17.6.5 What will happen when you attempt to run the following code?

```
import java.io.*;

public class Test {
  public static void main(String[] args) throws IOException {
    try ( ObjectOutputStream output =
        new ObjectOutputStream(new FileOutputStream("object.dat")); ) {
      output.writeObject(new A());
    }
  }
}

class A implements Serializable {
  B b = new B();
}

class B {
}
```

17.6.6 Can you write an array to an `ObjectOutputStream`?

17.7 Random-Access Files

Java provides the `RandomAccessFile` *class to allow data to be read from and written to at any locations in the file.*

read-only
write-only
sequential-access file

All of the streams you have used so far are known as *read-only* or *write-only* streams. These streams are called *sequential streams*. A file that is opened using a sequential stream is called a *sequential-access file*. The contents of a sequential-access file cannot be updated. However, it is often necessary to modify files. Java provides the `RandomAccessFile` class to allow data to be read from and written to at any locations in the file. A file that is opened using the `RandomAccessFile` class is known as a *random-access file*.

random-access file

The `RandomAccessFile` class implements the `DataInput` and `DataOutput` interfaces, as shown in Figure 17.18. The `DataInput` interface (see Figure 17.9) defines the methods for reading primitive-type values and strings (e.g., `readInt`, `readDouble`, `readChar`, `readBoolean`, and `readUTF`) and the `DataOutput` interface (see Figure 17.10) defines the methods for writing primitive-type values and strings (e.g., `writeInt`, `writeDouble`, `writeChar`, `writeBoolean`, and `writeUTF`).

When creating a `RandomAccessFile`, you can specify one of the two modes: `r` or `rw`. Mode `r` means that the stream is read-only, and mode `rw` indicates that the stream allows both read and write. For example, the following statement creates a new stream, `raf`, that allows the program to read from and write to the file **test.dat**:

```
RandomAccessFile raf = new RandomAccessFile("test.dat", "rw");
```

If **test.dat** already exists, `raf` is created to access it; if **test.dat** does not exist, a new file named **test.dat** is created and `raf` is created to access the new file. The method `raf.length()` returns the number of bytes in **test.dat** at any given time. If you append new data into the file, `raf.length()` increases.

Tip

If the file is not intended to be modified, open it with the `r` mode. This prevents unintentional modification of the file.

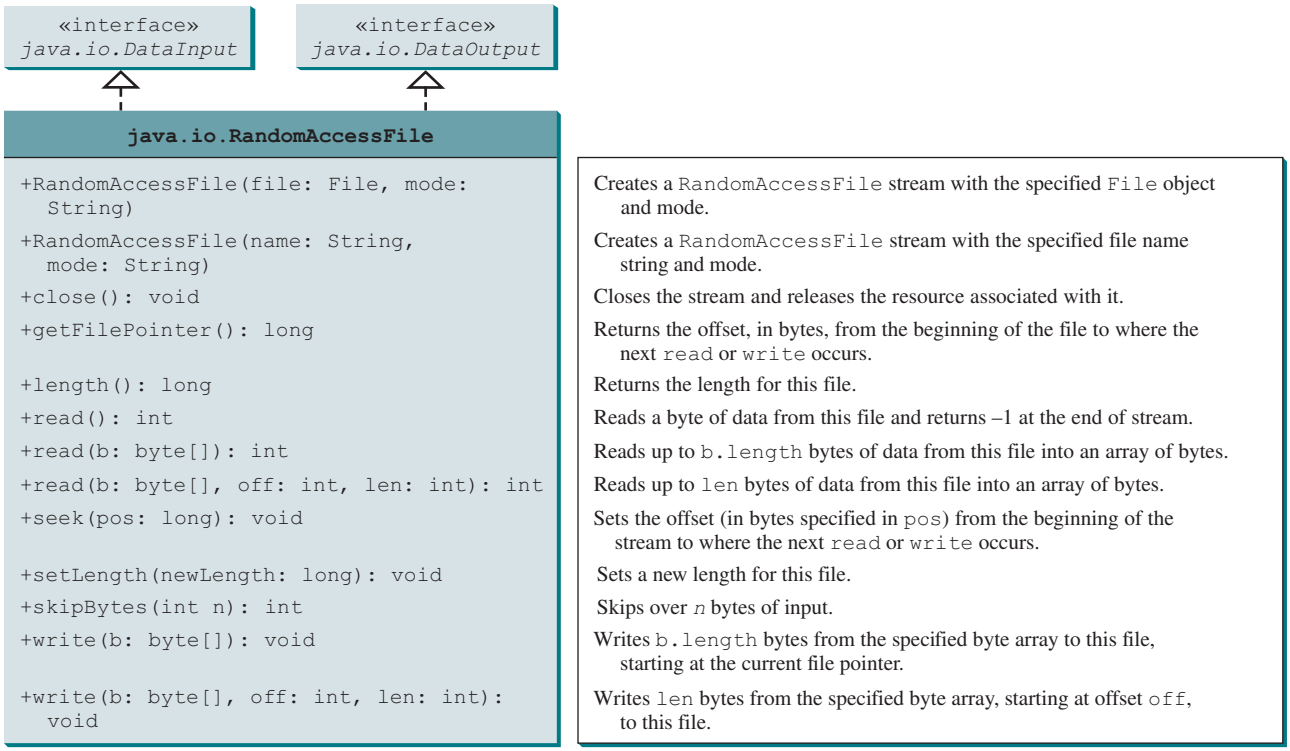

FIGURE 17.18 `RandomAccessFile` implements the `DataInput` and `DataOutput` interfaces with additional methods to support random access.

file pointer

A random-access file consists of a sequence of bytes. A special marker called a *file pointer* is positioned at one of these bytes. A read or write operation takes place at the location of the file pointer. When a file is opened, the file pointer is set at the beginning of the file. When you read from or write data to the file, the file pointer moves forward to the next data item. For example, if you read an `int` value using `readInt()`, the JVM reads `4` bytes from the file pointer and now the file pointer is `4` bytes ahead of the previous location, as shown in Figure 17.19.

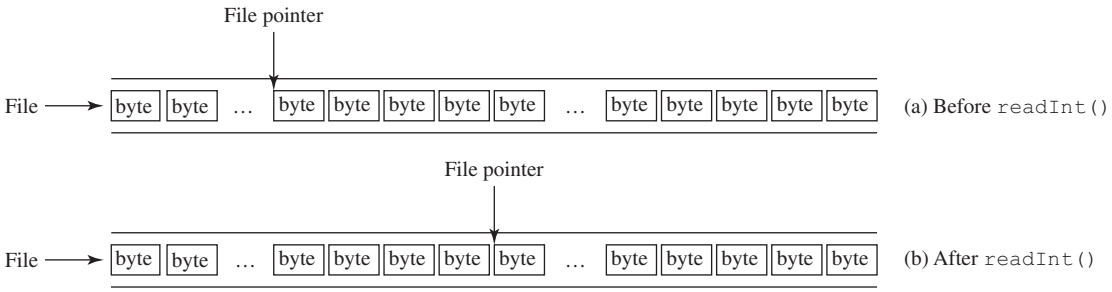

FIGURE 17.19 After an `int` value is read, the file pointer is moved 4 bytes ahead.

For a `RandomAccessFile raf`, you can use the `raf.seek(position)` method to move the file pointer to a specified position. `raf.seek(0)` moves it to the beginning of the file and `raf.seek(raf.length())` moves it to the end of the file. Listing 17.8 demonstrates `RandomAccessFile`. A large case study of using `RandomAccessFile` to organize an address book is given in Supplement VI.D.

LISTING 17.8 TestRandomAccessFile.java

```
1  import java.io.*;
2
3  public class TestRandomAccessFile {
4    public static void main(String[] args) throws IOException {
5      try ( // Create a random access file
6        RandomAccessFile inout = new RandomAccessFile("inout.dat", "rw");   RandomAccessFile
7      ) {
8        // Clear the file to destroy the old contents if exists
9        inout.setLength(0);                                                 empty file
10
11       // Write new integers to the file
12       for (int i = 0; i < 200; i++)
13         inout.writeInt(i);                                                write
14
15       // Display the current length of the file
16       System.out.println("Current file length is " + inout.length());
17
18       // Retrieve the first number
19       inout.seek(0); // Move the file pointer to the beginning           move pointer
20       System.out.println("The first number is " + inout.readInt());      read
21
22       // Retrieve the second number
23       inout.seek(1 * 4); // Move the file pointer to the second number
24       System.out.println("The second number is " + inout.readInt());
25
26       // Retrieve the tenth number
27       inout.seek(9 * 4); // Move the file pointer to the tenth number
28       System.out.println("The tenth number is " + inout.readInt());
29
30       // Modify the eleventh number
31       inout.writeInt(555);
32
33       // Append a new number
34       inout.seek(inout.length()); // Move the file pointer to the end
35       inout.writeInt(999);
36
37       // Display the new length
38       System.out.println("The new length is " + inout.length());
39
40       // Retrieve the new eleventh number
41       inout.seek(10 * 4); // Move the file pointer to the eleventh number
42       System.out.println("The eleventh number is " + inout.readInt());
43     }
44   }
45 }
```

```
Current file length is 800
The first number is 0
The second number is 1
The tenth number is 9
The new length is 804
The eleventh number is 555
```

A `RandomAccessFile` is created for the file named **inout.dat** with mode `rw` to allow both read and write operations in line 6.

`inout.setLength(0)` sets the length to `0` in line 9. This, in effect, deletes the old contents of the file.

The `for` loop writes `200` `int` values from `0` to `199` into the file in lines 12–13. Since each `int` value takes `4` bytes, the total length of the file returned from `inout.length()` is now `800` (line 16), as shown in the sample output.

Invoking `inout.seek(0)` in line 19 sets the file pointer to the beginning of the file. `inout.readInt()` reads the first value in line 20 and moves the file pointer to the next number. The second number is read in line 24.

`inout.seek(9 * 4)` (line 27) moves the file pointer to the tenth number. `inout.readInt()` reads the tenth number and moves the file pointer to the eleventh number in line 28. `inout.write(555)` writes a new eleventh number at the current position (line 31). The previous eleventh number is deleted.

`inout.seek(inout.length())` moves the file pointer to the end of the file (line 34). `inout.writeInt(999)` writes a `999` to the file (line 35). Now the length of the file is increased by `4`, so `inout.length()` returns `804` (line 38).

`inout.seek(10 * 4)` moves the file pointer to the eleventh number in line 41. The new eleventh number, `555`, is displayed in line 42.

17.7.1 Can `RandomAccessFile` streams read and write a data file created by `DataOutputStream`? Can `RandomAccessFile` streams read and write objects?

17.7.2 Create a `RandomAccessFile` stream for the file **address.dat** to allow the updating of student information in the file. Create a `DataOutputStream` for the file **address.dat**. Explain the differences between these two statements.

17.7.3 What happens if the file **test.dat** does not exist when you attempt to compile and run the following code?

```
import java.io.*;

public class Test {
  public static void main(String[] args) {
    try ( RandomAccessFile raf =
        new RandomAccessFile("test.dat", "r"); ) {
      int i = raf.readInt();
    }
    catch (IOException ex) {
      System.out.println("IO exception");
    }
  }
}
```

KEY TERMS