APPENDIX I

Working with Records and Random-Access Files

Data that is written to a file is commonly structured as fields and records. In file terminology, a *field* is an individual piece of data, such as a person's name or telephone number. A *record* is a collection of fields pertaining to a single item. For example, a record might consist of a specific person's name, age, address, and telephone number.

Quite often you can save the contents of an object as a record in a file. You do this by writing each of the object's fields to the file, one after the other. When you have saved all of the object's fields, a complete record has been written. When the fields from multiple objects have been saved, then multiple records have been written to the file.

Random access files are particularly useful for storing and retrieving records. However, the sizes of the items stored in a random access file must be known in order to calculate the position of a specific item. Records that are stored in a random access file must be the same size and must have a *fixed length*. This means that the size of a record cannot change.

In Java, the sizes of the primitive data types are well documented and guaranteed to be the same on all systems. If an object's fields are all of the primitive data types, you can easily calculate the size of the record: it will be the sum of the sizes of all the fields. However, a problem arises if an object has a field that is a `String` because it can vary in length. You can get around this problem by making sure that a `String` field is always written as a specific number of characters. The following example shows one way to do this.

In Chapter 10 you saw the `InventoryItem` class. This class has two fields: `description`, a `String` and `units`, an `int`. The `InventoryItemFile` class shown in Code Listing H-1 is designed to read and write `InventoryItem` objects as records in a random access file. The class can also move the file pointer to a specific record. To keep the code simple, none of the exceptions are caught.

Code Listing I-1 (InventoryItemFile.java)

```
 1 import java.io.*;
 2
 3 /**
 4  * This class manages a random access file which contains
 5  * InventoryItem records.
 6  */
 7
 8 public class InventoryItemFile
 9 {
10    private final int RECORD_SIZE = 44;     // Record length
11    private RandomAccessFile inventoryFile; // Random-Access file
12
13    /**
14     * Constructor
15     */
16
17    public InventoryItemFile(Stringfilename)
18                          throws FileNotFoundException
19    {
20       // Open the file for reading and writing.
21       inventoryFile = new RandomAccessFile(filename, "rw");
22    }
23
24    /**
25     * The writeInventoryItem method writes the
26     * InventoryItem argument to the file at the
27     * current file pointer position.
28     */
29
30    public void writeInventoryItem(InventoryItem item)
31                                   throws IOException
32    {
33       // Get the item's description.
34       String str = item.getDescription();
35
36       // Write the description.
37       if (str.length() > 20)
38       {
39          // If there are more than 20 characters in the
40          // string, then write only the first 20 to the file.
41          for (int i = 0; i < 20; i++)
42             inventoryFile.writeChar(str.charAt(i));
43       }
44       else
45       {
```

```
46          // Write the description to the file.
47          inventoryFile.writeChars(str);
48          // Write enough spaces to pad it out
49          // to 20 characters.
50          for (int i = 0; i < (20 - str.length()); i++)
51             inventoryFile.writeChar(' ');
52       }
53
54       // Write the units to the file.
55       inventoryFile.writeInt(item.getUnits());
56    }
57
58    /**
59     * The readInventoryItem method reads and returns
60     * the record at the current file pointer position.
61     */
62
63    public InventoryItem readInventoryItem() throws IOException
64    {
65       // Create a char array with 20 elements.
66       char[] charArray = new char[20];
67
68       // Read the description, character by character,
69       // from the file into the char array.
70       for (int i = 0; i < 20; i++)
71          charArray[i] = inventoryFile.readChar();
72
73       // Store the char array in a String.
74       String description = new String(charArray);
75
76       // Trim any trailing spaces from the string.
77       description.trim();
78
79       // Read the units from the file.
80       int units = inventoryFile.readInt();
81
82       // Create an InventoryItem object and initialize
83       // it with these values.
84       InventoryItem item = new InventoryItem(description, units);
85
86       // Return the object.
87       return item;
88    }
89
90    /**
91     * The getByteNum method accepts a record number
92     * and returns that record's starting byte number.
```

```
 93      */
 94
 95     private long getByteNum(long recordNum)
 96     {
 97        return RECORD_SIZE * recordNum;
 98     }
 99
100     /**
101      * The moveFilePointer method moves the file
102      * to the record indicated by the argument.
103      */
104
105     public void moveFilePointer(longrecordNum)
106                                 throws IOException
107     {
108        inventoryFile.seek(getByteNum(recordNum));
109     }
110
111     /**
112      * The getNumberOfRecords method returns the number
113      * of records stored in the file.
114      */
115
116     public long getNumberOfRecords() throws IOException
117     {
118        return inventoryFile.length() / RECORD_SIZE;
119     }
120
121     /**
122      * The close method closes the file.
123      */
124
125     public void close() throws IOException
126     {
127        inventoryFile.close();
128     }
129 }
```

The `RECORD_SIZE` field, declared in line 10, is a `final int` variable initialized with the value 44. This is the size, in bytes, of a record. In a moment you will see how this number was determined. The `inventoryFile` field, declared in line 11, is a `RandomAccessFile` reference variable that will be used to open and work with a random access file. The constructor accepts a file name as a `String`. This file name is used to open a random access file, referenced by the `inventoryFile` variable, file for reading and writing.

By looking at the `writeInventoryItem` method, which begins at line 30, we can see how the record size of 44 bytes was determined. The method accepts an `InventoryItem` object as an argument, the contents of which will be written as a record to the file. In line 34 the `InventoryItem` object's `description` field is retrieved (via a call to the `getDescription` method) and referenced by `str`, a local variable. Next, we write the `description` field to the file. To ensure that each record has the same fixed length, this method always writes the description as 20 characters. If the description has more than 20 characters, then the `for` loop in lines 41 through 42 executes, writing only the first 20 characters. Otherwise, the statement in line 47 writes the description to the file, and the `for` loop in lines 50 through 51 writes enough spaces to make up the difference. Next, in line 55, the method writes the `units` field, as an `int`, to the file.

Now we can see how the record size of 44 bytes was determined. When a character is written to the file, it is written as two bytes. The `description` field is always written as 20 characters, so that's 40 bytes. The `units` field is written as an `int`, which uses four bytes. That makes a total record size of 44 bytes.

The `readInventoryItem` method, which begins at line 63, reads a record from the file and returns an `InventoryItem` object containing the record's data. First, in line 66, a 20 element char array is created to hold the description. Then the `for` loop in lines 70 through 71 reads the 20 characters from the file and stores them in the array.

Next, in line 74, a `String` object is created and the `char` array is passed as an argument to the `String` constructor. This copies the characters from the array to the `String` object. If the description was less than 20 characters long, it will be padded with trailing spaces. So, line 77 trims any trailing spaces that might be in the string.

The statement in line 80 reads an integer from the file and stores it in the units variable. Line 84 constructs an `InventoryItem` object with the data we have read into `description` and `units`. Line 87 returns a reference to the object.

The class also has the ability to move the file pointer to a specific record. Two methods work together to perform this. First, `getByteNum`, which appears in lines 95 through 98, is a private method that accepts a record number as an argument, and returns the record's starting byte number. It calculates the starting byte number by multiplying the record size by the record number. (The first record in the file is considered record 0.) The `moveFilePointer` method, in lines 105 through 109, accepts a record number as its argument, and moves the file pointer to the specified record. This method calls the `getByteNum` method to determine the record's starting location.

The `getNumberOfRecords` method appears in lines 116 through 119. This method returns the number of records in the file. It calculates the number of records by dividing the length of the file by the record size. The length of the file is returned by the `RandomAccessFile` class's length method.

The last method in the class is the `close` method, in lines 125 through 128, which closes the file. The program in Code Listing H-2 shows a simple demonstration of this class. This program asks the user to enter data for five items, which is stored in an array of `InventoryItem` objects. The program then saves the contents of the array elements to a file.

Code Listing I-2 (CreateInventoryFile.java)

```
 1 import java.io.*;
 2 import java.util.Scanner;
 3
 4 /**
 5  * This program uses the InventoryFile class to create
 6  * a file containing data from 5 InventoryItem objects.
 7  */
 8
 9 public class CreateInventoryFile
10 {
11    public static void main(String[] args) throws IOException
12    {
13       final int NUM_ITEMS = 5;    // Number of items
14       String description;         // Item description
15       int units;                  // Units on hand
16
17       // Create a Scanner object for keyboard input.
18       Scanner keyboard = new Scanner(System.in);
19
20       // Create an array to hold InventoryItem objects.
21       InventoryItem[] items = new InventoryItem[NUM_ITEMS];
22
23       // Get data for the InventoryItem objects.
24       System.out.println("Enter data for " + NUM_ITEMS +
25                          " inventory items.");
26
27       for (int i = 0; i < items.length; i++)
28       {
29          // Get the description.
30          System.out.print("Enter an item description: ");
31          description = keyboard.nextLine();
32
33          // Get the units on hand.
34          System.out.print("Enter the number of units: ");
35          units = keyboard.nextInt();
36
37          // Consume the remaining newline.
38          keyboard.nextLine();
39
40          // Create an InventoryItem object in the array.
41          items[i] = new InventoryItem(description, units);
42       }
43
44       // Create an InventoryFile object.
45       InventoryItemFile file =
46                     new InventoryItemFile("Inventory.dat");
47
```

```
48        // Write the contents of the array to the file.
49        for (int i = 0; i < items.length; i++)
50        {
51           file.writeInventoryItem(items[i]);
52        }
53
54        // Close the file.
55        file.close();
56
57        System.out.println("The data was written to the " +
58                           "Inventory.dat file.");
59     }
60  }
```

Program Output with Example Input Shown in Bold

```
Enter data for 5 inventory items.
Enter an item description: Wrench [Enter]
Enter the number of units: 20 [Enter]
Enter an item description: Hammer [Enter]
Enter the number of units: 15 [Enter]
Enter an item description: Pliers [Enter]
Enter the number of units: 12 [Enter]
Enter an item description: Screwdriver [Enter]
Enter the number of units: 25 [Enter]
Enter an item description: Ratchet [Enter]
Enter the number of units: 10 [Enter]
The data was written to the Inventory.dat file.
```

The program in Code Listing I-3 demonstrates how records can be randomly read from the file.

Code Listing I-3 (ReadInventoryFile.java)

```
 1 import java.io.*;
 2 import java.util.Scanner;
 3
 4 /**
 5  * This program displays specified records from the
 6  * Inventory.dat file.
 7  */
 8
 9 public class ReadInventoryFile
10 {
11    public static void main(String[] args) throws IOException
12    {
```

```
13        int recordNumber;           // Record number
14        String again;               // To get a Y or an N
15        InventoryItem item;         // An object from the file
16
17        // Create a Scanner object for keyboard input.
18        Scanner keyboard = new Scanner(System.in);
19
20        // Open the file.
21        InventoryItemFile file =
22                 new InventoryItemFile("Inventory.dat");
23
24        // Report the number of records in the file.
25        System.out.println("The Inventory.dat file has " +
26                 file.getNumberOfRecords() + " records.");
27
28        // Get a record number from the user and
29        // display the record.
30        do
31        {
32           // Get the record number.
33           System.out.print("Enter the number of the record " +
34                            "you wish to see: ");
35           recordNumber = keyboard.nextInt();
36
37           // Consume the remaining newline.
38           keyboard.nextLine();
39
40           // Move the file pointer to that record.
41           file.moveFilePointer(recordNumber);
42
43           // Read the record at that location.
44           item = file.readInventoryItem();
45
46           // Display the record.
47           System.out.println("\nDescription: " +
48                              item.getDescription());
49           System.out.println("Units: " + item.getUnits());
50
51           // Ask the user whether to get another record.
52           System.out.print("\nDo you want to see another " +
53                            "record? (Y/N): ");
54           again = keyboard.nextLine();
55        } while (again.charAt(0) == 'Y' || again.charAt(0) == 'y');
56
57        // Close the file.
58        file.close();
59     }
60 }
```

Program Output with Example Input Shown in Bold

```
The Inventory.dat file has 5 records.
Enter the number of the record you wish to see: 4 [Enter]
Description: Ratchet
Units: 10
Do you want to see another record? (Y/N): y [Enter]
Enter the number of the record you wish to see: 2 [Enter]
Description: Pliers
Units: 12
Do you want to see another record? (Y/N): y [Enter]
Enter the number of the record you wish to see: 0 [Enter]
Description: Wrench
Units: 20
Do you want to see another record? (Y/N): y [Enter]
Enter the number of the record you wish to see: 1 [Enter]
Description: Hammer
Units: 15
Do you want to see another record? (Y/N): y [Enter]
Enter the number of the record you wish to see: 3 [Enter]
Description: Screwdriver
Units: 25
Do you want to see another record? (Y/N): n [Enter]
```

As a last demonstration, the program in Code Listing I-4 shows how an existing record in the file can be overwritten with a new record.

Code Listing I-4 (ModifyRecord.java)

```
 1 import java.io.*;
 2 import java.util.Scanner;
 3
 4 /**
 5  * This program allows the user to modify records in the
 6  * Inventory.dat file.
 7  */
 8
 9 public class ModifyRecord
10 {
11    public static void main(String[] args) throws IOException
12    {
13       int recordNumber;     // Record number
14       int units;            // Units on hand
15       String again;         // Want to change another one?
16       String sure;          // Is the user sure?
17       String description;   // Item description
18       InventoryItem item;   // To reference an item
19
```

```
20        // Create a Scanner object for keyboard input.
21        Scanner keyboard = new Scanner(System.in);
22
23        // Open the file.
24        InventoryItemFile file =
25                 new InventoryItemFile("Inventory.dat");
26
27        // Report the number of records in the file.
28        System.out.println("The Inventory.dat file has " +
29                           file.getNumberOfRecords() + " records.");
30
31        // Get a record number from the user and
32        // allow the user to modify it.
33        do
34        {
35           // Get the record number.
36           System.out.print("Enter the number of the record " +
37                            "you wish to modify: ");
38           recordNumber = keyboard.nextInt();
39
40           // Consume the remaining newline.
41           keyboard.nextLine();
42
43           // Move the file pointer to that record number.
44           file.moveFilePointer(recordNumber);
45
46           // Read the record at that location.
47           item = file.readInventoryItem();
48
49           // Display the existing contents.
50           System.out.println("Existing data:");
51           System.out.println("\nDescription: " +
52                              item.getDescription());
53           System.out.println("Units: " + item.getUnits());
54
55           // Get the new data.
56           System.out.print("\nEnter the new description: ");
57           description = keyboard.nextLine();
58           System.out.print("Enter the number of units: ");
59           units = keyboard.nextInt();
60           keyboard.nextLine(); // Consume the remaining newline.
61
62           // Store the new data in the object.
63           item.setDescription(description);
64           item.setUnits(units);
65
```

```
66           // Make sure the user wants to save this data.
67           System.out.print("Are you sure you want to save " +
68                            "this data? (Y/N) ");
69           sure = keyboard.nextLine();
70           if (sure.charAt(0) == 'Y' || sure.charAt(0) == 'y')
71           {
72              // Move back to the record's starting position.
73              file.moveFilePointer(recordNumber);
74              // Save the new data.
75              file.writeInventoryItem(item);
76           }
77
78           // Ask the user whether to change another record.
79           System.out.print("\nDo you want to modify another " +
80                            "record? (Y/N): ");
81           again = keyboard.nextLine();
82        } while (again.charAt(0) == 'Y' || again.charAt(0) == 'y');
83
84        // Close the file.
85        file.close();
86     }
87 }
```

Program Output with Example Input Shown in Bold

```
The Inventory.dat file has 5 records.
Enter the number of the record you wish to modify: 3 [Enter]
Existing data:
Description: Screwdriver
Units: 25
Enter the new description: Duct Tape [Enter]
Enter the number of units: 30 [Enter]
Are you sure you want to save this data? (Y/N) y [Enter]
Do you want to modify another record? (Y/N): n [Enter]
```

In the example running of the program, record 3 was modified. We can run the `ReadInventoryFile` program in Code Listing I-3 again to verify that the record was changed. Here is the output of that program if we run it again.

Program Output with Example Input Shown in Bold (ReadInventoryFile.java)

```
The Inventory.dat file has 5 records.
Enter the number of the record you wish to see: 3 [Enter]
Description: Duct Tape
Units: 30
Do you want to see another record? (Y/N): n [Enter]
```