charAt(i) ) ) {
                return line.substring( 0, i );
            }
        }
        return line;
    }

    /**
     * Read a word from the Terminal.
     * If an empty line is entered, throw an exception.
     * Words are terminated by whitespace.
     * Leading whitespace is trimmed; the rest of the line
     * is disposed of.
     *
     * @return the word read.
     *
     * @throws RuntimeException if it reads an empty line.
     */

    public String readWordOnce()
    {
        return readWordOnce( "" );
    }

    /**
     * Read a character from the Terminal.
     * Prompt again when an empty line is read.
     *
     * @param prompt output string to prompt for input.
     *
     * @return the character read.
     */

    public char readChar( String prompt )
    {
        String line = readNonNullLine(prompt);
        if (line.length() == 0) {
            println( "No character on line.  Please try again." );
            return readChar("");
        }
        return line.charAt(0);
    }

    /**
     * Read a character from the Terminal.
     * Throw an exception if an empty line is read.
     *
     * @param prompt output string to prompt for input.
```

```java
     * @return the character read.
     *
     * @throws RuntimeException if it reads an empty line.
     */

    public char readCharOnce( String prompt )
    {
        String line = readNonNullLine(prompt);
        if (line.length() == 0) {
            throw new RuntimeException("Empty line encountered.");
        }
        return line.charAt(0);
    }

    /**
     * Read a character from the Terminal.
     * Prompt again when an empty line is read.
     *
     *
     * @return the character read.
     */

    public char readChar()
    {
        return readChar("");
    }

    /**
     * Read a character from the Terminal.
     * Throw an exception if an empty line is read.
     *
     * @return the character read.
     *
     * @throws RuntimeException if it reads an empty line.
     */

    public char readCharOnce()
    {
        return readCharOnce("");
    }

    /**
     * Read "yes" or "no" from the Terminal.
     * If an empty line or improper character is read,
     * try again.
     * Look only at first character and accept any case.
     *
     * @param prompt output string to prompt for input.
     * @return true if yes, false if no.
     */

    public boolean readYesOrNo( String prompt )
    {
        printPrompt( prompt );
        while ( true ) {
```

```java
            char answer = readChar( " (y or n): ");
            if ( answer == ’y’ || answer == ’Y’ ) {
                return true;
            }
            else if ( answer == ’n’ || answer == ’N’ ) {
                return false;
            }
            else {
                printPrompt( "oops!" );
            }
        }
    }

    /**
     * Read "yes" or "no" from the Terminal.
     * If an empty line or improper character is read,
     * throw an exception.
     * Look only at first character and accept any case.
     *
     * @param prompt output string to prompt for input.
     * @return true if yes, false if no.
     *
     * @throws RuntimeException on improper input.
     */

    public boolean readYesOrNoOnce( String prompt )
    {
        printPrompt( prompt );
        while ( true ) {
            char answer = readCharOnce( " (y or n): ");
            if ( answer == ’y’ || answer == ’Y’ ) {
                return true;
            }
            else if ( answer == ’n’ || answer == ’N’ ) {
                return false;
            }
            else {
                throw new RuntimeException( "Must be y or n." );
            }
        }
    }

    /**
     * Read "yes" or "no" from the Terminal.
     * If an empty line or improper character is read,
     * try again. No prompting is done.
     * Look only at first character and accept any case.
     *
     * @return true if yes, false if no.
     */

    public boolean readYesOrNo()
    {
        while ( true ) {
            char answer = readChar();
            if ( answer == ’y’ || answer == ’Y’ ) {
                return true;
            }
            else if ( answer == ’n’ || answer == ’N’ ) {
                return false;
            }
        }
    }

    /**
     * Read "yes" or "no" from the Terminal.
     * If an empty line or improper character is read,
     * throw an exception.
     * Look only at first character and accept any case.
     *
     * @return true if yes, false if no.
     *
     * @throws RuntimeException on improper input.
     */

    public boolean readYesOrNoOnce()
    {
        char answer = readCharOnce( " (y or n): ");
        if ( answer == ’y’ || answer == ’Y’ ) {
            return true;
        }
        else if ( answer == ’n’ || answer == ’N’ ) {
            return false;
        }
        else {
            throw new RuntimeException( "Must be y or n." );
        }
    }

    /**
     * Read an integer, terminated by a new line, from the Terminal.
     * If a NumberFormatException is encountered, try again.
     *
     * @param prompt output string to prompt for input.
     * @return the input value as an int.
     */

    public int readInt( String prompt )
    {
        while( true ) {
            try {
                return Integer.
                    parseInt(readNonNullLine( prompt ).trim());
            }
            catch (NumberFormatException e) {
                println( "Not an integer.  Please try again." );
            }
        }
    }

    /**
     * Read an integer, terminated by a new line, from the Terminal.
```

```
     *
     * @param prompt output string to prompt for input.
     * @return the input value as an int.
     *
     * @throws NumberFormatException for a badly formed integer.
     */

    public int readIntOnce( String prompt )
        throws NumberFormatException
    {
        return Integer.parseInt(readNonNullLine( prompt ).trim());
    }

    /**
     * Read an integer, terminated by a new line, from the Terminal.
     * If a NumberFormatException is encountered, try again.
     *
     * @return the input value as an int.
     */

    public int readInt()
    {
        return readInt("");
    }

    /**
     * Read an integer, terminated by a new line, from the Terminal.
     *
     * @return the input value as an int.
     *
     * @throws NumberFormatException for a badly formed integer.
     */

    public int readIntOnce()
        throws NumberFormatException
    {
        return readIntOnce("");
    }

    /**
     * Read a double-precision floating point number,
     * terminated by a newline, from the Terminal.
     * If a NumberFormatException is encountered, try again.
     *
     * @param prompt output string to prompt for input.
     * @return the input value as a double.
     */

    public double readDouble( String prompt )
    {
        while( true ) {
            try {
                return Double.
                    parseDouble(readNonNullLine( prompt ).trim());
            }
            catch (NumberFormatException e) {
```

```java
                    println("Not a floating point number. Please try again.");
                }
            }
        }

        /**
         * Read a double-precision floating point number,
         * terminated by a newline, from the Terminal.
         *
         * @param prompt output string to prompt for input.
         * @return the input value as a double.
         *
         * @throws NumberFormatException for a badly formed number.
         */

        public double readDoubleOnce( String prompt )
            throws NumberFormatException
        {
            return Double.parseDouble(readNonNullLine( prompt ).trim());
        }

        /**
         * Read a double-precision floating point number,
         * terminated by a newline, from the Terminal.
         * If a NumberFormatException is encountered, try again.
         *
         * @return the input value as a double.
         */

        public double readDouble()
        {
            return readDouble("");
        }

        /**
         * Read a double-precision floating point number,
         * terminated by a newline, from the Terminal.
         *
         * @return the input value as a double.
         *
         * @throws NumberFormatException for a badly formed number.
         */

        public double readDoubleOnce()
            throws NumberFormatException
        {
            return readDouble("");
        }

        /**
         * Print a Boolean value
         * (<code>true</code> or <code>false</code>)
         * to standard output (without a newline).
         *
         * @param b Boolean to print.
         */
```

```java

        public void print( boolean b )
        {
            System.out.print( b );
        }

        /**
         * Print character to standard output (without a newline).
         *
         * @param ch character to print.
         */

        public void print( char ch )
        {
            System.out.print( ch );
        }

        /**
         * Print character array to standard output (without a newline).
         *
         * @param s character array to print.
         */

        public void print( char[] s )
        {
            System.out.print( s );
        }

        /**
         * Print a double-precision floating point number to standard
         * output (without a newline).
         *
         * @param val number to print.
         */

        public void print( double val )
        {
            System.out.print( val );
        }

        /**
         * Print a floating point number to standard output
         * (without a newline).
         *
         * @param val number to print.
         */

        public void print( float val )
        {
            System.out.print( val );
        }

        /**
         * Print integer to standard output (without a newline).
         *
         * @param val integer to print.
```

```java
     */

    public void print( int val )
    {
        System.out.print( val );
    }

    /**
     * Print a long integer to standard output (without a newline).
     *
     * @param val integer to print.
     */

    public void print( long val )
    {
        System.out.print( val );
    }

    /**
     * Print Object to standard output (without a newline).
     *
     * @param val Object to print.
     */

    public void print( Object val )
    {
        System.out.print( val.toString() );
    }

    /**
     * Print string to standard output (without a newline).
     *
     * @param str String to print.
     */

    public void print( String str )
    {
        System.out.print( str );
    }

    /**
     * Print a newline to standard output,
     * terminating the current line.
     */

    public void println()
    {
        System.out.println();
    }

    /**
     * Print a Boolean value
     * (<code>true</code> or <code>false</code>)
     * to standard output, followed by a newline.
     * @param b Boolean to print.
     */
```

```java

    public void println( boolean b )
    {
        System.out.println( b );
    }

    /**
     * Print character to standard output, followed by a newline.
     *
     * @param ch character to print.
     */

    public void println( char ch )
    {
        System.out.println( ch );
    }

    /**
     * Print a character array to standard output,
     * followed by a newline.
     *
     * @param s character array to print.
     */

    public void println( char[] s )
    {
        System.out.println( s );
    }

    /**
     * Print floating point number to standard output,
     * followed by a newline.
     *
     * @param val number to print.
     */

    public void println( float val )
    {
        System.out.println( val );
    }

    /**
     * Print a double-precision floating point number to standard
     * output, followed by a newline.
     *
     * @param val number to print.
     */

    public void println( double val )
    {
        System.out.println( val );
    }

    /**
     * Print integer to standard output, followed by a newline.
     *
     * @param val integer to print.
     */

    public void println( int val )
    {
        System.out.println( val );
    }

    /**
     * Print a long integer to standard output,
     * followed by a newline.
     *
     * @param val long integer to print.
     */

    public void println( long val )
    {
        System.out.println( val );
    }

    /**
     * Print Object to standard output, followed by a newline.
     *
     * @param val Object to print
     */

    public void println( Object val )
    {
        System.out.println( val.toString() );
    }

    /**
     * Print string to standard output, followed by a newline.
     *
     * @param str String to print
     */

    public void println( String str )
    {
        System.out.println( str );
    }

    /**
     * Print a Boolean value
     * (<code>true</code> or <code>false</code>)
     * to standard err (without a newline).
     *
     * @param b Boolean to print.
     */

    public void errPrint( boolean b )
    {
        System.err.print( b );
    }

    /**
```

```java
     * Print character to standard err (without a newline).
     *
     * @param ch character to print.
     */

    public void errPrint( char ch )
    {
        System.err.print( ch );
    }

    /**
     * Print character array to standard err (without a newline).
     *
     * @param s character array to print.
     */

    public void errPrint( char[] s )
    {
        System.err.print( s );
    }

    /**
     * Print a double-precision floating point number to standard
     * err (without a newline).
     *
     * @param val number to print.
     */

    public void errPrint( double val )
    {
        System.err.print( val );
    }

    /**
     * Print a floating point number to standard err
     * (without a newline).
     *
     * @param val number to print.
     */

    public void errPrint( float val )
    {
        System.err.print( val );
    }

    /**
     * Print integer to standard err (without a newline).
     *
     * @param val integer to print.
     */

    public void errPrint( int val )
    {
        System.err.print( val );
    }

    /**
     * Print a long integer to standard err (without a newline).
     *
     * @param val integer to print.
     */

    public void errPrint( long val )
    {
        System.err.print( val );
    }

    /**
     * Print Object to standard err (without a newline).
     *
     * @param val Object to print.
     */

    public void errPrint( Object val )
    {
        System.err.print( val.toString() );
    }

    /**
     * Print string to standard err (without a newline).
     *
     * @param str String to print.
     */

    public void errPrint( String str )
    {
        System.err.print( str );
    }

    /**
     * Print a newline to standard err,
     * terminating the current line.
     */

    public void errPrintln()
    {
        System.err.println();
    }

    /**
     * Print a Boolean value
     * (<code>true</code> or <code>false</code>)
     * to standard err, followed by a newline.
     *
     * @param b Boolean to print.
     */

    public void errPrintln( boolean b )
    {
        System.err.println( b );
    }

```

```java
    /**
     * Print character to standard err, followed by a newline.
     *
     * @param ch character to print.
     */

    public void errPrintln( char ch )
    {
        System.err.println( ch );
    }

    /**
     * Print a character array to standard err,
     * followed by a newline.
     *
     * @param s character array to print.
     */

    public void errPrintln( char[] s )
    {
        System.err.println( s );
    }

    /**
     * Print floating point number to standard err,
     * followed by a newline.
     *
     * @param val number to print.
     */

    public void errPrintln( float val )
    {
        System.err.println( val );
    }

    /**
     * Print a double-precision floating point number to
     * standard err, followed by a newline.
     *
     * @param val number to print.
     */

    public void errPrintln( double val )
    {
        System.err.println( val );
    }

    /**
     * Print integer to standard err, followed by a newline.
     *
     * @param val integer to print.
     */

    public void errPrintln( int val )
    {
        System.err.println( val );
    }

    /**
     * Print a long integer to standard err, followed by a newline.
     *
     * @param val long integer to print.
     */

    public void errPrintln( long val )
    {
        System.err.println( val );
    }

    /**
     * Print Object to standard err, followed by a newline.
     *
     * @param val Object to print
     */

    public void errPrintln( Object val )
    {
        System.err.println( val.toString() );
    }

    /**
     * Print string to standard err, followed by a newline.
     *
     * @param str String to print
     */

    public void errPrintln( String str )
    {
        System.err.println( str );
    }

    /**
     * Unit test for Terminal.
     *
     * @param args command line arguments:
     * <pre>
     *    -e echo all input.
     * </pre>
     */

    public static void main( String[] args )
    {
        Terminal t =
            new Terminal( args.length == 1 && args[0].equals("-e") );

        String line = t.readLine( "line:" );
        String word = t.readWord( "word:" );
        char   c    = t.readChar( "char:" );
        boolean yn  = t.readYesOrNo( "yorn:" );
        double d    = t.readDouble( "double:" );
        int    i    = t.readInt( "int:" );

```

```java
        t.print("  line:[");  t.print(line);    t.print("]");
        t.print("  line:[");  t.println(line);  t.print("]");

        t.print("  word:[");  t.print(word);    t.print("]");
        t.print("  word:[");  t.println(word);  t.print("]");

        t.print("  char:[");  t.print(c);       t.print("]");
        t.print("  char:[");  t.println(c);     t.print("]");

        t.print("  yorn:[");  t.print(yn);      t.print("]");
        t.print("  yorn:[");  t.println(yn);    t.print("]");

        t.print("  doub:[");  t.print(d);       t.print("]");
        t.print("  doub:[");  t.println(d);     t.print("]");

        t.print("  int:[");   t.print(i);       t.print("]");
        t.print("  int:[");   t.println(i);     t.print("]");

        t.errPrint("  line:[");  t.errPrint(line);    t.errPrint("]");
        t.errPrint("  line:[");  t.errPrintln(line);  t.errPrint("]");

        t.errPrint("  word:[");  t.errPrint(word);    t.errPrint("]");
        t.errPrint("  word:[");  t.errPrintln(word);  t.errPrint("]");

        t.errPrint("  char:[");  t.errPrint(c);       t.errPrint("]");
        t.errPrint("  char:[");  t.errPrintln(c);     t.errPrint("]");

        t.errPrint("  yorn:[");  t.errPrint(yn);      t.errPrint("]");
        t.errPrint("  yorn:[");  t.errPrintln(yn);    t.errPrint("]");

        t.errPrint("  doub:[");  t.errPrint(d);       t.errPrint("]");
        t.errPrint("  doub:[");  t.errPrintln(d);     t.errPrint("]");

        t.errPrint("  int:[");   t.errPrint(i);       t.errPrint("]");
        t.errPrint("  int:[");   t.errPrintln(i);     t.errPrint("]");
    }
}
```

```java
// joi/8/juno/Password.java//
//
//
//  Copyright 2003 Bill Campbell and Ethan Bolker

/**
 * Model a good password.
 *
 * <p>
 * A password is a String satisfying the following conditions
 * (close to those required of Unix passwords, according to
 * the <code> man passwd </code> command in Unix):
 * <br>
 * <ul>
 * <li> A password must have at least PASSLENGTH characters, where
 *      PASSLENGTH defaults to 6. Only the first eight characters
 *      are significant.
 *
 * <li> A password must contain at least two alphabetic characters
 *      and at least one numeric or special character. In this  case,
 *      "alphabetic" refers to all upper or lower case letters.
 *
 * <li> A password must not contain a specified string as a substring
 *      For comparison purposes, an upper case letter and its
 *      corresponding lower case letter are equivalent.
 *
 * <li> A password must not be a substring of a specified string.
 *      For comparison purposes, an upper case letter and its
 *      corresponding lower case letter are equivalent.
 *
 * </ul>
 * <br>
 * A password string may be stored in a Password object only in
 * encrypted form.
 */

public class Password
{
    private String password;

    /**
     * Construct a new Password.
     *
     * @param password the new password.
     * @param notSubstringOf a String that may not contain the password.
     * @param doesNotContain a String the password may not contain.
     *
     * @exception BadPasswordException when password is unacceptable.
     */

    public Password(String password, String notSubstringOf,
                    String doesNotContain)
         throws BadPasswordException
    {
        // if password is not acceptable
        // throw new BadPasswordException( reason )
```

```java
        this.password = encrypt(password);
    }

    // Rewrite s in a form that makes it hard to guess s.

    private String encrypt( String s )
    {
        return Integer.toHexString(s.hashCode());
    }

    /**
     * See whether a supplied guess matches this password.
     *
     * @param guess the trial password.
     *
     * @exception BadPasswordException when match fails.
     */

    public void match(String guess)
         throws BadPasswordException
    {
    }

    /**
     * Unit test for Password objects.
     */

    public static void main( String[] args )
    {
    }
}
```

```java
// joi/8/juno/BadPasswordException.java
//
//
//  Copyright 2003 Bill Campbell and Ethan Bolker

/**
 * The exception thrown when an initial password is unacceptable
 * or a match against an existing password fails.
 */

public class BadPasswordException extends Exception
{
    BadPasswordException()
    {
        super();
    }

    BadPasswordException(String message)
    {
        super(message);
    }
}
```