Supplement: Case Study: AddressBook

For Introduction to Java Programming

By Y. Daniel Liang

This case study can be presented after Chapter 19, "Binary I/O."

Now let us use RandomAccessFile to create a useful project for storing and viewing an address book. The user interface of the program is shown in Figure 1. The *Add* button stores a new address at the end of the file. The *First*, *Next*, *Previous*, and *Last* buttons retrieve the first, next, previous, and last addresses from the file, respectively.

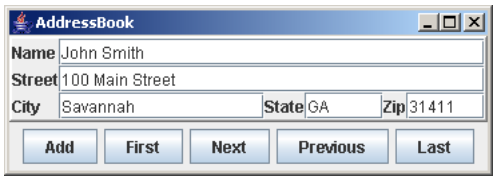

Figure 1

AddressBook stores and retrieves addresses from a file.

<Side Remark: fixed-length record>

Random access files are often used to process files of records. For convenience, fixed-length records are used in random access files so that a record can be located easily, as shown in Figure 2. A record consists of a fixed number of fields. A field can be a string or a primitive data type. A string in a fixed-length record has a maximum size. If a string is smaller than the maximum size, the rest of the string is padded with blanks.

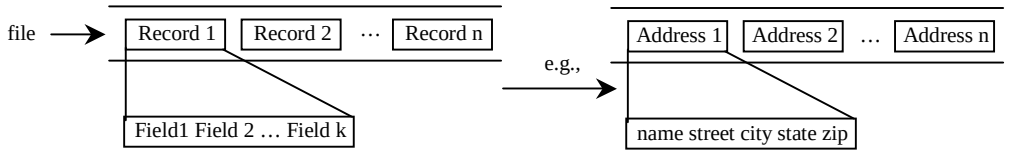

Figure 2

Random access files are often used to process files of fixed-length records.

Let **address.dat** be the file to store addresses. A RandomAccessFile for both read and write can be created using

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

Let each address consist of a name (32 characters), street (32 characters), city (20 characters), state (2 characters), and zip (5 characters). If the actual size of a field (e.g., name) is less than the fixed maximum size, fill it with blank characters. If the actual size of a field is greater than the fixed maximum size, truncate the string. Thus the total size of an address is 32 + 32 + 20 + 2 + 5 = 91 characters. Since each character occupies two bytes, one address takes 2 * 91 = 182 bytes. After an address record is read, the file pointer is 182 bytes ahead of the previous file pointer.

For convenience, Listing 1 contains two methods for reading and writing a fixed-length string.

Listing 1 FixedLengthStringIO.java

******PD: Please add line numbers in the following code*****
<Side Remark line 12: read characters>
<Side Remark line 23: fill string>
<Side Remark line 27: fill blank>
<Side Remark line 30: write string>

```
import java.io.*;

public class FixedLengthStringIO {
  /** Read fixed number of characters from a DataInput stream */
  public static String readFixedLengthString(int size,
      DataInput in) throws IOException {
    // Declare an array of characters
    char[] chars = new char[size];

    // Read fixed number of characters to the array
    for (int i = 0; i < size; i++)
      chars[i] = in.readChar();

    return new String(chars);
  }

  /** Write fixed number of characters to a DataOutput stream */
  public static void writeFixedLengthString(String s, int size,
      DataOutput out) throws IOException {
    char[] chars = new char[size];

    // Fill an array of characters from the string
    s.getChars(0, Math.min(s.length(), size), chars, 0);

    // Fill in blank characters in the rest of the array
    for (int i = Math.min(s.length(), size); i < chars.length; i++)
      chars[i] = ' ';

    // Create and write a new string padded with blank characters
    out.writeChars(new String(chars));
  }
```

}

The writeFixedLengthString(String s, int size, DataOutput out) method writes a string in a fixed size to a DataOutput stream. If the string is longer than the specified size, it is truncated (line 23); if it is shorter than the specified size, blanks are padded into it (lines 26–27). In any case, a new fixed-length string is written to a specified output stream. Since RandomAccessFile implements DataOutput, this method can be used to write a string to a RandomAccessFile. For example, Invoking writeFixedLengthString("John", 2, raf) actually writes "Jo" to the RandomAccessFile raf, since the size is 2. Invoking writeFixedLengthString("John", 6, raf) actually writes "John " to the RandomAccessFile raf, since the size is 6.

The readFixedLengthString(int size, InputOutput in) method reads a fixed number of characters from an InputStream and returns as a string. Since RandomAccessFile implements InputOutput, this method can be used to read a string from a writeFixedLengthString(String s, int size, DataOutput out).

The rest of the work can be summarized in the following steps:

1. Create the user interface.
2. Add a record to the file.
3. Read a record from the file.
4. Write the code to implement the button actions.

The program is shown in Listing 2.

Listing 2 AddressBook.java

******PD: Please add line numbers in the following code******
<Side Remark line 9: constant>
<Side Remark line 18: raf>
<Side Remark line 21: GUI component>
<Side Remark line 37: open file>
<Side Remark line 45: create UI>
<Side Remark line 102: register listener>
<Side Remark line 104: add address>
<Side Remark line 107: register listener>
<Side Remark line 110: first record>
<Side Remark line 117: register listener>
<Side Remark line 122: next address>
<Side Remark line 144: register listener>
<Side Remark line 150: last address>

<Side Remark line 160: first address>

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class AddressBook extends JFrame {
  // Specify the size of five string fields in the record
  final static int NAME_SIZE = 32;
  final static int STREET_SIZE = 32;
  final static int CITY_SIZE = 20;
  final static int STATE_SIZE = 2;
  final static int ZIP_SIZE = 5;
  final static int RECORD_SIZE =
    (NAME_SIZE + STREET_SIZE + CITY_SIZE + STATE_SIZE + ZIP_SIZE);

  // Access address.dat using RandomAccessFile
  private RandomAccessFile raf;

  // Text fields
  private JTextField jtfName = new JTextField(NAME_SIZE);
  private JTextField jtfStreet = new JTextField(STREET_SIZE);
  private JTextField jtfCity = new JTextField(CITY_SIZE);
  private JTextField jtfState = new JTextField(STATE_SIZE);
  private JTextField jtfZip = new JTextField(ZIP_SIZE);

  // Buttons
  private JButton jbtAdd = new JButton("Add");
  private JButton jbtFirst = new JButton("First");
  private JButton jbtNext = new JButton("Next");
  private JButton jbtPrevious = new JButton("Previous");
  private JButton jbtLast = new JButton("Last");

  public AddressBook() {
    // Open or create a random access file
    try {
      raf = new RandomAccessFile("address.dat", "rw");
    }
    catch(IOException ex) {
      System.out.print("Error: " + ex);
      System.exit(0);
    }

    // Panel p1 for holding labels Name, Street, and City
    JPanel p1 = new JPanel();
    p1.setLayout(new GridLayout(3, 1));
    p1.add(new JLabel("Name"));
    p1.add(new JLabel("Street"));
    p1.add(new JLabel("City"));

    // Panel jpState for holding state
    JPanel jpState = new JPanel();
```

```
    jpState.setLayout(new BorderLayout());
    jpState.add(new JLabel("State"), BorderLayout.WEST);
    jpState.add(jtfState, BorderLayout.CENTER);

    // Panel jpZip for holding zip
    JPanel jpZip = new JPanel();
    jpZip.setLayout(new BorderLayout());
    jpZip.add(new JLabel("Zip"), BorderLayout.WEST);
    jpZip.add(jtfZip, BorderLayout.CENTER);

    // Panel p2 for holding jpState and jpZip
    JPanel p2 = new JPanel();
    p2.setLayout(new BorderLayout());
    p2.add(jpState, BorderLayout.WEST);
    p2.add(jpZip, BorderLayout.CENTER);

    // Panel p3 for holding jtfCity and p2
    JPanel p3 = new JPanel();
    p3.setLayout(new BorderLayout());
    p3.add(jtfCity, BorderLayout.CENTER);
    p3.add(p2, BorderLayout.EAST);

    // Panel p4 for holding jtfName, jtfStreet, and p3
    JPanel p4 = new JPanel();
    p4.setLayout(new GridLayout(3, 1));
    p4.add(jtfName);
    p4.add(jtfStreet);
    p4.add(p3);

    // Place p1 and p4 into jpAddress
    JPanel jpAddress = new JPanel(new BorderLayout());
    jpAddress.add(p1, BorderLayout.WEST);
    jpAddress.add(p4, BorderLayout.CENTER);

    // Set the panel with line border
    jpAddress.setBorder(new BevelBorder(BevelBorder.RAISED));

    // Add buttons to a panel
    JPanel jpButton = new JPanel();
    jpButton.add(jbtAdd);
    jpButton.add(jbtFirst);
    jpButton.add(jbtNext);
    jpButton.add(jbtPrevious);
    jpButton.add(jbtLast);

    // Add jpAddress and jpButton to the frame
    add(jpAddress, BorderLayout.CENTER);
    add(jpButton, BorderLayout.SOUTH);

    jbtAdd.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        writeAddress();
      }
    });
```

```
    jbtFirst.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          if (raf.length() > 0) readAddress(0);
        }
        catch (IOException ex) {
          ex.printStackTrace();
        }
      }
    });
    jbtNext.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          long currentPosition = raf.getFilePointer();
          if (currentPosition < raf.length())
            readAddress(currentPosition);
        }
        catch (IOException ex) {
          ex.printStackTrace();
        }
      }
    });
    jbtPrevious.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          long currentPosition = raf.getFilePointer();
          if (currentPosition - 2 * RECORD_SIZE > 0)
            // Why 2 * 2 * RECORD_SIZE? See the follow-up remarks
            readAddress(currentPosition - 2 * 2 * RECORD_SIZE);
          else
            readAddress(0);
        }
        catch (IOException ex) {
          ex.printStackTrace();
        }
      }
    });
    jbtLast.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          long lastPosition = raf.length();
          if (lastPosition > 0)
            // Why 2 * RECORD_SIZE? See the follow-up remarks
            readAddress(lastPosition - 2 * RECORD_SIZE);
        }
        catch (IOException ex) {
          ex.printStackTrace();
        }
      }
    });

    // Display the first record if exists
    try {
      if (raf.length() > 0) readAddress(0);
```

```
    }
    catch (IOException ex) {
      ex.printStackTrace();
    }
  }

  /** Write a record at the end of the file */
  public void writeAddress() {
    try {
      raf.seek(raf.length());
      FixedLengthStringIO.writeFixedLengthString(
        jtfName.getText(), NAME_SIZE, raf);
      FixedLengthStringIO.writeFixedLengthString(
        jtfStreet.getText(), STREET_SIZE, raf);
      FixedLengthStringIO.writeFixedLengthString(
        jtfCity.getText(), CITY_SIZE, raf);
      FixedLengthStringIO.writeFixedLengthString(
        jtfState.getText(), STATE_SIZE, raf);
      FixedLengthStringIO.writeFixedLengthString(
        jtfZip.getText(), ZIP_SIZE, raf);
    }
    catch (IOException ex) {
      ex.printStackTrace();
    }
  }

  /** Read a record at the specified position */
  public void readAddress(long position) throws IOException {
    raf.seek(position);
    String name = FixedLengthStringIO.readFixedLengthString(
      NAME_SIZE, raf);
    String street = FixedLengthStringIO.readFixedLengthString(
      STREET_SIZE, raf);
    String city = FixedLengthStringIO.readFixedLengthString(
      CITY_SIZE, raf);
    String state = FixedLengthStringIO.readFixedLengthString(
      STATE_SIZE, raf);
    String zip = FixedLengthStringIO.readFixedLengthString(
      ZIP_SIZE, raf);

    jtfName.setText(name);
    jtfStreet.setText(street);
    jtfCity.setText(city);
    jtfState.setText(state);
    jtfZip.setText(zip);
  }

  public static void main(String[] args) {
    AddressBook frame = new AddressBook();
    frame.pack();
    frame.setTitle("AddressBook");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
```

```
}
```

A random access file, **address.dat**, is created to store address information if the file does not yet exist (line 37). If it already exists, the file is opened. A random file object, raf, is used for both write and read operations. The size of each field in the record is fixed and therefore defined as a constant in lines 9–15.

The user interface is created in lines 44–100. The listeners are registered in lines 102–156. When the program starts, it displays the first record, if it exists, in lines 159–164.

The writeAddress() method sets the file pointer to the end of the file (line 170) and writes a new record to the file (lines 171–180).

The readAddress() method sets the file pointer at the specified position (line 189) and reads a record from the file (lines 190–199). The record is displayed in lines 201–205.

To add a record, you need to collect the address information from the user interface and write the address into the file (line 104).

The code to process button events is implemented in lines 102–156. For the *First* button, read the record from position 0 (line 110). For the *Next* button, read the record from the current file pointer (line 122). When a record is read, the file pointer is moved 2 * RECORD_SIZE number of bytes ahead of the previous file pointer. For the *Previous* button, you need to display the record prior to the one being displayed now. You have to move the file pointer two records before the current file pointer (line 135). For the *Last* button, read the record from the position at raf.length() - 2 * RECORD_SIZE.