

every change to the file's data or metadata will be immediately written to the physical device. If it is "rwd", the file is opened for read-write operations and every change to the file's data will be immediately written to the physical device.

The method **seek()**, shown here, is used to set the current position of the file pointer within the file:

```
void seek(long newPos) throws IOException
```

Here, *newPos* specifies the new position, in bytes, of the file pointer from the beginning of the file. After a call to **seek()**, the next read or write operation will occur at the new file position.

RandomAccessFile implements the standard input and output methods, which you can use to read and write to random access files. It also includes some additional methods. One is **setLength()**. It has this signature:

```
void setLength(long len) throws IOException
```

This method sets the length of the invoking file to that specified by *len*. This method can be used to lengthen or shorten a file. If the file is lengthened, the added portion is undefined.

The Character Streams

While the byte stream classes provide sufficient functionality to handle any type of I/O operation, they cannot work directly with Unicode characters. Since one of the main purposes of Java is to support the "write once, run anywhere" philosophy, it was necessary to include direct I/O support for characters. In this section, several of the character I/O classes are discussed. As explained earlier, at the top of the character stream hierarchies are the **Reader** and **Writer** abstract classes. We will begin with them.

Reader

Reader is an abstract class that defines Java's model of streaming character input. It implements the **AutoCloseable**, **Closeable**, and **Readable** interfaces. All of the methods in this class (except for **markSupported()**) will throw an **IOException** on error conditions. Table 20-3 provides a synopsis of the methods in **Reader**.

Writer

Writer is an abstract class that defines streaming character output. It implements the **AutoCloseable**, **Closeable**, **Flushable**, and **Appendable** interfaces. All of the methods in this class throw an **IOException** in the case of errors. Table 20-4 shows a synopsis of the methods in **Writer**.

Method	Description
<code>abstract void close()</code>	Closes the input source. Further read attempts will generate an IOException .
<code>void mark(int numChars)</code>	Places a mark at the current point in the input stream that will remain valid until <i>numChars</i> characters are read.
<code>boolean markSupported()</code>	Returns true if mark() / reset() are supported on this stream.
<code>int read()</code>	Returns an integer representation of the next available character from the invoking input stream. -1 is returned when the end of the file is encountered.
<code>int read(char buffer[])</code>	Attempts to read up to <i>buffer.length</i> characters into <i>buffer</i> and returns the actual number of characters that were successfully read. -1 is returned when the end of the file is encountered.
<code>int read(CharBuffer buffer)</code>	Attempts to read characters into <i>buffer</i> and returns the actual number of characters that were successfully read. -1 is returned when the end of the file is encountered.
<code>abstract int read(char buffer[], int offset, int numChars)</code>	Attempts to read up to <i>numChars</i> characters into <i>buffer</i> starting at <i>buffer[offset]</i> , returning the number of characters successfully read. -1 is returned when the end of the file is encountered.
<code>boolean ready()</code>	Returns true if the next input request will not wait. Otherwise, it returns false .
<code>void reset()</code>	Resets the input pointer to the previously set mark.
<code>long skip(long numChars)</code>	Skips over <i>numChars</i> characters of input, returning the number of characters actually skipped.

Table 20-3 The Methods Defined by **Reader**

Method	Description
<code>Writer append(char ch)</code>	Appends <i>ch</i> to the end of the invoking output stream. Returns a reference to the invoking stream.
<code>Writer append(CharSequence chars)</code>	Appends <i>chars</i> to the end of the invoking output stream. Returns a reference to the invoking stream.
<code>Writer append(CharSequence chars, int begin, int end)</code>	Appends the subrange of <i>chars</i> specified by <i>begin</i> and <i>end-1</i> to the end of the invoking output stream. Returns a reference to the invoking stream.
<code>abstract void close()</code>	Closes the output stream. Further write attempts will generate an IOException .
<code>abstract void flush()</code>	Finalizes the output state so that any buffers are cleared. That is, it flushes the output buffers.

Table 20-4 The Methods Defined by **Writer**

Method	Description
<code>void write(int <i>ch</i>)</code>	Writes a single character to the invoking output stream. Note that the parameter is an int , which allows you to call write with an expression without having to cast it back to char . However, only the low-order 16 bits are written.
<code>void write(char <i>buffer</i>[])</code>	Writes a complete array of characters to the invoking output stream.
abstract <code>void write(char <i>buffer</i>[], int <i>offset</i>, int <i>numChars</i>)</code>	Writes a subrange of <i>numChars</i> characters from the array <i>buffer</i> , beginning at <i>buffer</i> [<i>offset</i>] to the invoking output stream.
<code>void write(String <i>str</i>)</code>	Writes <i>str</i> to the invoking output stream.
<code>void write(String <i>str</i>, int <i>offset</i>, int <i>numChars</i>)</code>	Writes a subrange of <i>numChars</i> characters from the string <i>str</i> , beginning at the specified <i>offset</i> .

Table 20-4 The Methods Defined by **Writer** (continued)

FileReader

The **FileReader** class creates a **Reader** that you can use to read the contents of a file. Two commonly used constructors are shown here:

```
FileReader(String filePath)
FileReader(File fileObj)
```

Either can throw a **FileNotFoundException**. Here, *filePath* is the full path name of a file, and *fileObj* is a **File** object that describes the file.

The following example shows how to read lines from a file and display them on the standard output device. It reads its own source file, which must be in the current directory.

```
// Demonstrate FileReader.
// This program uses try-with-resources. It requires JDK 7 or later.

import java.io.*;

class FileReaderDemo {
    public static void main(String args[]) {

        try ( FileReader fr = new FileReader("FileReaderDemo.java") )
        {
            int c;

            // Read and display the file.
            while((c = fr.read()) != -1) System.out.print((char) c);

        } catch(IOException e) {
            System.out.println("I/O Error: " + e);
        }
    }
}
```

FileWriter

FileWriter creates a **Writer** that you can use to write to a file. Four commonly used constructors are shown here:

```
FileWriter(String filePath)
FileWriter(String filePath, boolean append)
FileWriter(File fileObj)
FileWriter(File fileObj, boolean append)
```

They can all throw an **IOException**. Here, *filePath* is the full path name of a file, and *fileObj* is a **File** object that describes the file. If *append* is **true**, then output is appended to the end of the file.

Creation of a **FileWriter** is not dependent on the file already existing. **FileWriter** will create the file before opening it for output when you create the object. In the case where you attempt to open a read-only file, an **IOException** will be thrown.

The following example is a character stream version of an example shown earlier when **FileOutputStream** was discussed. This version creates a sample buffer of characters by first making a **String** and then using the **getChars()** method to extract the character array equivalent. It then creates three files. The first, **file1.txt**, will contain every other character from the sample. The second, **file2.txt**, will contain the entire set of characters. Finally, the third, **file3.txt**, will contain only the last quarter.

```
// Demonstrate FileWriter.
// This program uses try-with-resources. It requires JDK 7 or later.

import java.io.*;

class FileWriterDemo {
    public static void main(String args[]) throws IOException {
        String source = "Now is the time for all good men\n"
            + "to come to the aid of their country\n"
            + "and pay their due taxes.";
        char buffer[] = new char[source.length()];
        source.getChars(0, source.length(), buffer, 0);

        try ( FileWriter f0 = new FileWriter("file1.txt");
              FileWriter f1 = new FileWriter("file2.txt");
              FileWriter f2 = new FileWriter("file3.txt") )
        {
            // write to first file
            for (int i=0; i < buffer.length; i += 2) {
                f0.write(buffer[i]);
            }

            // write to second file
            f1.write(buffer);

            // write to third file
            f2.write(buffer, buffer.length-buffer.length/4, buffer.length/4);
        }
    }
}
```

```

    } catch(IOException e) {
        System.out.println("An I/O Error Occurred");
    }
}
}

```

CharArrayReader

CharArrayReader is an implementation of an input stream that uses a character array as the source. This class has two constructors, each of which requires a character array to provide the data source:

```

CharArrayReader(char array [ ])
CharArrayReader(char array [ ], int start, int numChars)

```

Here, *array* is the input source. The second constructor creates a **Reader** from a subset of your character array that begins with the character at the index specified by *start* and is *numChars* long.

The **close()** method implemented by **CharArrayReader** does not throw any exceptions. This is because it cannot fail.

The following example uses a pair of **CharArrayReaders**:

```

// Demonstrate CharArrayReader.
// This program uses try-with-resources. It requires JDK 7 or later.

import java.io.*;

public class CharArrayReaderDemo {
    public static void main(String args[]) {
        String tmp = "abcdefghijklmnopqrstuvwxyz";
        int length = tmp.length();
        char c[] = new char[length];

        tmp.getChars(0, length, c, 0);
        int i;

        try (CharArrayReader input1 = new CharArrayReader(c) )
        {
            System.out.println("input1 is:");
            while((i = input1.read()) != -1) {
                System.out.print((char)i);
            }
            System.out.println();
        } catch(IOException e) {
            System.out.println("I/O Error: " + e);
        }

        try ( CharArrayReader input2 = new CharArrayReader(c, 0, 5) )
        {
            System.out.println("input2 is:");
            while((i = input2.read()) != -1) {
                System.out.print((char)i);
            }
        }
    }
}

```