APPENDIX I

Working with Records and Random Access Files

Structuring Data into Fields and Fixed-Length Records

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 its contents 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 example in this appendix shows one way to do this.

First we will introduce the `InventoryItem` class shown in Code Listing I-1. An object of this class can represent an item that a company might have in its inventory. This class has two fields: `description`, a `String` that holds an item's description, and `units`, an `int` that holds the number of units on hand. The class also has the necessary accessor methods, mutator methods, and two constructors.

Code Listing I-1 (InventoryItem.java)

```
1  /**
2     InventoryItem class
3  */
4
5  public class InventoryItem
6  {
7     private String description;    // Item description
8     private int units;             // Units on hand
9
10    /**
11       This constructor assigns an empty string
12       to description and 0 to units.
13    */
14
15    public InventoryItem()
16    {
17       description = "";
18       units = 0;
19    }
20
21    /**
22       This constructor assigns values
23       to the description and units fields.
24       @param d The description.
25       @param u The units on hand.
26    */
27
28    public InventoryItem(String d, int u)
29    {
30       description = d;
31       units = u;
32    }
33
34    /**
35       The setDescription method assigns a string
36       to the description field.
37       @param d The string to assign to description.
38    */
39
40    public void setDescription(String d)
41    {
42       description = d;
43    }
44
45    /**
```

```
46       The setUnits method assigns a value
47       to the units field.
48       @param u The value to assign to units.
49    */
50
51    public void setUnits(int u)
52    {
53       units = u;
54    }
55
56    /**
57       The getDescription method returns the item's
58       description.
59       @return The description field.
60    */
61
62    public String getDescription()
63    {
64       return description;
65    }
66
67    /**
68       The getUnits method returns the number of
69       units on hand.
70       @return The units field.
71    */
72
73    public int getUnits()
74    {
75       return units;
76    }
77 }
```

The `InventoryItemFile` class shown in Code Listing I-2 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-2 **(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;
11    private RandomAccessFile inventoryFile;
12
13    /**
14       The constructor opens a random access file
15       for both reading and writing.
16       @param filename The name of the file.
17       @exception  FileNotFoundException When the file
18                   is not found.
19    */
20
21    public InventoryItemFile(String filename)
22                          throws FileNotFoundException
23    {
24       // Open the file for reading and writing.
25       inventoryFile =
26            new RandomAccessFile(filename, "rw");
27    }
28
29    /**
30       The writeInventoryItem method writes the contents
31       of an InventoryItem object to the file at the
32       current file pointer position.
33       @param item The InventoryItem object to write.
34       @exception IOException When a file error occurs.
35    */
36
37    public void writeInventoryItem(InventoryItem item)
38                                    throws IOException
39    {
40       // Get the item's description.
41       String str = item.getDescription();
42
43       // Write the description.
44       if (str.length() > 20)
45       {
46          // If there are more than 20 characters in the
47          // string, then write only the first 20.
48          for (int i = 0; i < 20; i++)
49             inventoryFile.writeChar(str.charAt(i));
50       }
51       else
52       {
53          // Write the description to the file.
54          inventoryFile.writeChars(str);
```

```
55          // Write enough spaces to pad it out
56          // to 20 characters.
57          for (int i = 0; i < (20 - str.length()); i++)
58             inventoryFile.writeChar(' ');
59       }
60
61       // Write the units to the file.
62       inventoryFile.writeInt(item.getUnits());
63    }
64
65    /**
66       The readInventoryItem method reads and returns
67       the record at the current file pointer position.
68       @return A reference to an InventoryItem object.
69       @exception IOException When a file error occurs.
70    */
71
72    public InventoryItem readInventoryItem()
73                                throws IOException
74    {
75       char[] charArray = new char[20];
76
77       // Read the description, character by character,
78       // from the file into the char array.
79       for (int i = 0; i < 20; i++)
80          charArray[i] = inventoryFile.readChar();
81
82       // Store the char array in a String.
83       String desc = new String(charArray);
84
85       // Trim any trailing spaces from the string.
86       desc.trim();
87
88       // Read the units from the file.
89       int u = inventoryFile.readInt();
90
91       // Create an InventoryItem object and initialize
92       // it with these values.
93       InventoryItem item =
94              new InventoryItem(desc, u);
95
96       // Return the object.
97       return item;
98    }
99
100   /**
101      The getByteNum method returns a record's
102      starting byte number.
```

```
103       @param recordNum The record number of the
104                        desired record.
105    */
106
107    private long getByteNum(long recordNum)
108    {
109       return RECORD_SIZE * recordNum;
110    }
111
112    /**
113       The moveFilePointer method moves the file
114       pointer to a specified record.
115       @param recordNum  The number of the record to
116                         move to.
117       @exception IOException When a file error occurs.
118    */
119
120    public void moveFilePointer(long recordNum)
121                             throws IOException
122    {
123       inventoryFile.seek(getByteNum(recordNum));
124    }
125
126    /**
127       The getNumberOfRecords method returns the number
128       of records stored in the file.
129       @return The number of records in the file.
130       @exception IOException When a file error occurs.
131    */
132
133    public long getNumberOfRecords() throws IOException
134    {
135       return inventoryFile.length() / RECORD_SIZE;
136    }
137
138    /**
139       The close method closes the file.
140       @exception IOException When a file error occurs.
141    */
142
143    public void close() throws IOException
144    {
145       inventoryFile.close();
146    }
147 }
```

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 filename as a `String`. This filename is used to open a random access file, referenced by the `inventoryFile` variable, for reading and writing.

By looking at the `writeInventoryItem` method, in lines 37 through 63, 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 41 the `description` field is retrieved and referenced by `str`, a local variable. Next, in lines 44 through 59, 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 only the first 20 are written. If the description has fewer than 20 characters, spaces are added to make up the difference. Next, in line 62, 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 written as 20 characters, so that's 40 bytes. The `units` field is written as an `int`, which uses 4 bytes. That makes a total record size of 44 bytes.

The `readInventoryItem` method in lines 72 through 98 reads a record from the file and returns an `InventoryItem` object containing the record's data. In line 75 the reference variable `charArray` is declared and a 20-element `char` array is created to hold the description. Then the code in lines 79 and 80 reads the 20 characters from the file and stores them in the array. Next, in line 83, a `String` object is created and the `char` array is passed as an argument. 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. The statement in line 86 trims any trailing spaces that might be in the string. Then the statement in line 89 reads the `units` field from the file and stores it in the `u` variable.

Now we can construct an `InventoryItem` object with the data we have read. This is done in lines 93 and 94. The last step, in line 97, is to return 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` (in lines 107 through 110) 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 120 through 124) 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 133 through 136. 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, which closes the file. The program in Code Listing I-3 shows a simple demonstration of this class. This program asks the user to enter data for five items, which are stored in an array of `InventoryItem` objects. The program then saves the contents of the array elements to a file.

Code Listing I-3 (CreateInventoryFile.java)

```
1  import java.io.*;
2  import java.util.Scanner;
3
4  /**
5     This program uses the InventoryFile class to create a
6     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-4 demonstrates how records can be randomly read from the file.

Code Listing I-4 (ReadInventoryFile.java)

```
1 import java.io.*;
2 import java.util.Scanner;
3
4 /**
5    This program displays specified records from
6    the 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-5 shows how an existing record in the file can be overwritten with a new record.

Code Listing I-5 (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-4 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]
```