## Dies ist das erste Dokument mit Prosabeschreibung zur Übung

Es wird zuerst eingebunden.

## Dies ist das zweite (2.) Dokument mit weiterer Prosa...

Es wird nach dem ersten eingebunden.

Listing 1: code/In.java

```
import java.io.*;
import java.util.LinkedList;

/** Simple input from the keyboard or from a file.
<p>This class allows reading formatted data either from the keyboard
or from a file. It is intended to be used in an introductory
programming course when classes, packages and exceptions are unknown
at the beginning. To use it, simply copy In.class into the
source file directory. </p>

<p>All input comes from the current input file, which is initially
the keyboard. Opening a file with open() makes it the new current
input file. Closing a file with close() switches back to the previous
input file.</p>

<p>When reading from the keyboard, reading blocks until the user has entered
a sequence of characters terminated by the return key. All methods read
from this input buffer (including the terminating '\r' and '\n') until the
buffer is fully consumed. When a method tries to read beyond the end
of the buffer, it blocks again waiting for the next buffer.</p>

<p>End of file detection: When reading from the keyboard, eof can be
signaled as ctrl-Z at the beginning of a new line. When reading from a file,
eof occurs when an attempt is made to read beyond the end of the file.
In either case In.done() returns false if the requested data could not
be read because of eof. </p>
*/
public class In {

/** End of file indicator returned by read() or peek() when no more
characters can be read.
*/
public  static final char eof   = '\uffff';

private static final int empty = '\ufffe';

private static final char eofChar = '\u0005';  // ctrl E
private static InputStream in;
private static LinkedList inputStack, bufferStack;
private static boolean done; // true if recent operation was successful
private static char buf;     // last read character
private static char[] LS;    // line separator (eol)

private static char charAfterWhiteSpace() {
  char c;
  do c = read(); while (done && c <= ' ');
  return c;
}

private static String readDigits() {
  StringBuffer b = new StringBuffer();
  char c = charAfterWhiteSpace();
  if (done && c == '-') {
    b.append(c);
    c = read();
  }
  while (done && Character.isDigit(c)) {
    b.append(c);
    c = read();
  }
```

```
    buf = c;
    return b.toString();
  }

  private static String readFloatDigits() {
    StringBuffer b = new StringBuffer();
    char c = charAfterWhiteSpace();
    if (done && (c == '+' || c == '-')) {
      b.append(c);
      c = read();
    }
    while (done && Character.isDigit(c)) {
      b.append(c);
      c = read();
    }
    if (done && (c == '.')) {
      b.append(c);
      c = read();
      while (done && Character.isDigit(c)) {
        b.append(c);
        c = read();
      }
    }
    if (done && (c == 'e' || c == 'E')) {
      b.append(c);
      c = read();
      if (done && (c == '+' || c == '-')) {
        b.append(c);
        c = read();
      }
      while (done && Character.isDigit(c)) {
        b.append(c);
        c = read();
      }
    }
    buf = c;
    return b.toString();
  }


  /** Read a raw character (byte).
  If an attempt is made to read beyond the end of the file,
  eof is returned and done() yields false. Otherwise the read byte
  is in the range 0..255.
  */
  public static char read() {
    char c;
    if (buf != empty) {
      c = buf;
      if (buf != eof) buf = empty;
    } else {
      try {
        c = (char)in.read();
      } catch (IOException e) {
        done = false;
        c = eof; buf = eof;
      }
    }
    if (c == eofChar && inputStack.size() == 0) { c = eof; buf = eof; }
    done = c != eof;
    return c;
  }
```

```
  /** Current available raw characters.
125 In case of an error 0 is returned and done() yields false.
  */
  public static int available() {
    int avail;

130   try {
      avail = in.available();
    } catch(IOException exc) {
      avail = 0;
      done = false;
135   }

    return avail;
  }

140 /** Read a character, but skip white spaces (byte).
  If an attempt is made to read beyond the end of the file,
  eof is returned and done() yields false. Otherwise the read byte
  is in the range 0..255.
  */
145 public static char readChar() {
    return charAfterWhiteSpace();
  }

  /** Read a boolean value.
150 This method skips white space and tries to read an identifier. If its value
  is "true" the method returns true otherwise false. If the identifier is neither
  "true" nor "false" done() yields false.
  */
  public static boolean readBoolean() {
155   String s = readIdentifier();
    done = true;
    if (s.equals("true")) return true;
    else { done = s.equals("false"); return false; }
  }
160
  /** Read an identifier.
  This method skips white space and tries to read an identifier starting
  with a letter and continuing with letters or digits. If a token of this
  structure could be read, it is returned otherwise the empty string is
165 returned and done() yields false.
  */
  public static String readIdentifier() {
    StringBuffer b = new StringBuffer();
    char c = charAfterWhiteSpace();
170   if (done && Character.isLetter(c)) {
      b.append(c);
      c = read();
      while (done && (Character.isLetter(c) || Character.isDigit(c))) {
        b.append(c);
175     c = read();
      }
    }
    buf = c;
    done = b.length() > 0;
180 return b.toString();
  }

  /** Read a word.
  This method skips white space and tries to read a word consisting of
```

```
185   all characters up to the next white space or to the end of the file.
      If a token of this structure could be read, it is returned otherwise
      an empty string is returned and done() yields false.
      */
      public static String readWord() {
190     StringBuffer b = new StringBuffer();
        char c = charAfterWhiteSpace();
        while (done && c > ' ') {
          b.append(c);
          c = read();
195     }
        buf = c;
        done = b.length() > 0;
        return b.toString();
      }
200
      /** Read a line of text.
      This method reads the rest of the current line (including eol) and
      returns it (excluding eol). A line may be empty.
      */
205   public static String readLine() {
        StringBuffer b = new StringBuffer();
        char c = read();
        while (done && c != LS[0]) {
          b.append(c);
210       c = read();
        }

        int i = 0;
        while (c == LS[i]) {
215       ++i;
          if (i >= LS.length) { break; }
          c = read();
        }

220     if (i < LS.length) {
          buf = c;
        } else {
          buf = empty;
        }
225     if (b.length() > 0) done = true;
        return b.toString();
      }

      /** Read the whole file.
230   This method reads from the current position to the end of the
      file and returns its text in a single large string. done() yields
      always true.
      */
      public static String readFile() {
235     StringBuffer b = new StringBuffer();
        char c = charAfterWhiteSpace();
        while (done) {
          b.append(c);
          c = read();
240     }
        buf = eof;
        done = true;
        return b.toString();
      }
245
      /** Read a quote-delimited string.
```

```
  This method skips white space and tries to read a string in the form "...".
  It can be used to read pieces of text that contain white space.
  */
250 public static String readString() {
    StringBuffer b = new StringBuffer();
    char c = charAfterWhiteSpace();
    if (done && c == '"') {
      c = read();
255   while (done && c != '"') {
        b.append(c);
        c = read();
      }
      if (c == '"') { c = read(); done = true; } else done = false;
260 } else done = false;
    buf = c;
    return b.toString();
  }

265 /** Read an integer.
  This method skips white space and tries to read an integer. If the
  text does not contain an integer or if the number is too big, the
  value 0 is returned and the subsequent call of done() yields false.
  An integer is a sequence of digits, possibly preceded by '-'.
270 */
  public static int readInt() {
    String s = readDigits();
    try {
      done = true;
275   return Integer.parseInt(s);
    } catch (Exception e) {
      done = false; return 0;
    }
  }
280
  /** Read a long integer.
  This method skips white space and tries to read a long integer. If the
  text does not contain a number or if the number is too big, the
  value 0 is returned and the subsequent call of done() yields false.
285 A long integer is a sequence of digits, possibly preceded by '-'.
  */
  public static long readLong() {
    String s = readDigits();
    try {
290   done = true;
      return Long.parseLong(s);
    } catch (Exception e) {
      done = false; return 0;
    }
295 }

  /** Read a float value.
  This method skips white space and tries to read a float value. If the
  text does not contain a float value or if the number is not well-formed,
300 the value 0f is returned and the subsequent call of done() yields false.
  An float value is as specified in the Java language description. It may
  be preceded by a '+' or a '-'.
  */
  public static float readFloat() {
305   String s = readFloatDigits();
    try {
      done = true;
      return Float.parseFloat(s);
```

```
    } catch (Exception e) {
      done = false; return 0f;
    }
  }

  /** Read a double value.
  This method skips white space and tries to read a double value. If the
  text does not contain a double value or if the number is not well-formed,
  the value 0.0 is returned and the subsequent call of done() yields false.
  An double value is as specified in the Java language description. It may
  be preceded by a '+' or a '-'.
  */
  public static double readDouble() {
    String s = readFloatDigits();
    try {
      done = true;
      return Double.parseDouble(s);
    } catch (Exception e) {
      done = false; return 0.0;
    }
  }

  /** Peek at the next character.
  This method skips white space and returns the next character without removing
  it from the input stream. It can be used to find out, what token comes next
  in the input stream.
  */
  public static char peek() {
    char c = charAfterWhiteSpace();
    buf = c;
    return c;
  }

  /** Open a text file for reading
  The text file with the name fn is opened as the new current input
  file. When it is closed again, the previous input file is restored.
  */
  public static void open(String fn) {
    try {
      InputStream s = new FileInputStream(fn);
      bufferStack.add(new Character(buf));
      inputStack.add(in);
      in = s;
      done = true;
    } catch (FileNotFoundException e) {
      done = false;
    }
    buf = empty;
  }

  /** Close the current input file.
  The current input file is closed and the previous input file is
  restored. Closing the keyboard input has no effect but causes
  done() to yield false.
  */
  public static void close() {
    try {
      if (inputStack.size() > 0) {
        in.close();
        in = (InputStream) inputStack.removeLast();
        buf = ((Character) bufferStack.removeLast()).charValue();
        done = true;
```

```
    } else {
      done = false; buf = empty;
    }
  } catch (IOException e) {
    done = false; buf = empty;
  }
}

/** Check if the previous operation was successful.
This method returns true if the previous read operation was able
to read a token of the requested structure. It can also be called
after open() and close() to check if these operations were successful.
If done() is called before any other operation it yields true.
*/
public static boolean done() {
  return done;
}

static { // initializer
  done = true;
  in = System.in;
  buf = empty;
  inputStack = new LinkedList();
  bufferStack = new LinkedList();
  LS = System.getProperty("line.separator").toCharArray();
  if (LS == null || LS.length == 0) {
    LS = new char[] { '\n' };
  }
}

}
```

Listing 2: code/Out.java

```
import java.io.*;

/** Simple output to the console and to files.
<p>This class allows printing formatted data either to the console
or to a file. It is intended to be used in an introductory
programming course when classes, packages and exceptions are unknown
at the beginning. To use it, simply copy Out.class into the
current directory. </p>
<p>All output goes to the current output file, which is initially
the console. Opening a file with open() makes it the new current
output file. Closing a file with close() switches back to the previous
output file.</p>
*/

public class Out {

private static PrintStream out;
private static PrintStream[] stack;
private static int sp;
private static boolean done;

/** Return true if the previous Out operation was
successful, otherwise return false. */
public static boolean done() {
  return done && ! out.checkError();
}
```

```
    /** Print the boolean value b either as "true" or "false". */
    public static void print(boolean b) { out.print(b); }

    /** Print the character value c. */
    public static void print(char s) { out.print(s); }

    /** Print the integer value i. */
    public static void print(int i) { out.print(i); }

    /** Print the long value l. */
    public static void print(long l) { out.print(l); }

    /** Print the float value f. */
    public static void print(float f) { out.print(f); }

    /** Print the double value d. */
    public static void print(double d) { out.print(d); }

    /** Print the character array a. */
    public static void print(char[] a) { out.print(a); }

    /** Print the String s. */
    public static void print(String s) { out.print(s); }

    /** Print the Object o as resulting from String.valueOf(o). */
    public static void print(Object o) { out.print(o); }

    /** Terminate the current line by writing a line separator string.
    On windows this is the character sequence '\r' and '\n' */
    public static void println() { out.println(); }

    /** Print the boolean value b and terminate the line. */
    public static void println(boolean b) { out.println(b); }

    /** Print the character value c and terminate the line. */
    public static void println(char s) { out.println(s); }

    /** Print the integer value i and terminate the line. */
    public static void println(int i) { out.println(i); }

    /** Print the long value l and terminate the line. */
    public static void println(long l) { out.println(l); }

    /** Print the float value f and terminate the line. */
    public static void println(float f) { out.println(f); }

    /** Print the double value d and terminate the line. */
    public static void println(double d) { out.println(d); }

    /** Print the character array a and terminate the line. */
    public static void println(char[] a) { out.println(a); }

    /** Print the String s and terminate the line. */
    public static void println(String s) { out.println(s); }

    /** Print the Object o as resulting from String.valueOf(o)
    and terminate the line. */
    public static void println(Object o) { out.println(o); }

    /** Open the file with the name fn as the current output file.
    All subsequent output goes to this file until it is closed.
```

```
  The old output file will be restored when the new output file is closed. */
90 public static void open(String fn) {
     try {
       PrintStream s = new PrintStream(new FileOutputStream(fn));
       stack[sp++] = out;
       out = s;
95   } catch (Exception e) {
       done = false;
     }
   }

100 /** Close the current output file.
   The previous output file is restored and becomes the current output file. */
   public static void close() {
     out.flush();
     out.close();
105  if (sp > 0) out = stack[--sp];
   }

   static { // initializer
     done = true;
110  out = System.out;
     stack = new PrintStream[8];
     sp = 0;
   }

115 }
```