UCLA PIC 20A
Java Programming

- **Instructor:** Ivo Dinov,
  Asst. Prof. In Statistics, Neurology and Program in Computing
- **Teaching Assistant:** Yon Seo Kim, PIC

University of California, Los Angeles, Summer 2002
http://www.stat.ucla.edu/~dinov/

Chapter 9 – File Input / Output

Introduction

- Storage
  - Data in arrays and variables temporary
    - Lost when program ends
  - Files
    - Long term data storage
    - Persistent data
  - Secondary storage devices
    - Store files
    - Magnetic disks, optical disks, magnetic tape
- This chapter
  - File processing and stream input/output (I/O)features

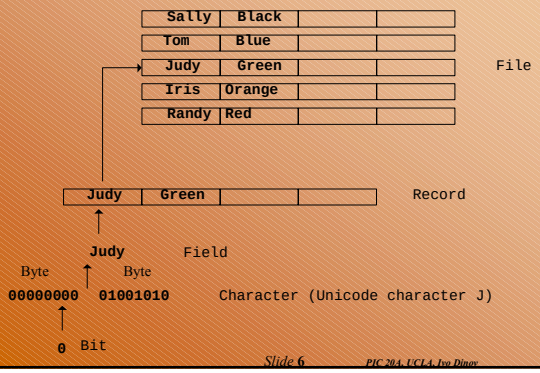
Data Hierarchy

- Data storage on computers
  - Smallest data unit: the bit (binary digit)
    - Value of `0` or `1`
    - All data ultimately represented as bits
    - Computer circuits can examine bits, set values, reverse them
  - Bits cumbersome for humans
    - We use characters - digits, symbols, letters
    - Character set - set of all characters on a computer
  - Byte
    - 8 bits
    - 2 bytes (16 bits) used to represent a character (Unicode)
  - Programmers write programs using characters
    - Computers process characters as patterns of bits

Data Hierarchy

- Data hierarchy
  - Field
    - Group of characters conveying meaning
    - Example: Employee's name
  - Record
    - Group of related fields
    - Represented as a `class` with instance variables
    - Example: Name, address, age, phone number of employee
  - File
    - Group of related records
    - Example: Info about many employees
  - Database
    - Group of related files
    - Example: payroll file, inventory file, accounts file

Data Hierarchy



Data Hierarchy

- Record key
  - Easy retrieval of records
  - One field chosen as record key
    - Identifies record
    - I.e., employee identification number
- Organizing records
  - Sequential file
    - Records in order by key

Files and Streams

- **Sequential files**
  - Java sees as *stream* of bytes
  - File ends with *end-of-file marker* or specific byte number
- Opening files
  - When opened, object created
    - Stream associated with object
  - Stream objects automatically created for us:
    - `System.in` - standard input (from keyboard)
    - `System.out` - standard output (to screen)
    - `System.err` - standard error (to screen)
    - Can be redirected (i.e., to disk instead of to screen)

Files and Streams

- File processing
  - Import `java.io`
    - Definitions of stream classes `FileInputStream` and `FileOutputStream`
    - Inherit from `InputStream` and `OutputStream`
  - Java has many input/output classes
    - In this section, overview each
    - See how they relate
  - `InputStream` and `OutputStream`
    - `abstract` classes
    - Define methods for input and output (overridden in subclasses)
  - `FileInputStream` and `FileOutputStream`
    - File input/output

Files and Streams

- File processing
  - Pipes
    - Synchronized communication channels between threads
    - Sending thread writes to `PipedOutputStream`
    - Receiving thread reads from `PipedInputStream`
  - `PrintStream`
    - Output to screen
    - `System.out` and `System.err` are `PrintStream`s
- Reading raw bytes
  - Fast but crude
  - Usually read groups of bytes that form `int`, `double`, etc.

Files and Streams

- Random access file
  - Class `RandomAccessFile`
  - Read from anywhere in file
    - In sequential file, must read next consecutive set of data
  - Rapid access
- Binary data I/O
  - `DataInputStreams`
    - Read binary data from InputStream
    - Methods `read`, `readByte`, `readChar`, `readDouble`...
  - `DataOutputStreams`
    - Write binary data to `OutputStream`
    - Methods `write`, `writeChar`, `writeInt`...

Files and Streams

- Buffering
  - Improves I/O performance
    - I/O is slow compared to processor
  - Buffer stores data of many I/O operations
    - When full, sends data
  - Can be explicitly flushed (forced to send data)
- Object I/O
  - When object's instance variables in a file, lose object's type information
    - Only have pure data
  - `ObjectInputStream` and `ObjectOutputStream`
    - Read/write entire object to file

Files and Streams

- Class **`File`**
  - Obtain information about file or directory

Creating a Sequential-Access File

- Java imposes no file structure
  - No notion of "records" in a file
  - Programmer must provide file structure
- Simple program
  - Input client account number (key), name, and balance
  - Store data to file
  - Classes
    - **`BankUI`** - has GUI components for program
      - Reused by many programs in this chapter
    - **`BankAccountRecord`**
      - Holds client information, reused in chapter
    - **`CreateSequentialFile`**
      - Driver, creates a sequential file

Creating a Sequential-Access File

- Class BankUI (extends **`JPanel`**)
  - Two **`JButton`**s
  - Arrays of **`JLabel`**s and **`JTextField`**s
    - Number set with constructor
    - Default value of four (no argument constructor)
  - Methods
    - Manipulate text of **`JTextField`**s
      - **`getFieldValues`**
      - **`setFieldValues`**
      - **`clearFields`**
    - Return GUI components (another program can add **`actionListeners`**, etc)
      - **`getFields`**
      - **`getDoTask`**
      - **`getDoTask2`**


```
1  // BankUI.java
2  // A reusable GUI for the examples in this chapter.
3  package bankAccount;
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class BankUI extends JPanel {
8     protected final static String names[] = { "Account number",
9        "First name", "Last name", "Balance",
10       "Transaction Amount" };
11    protected JLabel labels[];
12    protected JTextField fields[];
13    protected JButton doTask, doTask2;
14    protected JPanel innerPanelCenter, innerPanelSouth;
15    protected int size = 4;
16    public static final int ACCOUNT = 0, FIRST = 1, LAST = 2,
17                            BALANCE = 3, TRANSACTION = 4;
18
19    public BankUI()
20    {
21       this( 4 );
22    }
23
24    public BankUI( int mySize )
25    {
26       size = mySize;
27       labels = new JLabel[ size ];
28       fields = new JTextField[ size ];
29
```

Default constructor with argument of 4.

- 1. Class `BankUI (extends JPanel)`
- 1.1 Declarations
- 1.2 Constructors


```
30       for ( int i = 0; i < labels.length; i++ )
31          labels[ i ] = new JLabel( names[ i ] );
32
33       for ( int i = 0; i < fields.length; i++ )
34          fields[ i ] = new JTextField();
35
36       innerPanelCenter = new JPanel();
37       innerPanelCenter.setLayout( new GridLayout( size, 2 ) );
38
39       for ( int i = 0; i < size; i++ ) {
40          innerPanelCenter.add( labels[ i ] );
41          innerPanelCenter.add( fields[ i ] );
42       }
43
44       doTask = new JButton();
45       doTask2 = new JButton();
46       innerPanelSouth = new JPanel();
47       innerPanelSouth.add( doTask2 );
48       innerPanelSouth.add( doTask );
49
50       setLayout( new BorderLayout() );
51       add( innerPanelCenter, BorderLayout.CENTER );
52       add( innerPanelSouth, BorderLayout.SOUTH );
53       validate();
54    }
55
56    public JButton getDoTask() { return doTask; }
57
58    public JButton getDoTask2() { return doTask2; }
59
```

Create **`JPanel`**s to attach components. Attach to **`BankUI`**.

- 1.3 Initialize arrays
- 1.4 add
- 2. Class methods


```
60    public JTextField[] getFields() { return fields; }
61
62    public void clearFields()
63    {
64       for ( int i = 0; i < size; i++ )
65          fields[ i ].setText( "" );
66    }
67
68    public void setFieldValues( String s[] )
69       throws IllegalArgumentException
70    {
71       if ( s.length != size )
72          throw new IllegalArgumentException( "There must be "
73             + size + " Strings in the array" );
74
75       for ( int i = 0; i < size; i++ )
76          fields[ i ].setText( s[ i ] );
77    }
78
79    public String[] getFieldValues()
80    {
81       String values[] = new String[ size ];
82
83       for ( int i = 0; i < size; i++ )
84          values[ i ] = fields[ i ].getText();
85
86       return values;
87    }
88 }
```

- 2. Class methods

Creating a Sequential-Access File

- Class **BankAccountRecord**

```
93 import java.io.Serializable;

95 public class BankAccountRecord implements Serializable {
```

  - Allows **BankAccountRecord** to be used with **ObjectInputStream**s and **ObjectOutputStream**s
    - Read/write entire objects
  - Instance variables
    - **account**, **firstName**, **lastName**, **balance**
  - **public** set and get methods


```
90 // BankAccountRecord.java
91 // A class that represents one record of information.
92 package bankAccount;
93 import java.io.Serializable;
94
95 public class BankAccountRecord implements Serializable {
96    private int account;
97    private String firstName;
98    private String lastName;
99    private double balance;
100
101   public BankAccountRecord()
102   {
103      this( 0, "", "", 0.0 );
104   }
105
106   public BankAccountRecord( int acct, String first,
107                             String last, double bal )
108   {
109      setAccount( acct );
110      setFirstName( first );
111      setLastName( last );
112      setBalance( bal );
113   }
114
115   public void setAccount( int
116   {
117      account = acct;
118   }
119
```

Allows objects to be used with **ObjectOutputStream** and **ObjectInputStream**.

- 1. implements Serializable
- 1.1 Instance variables
- 1.2 Constructors
- 2. Class methods


```
120   public int getAccount() { return account; }
121
122   public void setFirstName( String first )
123   {
124      firstName = first;
125   }
126
127   public String getFirstName() { return firstName; }
128
129   public void setLastName( String last )
130   {
131      lastName = last;
132   }
133
134   public String getLastName() { return lastName; }
135
136   public void setBalance( double bal )
137   {
138      balance = bal;
139   }
140
141   public double getBalance() { return balance; }
142}
```

- 2. Class methods

Creating a Sequential-Access File

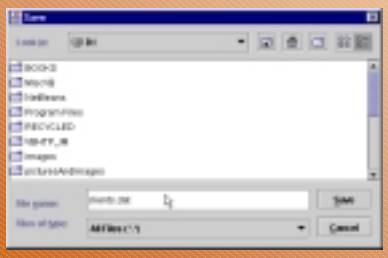
- Class **CreateSequentialFile**
  - Class **JFileChooser** (**javax.swing**)
    - Used to select files

```
213          JFileChooser fileChooser = new JFileChooser();
214          fileChooser.setFileSelectionMode(
215             JFileChooser.FILES_ONLY );
```

  - Create object
  - Method **setFileSelectionMode(CONSTANT)**
    - **static** constant of class **JFileChooser**
    - **FILES_ONLY**, **FILES_AND_DIRECTORIES**, **DIRECTORIES_ONLY**

Creating a Sequential-Access File

Save JFileChooser dialog box.



Creating a Sequential-Access File

```
217          int result = fileChooser.showSaveDialog( this );
```

- Method **showSaveDialog(parent)**
  - parent - Determines position on screen (**null** is center)
- Modal dialog
  - User cannot interact with other windows
- User selects file, clicks **Save** (or **Cancel**)
  - Returns integer representing choice
  - **static** constant **CANCEL_OPTION**

```
223          File fileName = fileChooser.getSelectedFile();
```

- **getSelectedFile**
  - Returns a **File** object (contains file info)
  - **File** object does not open file

Creating a Sequential-Access File

- Opening files
  - Files opened using **`FileInputStream`** and **`FileOutputStream`** objects
    - In this case, file opened for output
  - **`FileOutputStream`** constructor
    - Takes **`File`** reference
    - Files truncated (existing contents discarded)

```
235                         new FileOutputStream( fileName ) );
```


Creating a Sequential-Access File

- Opening files
  - Need to use **`ObjectOutputStream`**
    - Want to write objects
  - Chaining stream objects
    - Adding services of one stream to another
  - To chain **`ObjectOutputStream`** to **`FileOutputStream`**
    - Pass **`FileOutputStream`** object to constructor

```
234                output = new ObjectOutputStream(
235                            new FileOutputStream( fileName ) );
```

  - **`IOException`** occurs if error opening file
  - If no exception, file is open
    - **`output`** now used to write to file

Creating a Sequential-Access File

- Writing objects
  - Program gathers info from **`JTextField`**s

```
280                output.writeObject( record );
```

  - Method **`writeObject( object )`**
    - Writes **`object`** to file

```
281                output.flush();
```

  - Method **`flush`**
    - Ensures any data stored in memory written

Creating a Sequential-Access File

- Closing files
  - Close files when done
  - When using chaining, outermost object used to close file
    - In this case, **`ObjectOutputStream`**

```
250                output.close();
```



```
145// CreateSequentialFile.java
146// Demonstrating object output with class ObjectOutputStre
147// The objects are written sequentially to a file.
148import java.io.*;
149import java.awt.*;
150import java.awt.event.*;
151import javax.swing.*;
152import bankAccount;
153import bankAccount.BankAccountRecord;
154
155public class CreateSequentialFile extends JFrame {
156   private ObjectOutputStream output;
157   private BankUI userInterface;
158   private JButton enter, open;
159
160   public CreateSequentialFile()
161   {
162      super( "Creating a Sequential File of Objects" );
163
164      getContentPane().setLayout( new BorderLayout() );
165      userInterface = new BankUI();
166
167      enter = userInterface.getDoTask();
168      enter.setText( "Enter" );
169      enter.setEnabled( false );  // disable button to start
170      enter.addActionListener(
171         new ActionListener() {
172            public void actionPerformed( ActionEvent e )
173            {
174               addRecord();
175            }
```

Import the interface and data record class.

Create new **`BankUI`** object (default 4 labels and text fields).

Customize buttons in **`BankUI`** object, add event handlers and set text.

- 1. `import`
- 1.1 Declarations
- 1.2 Constructor
- 1.3 Set up GUI
- 1.4 `BankUI`
- 1.5 `getDoTask`


```
176         }
177      );
178
179      addWindowListener(
180         new WindowAdapter() {
181            public void windowClosing( WindowEvent e )
182            {
183               if ( output != null ) {
184                  addRecord();
185                  closeFile();
186               }
187               else
188                  System.exit( 0 );
189            }
190         }
191      );
192      open = userInterface.getDoTask2();
193
194      open.setText( "Save As" );
195      open.addActionListener(
196         new ActionListener() {
197
198            public void actionPerformed( ActionEvent e )
199            {
200               openFile();
201            }
202         }
203      );
204      getContentPane().add( userInterface,
205                            BorderLayout.CENTER );
```

Add **`BankUI`** object to content pane. Remember, **`BankUI`** is a **`JPanel`**.

```
206
207    setSize( 300, 200 );
208    show();
209 }
210
211 private void openFile()
212 {
213    JFileChooser fileChooser = new JFileChooser();
214    fileChooser.setFileSelectionMode(
215       JFileChooser.FILES_ONLY );
216
217    int result = fileChooser.showSaveDialog( this );
218
219    // user clicked Cancel button on dialog
220    if ( result == JFileChooser.CANCEL_OPTION )
221       return;
222
223    File fileName = fileChooser.getSelectedFile();
224
225    if ( fileName == null ||
226         fileName.getName().equals( "" ) )
227       JOptionPane.showMessageDialog( this,
228          "Invalid File Name",
229          "Invalid File Name",
230          JOptionPane.ERROR_MESSAGE );
```

Create a **JFileChooser** object, display dialog. If cancel selected, return from method.

Assign selected file to **fileName**, check if **null**.

2.2 setFileSelection Mode


```
231    else {
232       // Open the file
233       try {
234          output = new ObjectOutputStream(
235                   new FileOutputStream( fileName ) );
236          open.setEnabled( false );
237          enter.setEnabled( true );
238       }
239       catch ( IOException e ) {
240          JOptionPane.showMessageDialog( this,
241             "Error Opening File", "Error",
242             JOptionPane.ERROR_MESSAGE );
243       }
244    }
245 }
246
247 private void closeFile()
248 {
249    try {
250       output.close();
251
252       System.exit( 0 );
253    }
254    catch( IOException ex ) {
255       JOptionPane.showMessageDialog( this,
256          "Error closing file",
257          "Error", JOptionPane.ERROR_MESSAGE );
258       System.exit( 1 );
259    }
260 }
261
```

Open file for output, chain with **ObjectOutputStream** object.

Check for thrown exceptions.

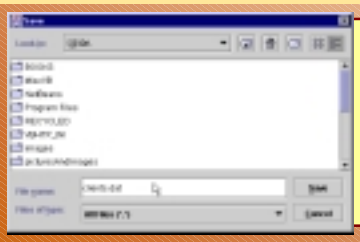
Call **close** for outermost chained object.

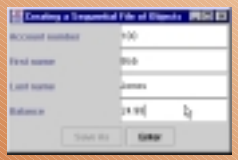

```
262 public void addRecord()
263 {
264    int accountNumber = 0;
265    BankAccountRecord record;
266    String fieldValues[] = userInterface.getFieldValues();
267
268    // If the account field value is not empty
269    if ( ! fieldValues[ 0 ].equals( "" ) ) {
270       // output the values to the file
271       try {
272          accountNumber =
273             Integer.parseInt( fieldValues[ 0 ] );
274
275          if ( accountNumber > 0 ) {
276             record = new BankAccountRecord(
277                accountNumber, fieldValues[ 1 ],
278                fieldValues[ 2 ],
279                Double.parseDouble( fieldValues[ 3 ] ) );
280             output.writeObject( record );
281             output.flush();
282          }
283
284          // clear the TextFields
285          userInterface.clearFields();
286       }
287       catch ( NumberFormatException nfe ) {
288          JOptionPane.showMessageDialog( this,
289             "Bad account number or balance",
290             "Invalid Number Format",
291             JOptionPane.ERROR_MESSAGE );
292       }
```

Create new record, set instance variables (**getFieldValues**).

Write object to file using **object.writeObject**.


```
293       catch ( IOException io ) {
294          closeFile();
295       }
296    }
297 }
298
299 public static void main( String args[] )
300 {
301    new CreateSequentialFile();
302 }
303}
```





- Program Output

Sample Data			
100	Bob	Jones	24.98
200	Steve	Doe	-345.67
300	Pam	White	0.00
400	Sam	Stone	-42.16
500	Sue	Rich	224.62

Reading Data from a Sequential-Access File

- In this section
  - Read data sequentially from a file
  - Read records created in previous section
- Opening files
  - Class **JFileChooser**
    - **showOpenDialog**

```
68    JFileChooser fileChooser = new JFileChooser();
72    int result = fileChooser.showOpenDialog( this );
```

Reading Data from a Sequential-Access File

- Opening files
  - Need to be able to read objects
    - Open file for input using **`FileInputStream`**
    - Chain with **`ObjectInputStream`**

```
78      File fileName = fileChooser.getSelectedFile();
89           input = new ObjectInputStream(
90                       new FileInputStream( fileName ) );
```


Reading Data from a Sequential-Access File

- Reading Objects

```
104          BankAccountRecord record;
108             record = ( BankAccountRecord ) input.readObject();
```

  - **`ObjectOutputStream`** method **`readObject`**
    - Returns an **`Object`** (must be cast)
    - **`EndOfFileException`** thrown if end-of-file marker reached
- Retrieving data
  - Programs normally read from beginning to end
  - **`FileInputStream`** cannot reposition pointer
    - **`RandomAccessFile`** can


```
1  // ReadSequentialFile.java
2  // This program reads a file of objects sequentially
3  // and displays each record.
4  import java.io.*;
5  import java.awt.*;
6  import java.awt.event.*;
7  import javax.swing.*;
8  import bankAccount.*;
9
10 public class ReadSequentialFile extends JFrame {
11    private ObjectInputStream input;
12    private BankUI userInterface;
13    private JButton nextRecord, open;
14
15    // Constructor -- initialize the Frame
16    public ReadSequentialFile()
17    {
18       super( "Reading a Sequential File of Objects" );
19
20       getContentPane().setLayout( new BorderLayout() );
21       userInterface = new BankUI();
22       nextRecord = userInterface.getDoTask();
23       nextRecord.setText( "Next Record" );
24       nextRecord.setEnabled( false );
25
26       nextRecord.addActionListener(
27          new ActionListener() {
28             public void actionPerformed( ActionEvent e )
29             {
30                readRecord();
31             }
```

Import **`BankUI`** and **`BankAccountRecord`**.

Customize buttons.


```
32          }
33       );
34
35       addWindowListener(
36          new WindowAdapter() {
37             public void windowClosing( WindowEvent e )
38             {
39                if ( input != null )
40                   closeFile();
41
42
43                System.exit( 0 );
44             }
45          }
46       );
47       open = userInterface.getDoTask2();
48
49       open.setText( "Open File" );
50       open.addActionListener(
51          new ActionListener() {
52             public void actionPerformed( ActionEvent e )
53             {
54                openFile();
55             }
56          }
57       );
58
59       getContentPane().add( userInterface,
60                             BorderLayout.CENTER );
```



```
61       pack();
62       setSize( 300, 200 );
63       show();
64    }
65
66    private void openFile()
67    {
68       JFileChooser fileChooser = new JFileChooser();
69
70       fileChooser.setFileSelectionMode(
71          JFileChooser.FILES_ONLY );
72       int result = fileChooser.showOpenDialog( this );
73
74       // user clicked Cancel button on dialog
75       if ( result == JFileChooser.CANCEL_OPTION )
76          return;
77
78       File fileName = fileChooser.getSelectedFile();
79
80       if ( fileName == null ||
81            fileName.getName().equals( "" ) )
82          JOptionPane.showMessageDialog( this,
83             "Invalid File Name",
84             "Invalid File Name",
85             JOptionPane.ERROR_MESSAGE );
86       else {
87          // Open the file
88          try {
89             input = new ObjectInputStream(
90                         new FileInputStream( fileName ) );
```

Use method **`showOpenDialog`** to open a file.

Use chaining with **`ObjectInputStream`**, to read of objects.

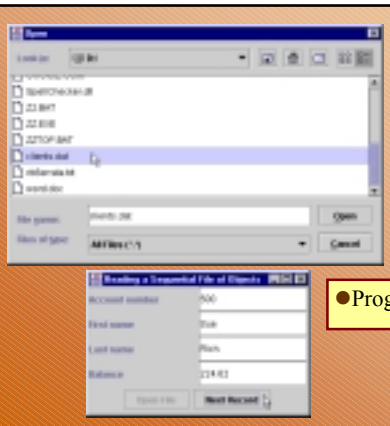

```
91             open.setEnabled( false );
92             nextRecord.setEnabled( true );
93          }
94          catch ( IOException e ) {
95             JOptionPane.showMessageDialog( this,
96                "Error Opening File", "Error",
97                JOptionPane.ERROR_MESSAGE );
98          }
99       }
100   }
101
102   public void readRecord()
103   {
104      BankAccountRecord record;
105
106      // input the values from the file
107      try {
108         record = ( BankAccountRecord ) input.readObject();
109         String values[] = {
110            String.valueOf( record.getAccount() ),
111            record.getFirstName(),
112            record.getLastName(),
113            String.valueOf( record.getBalance() ) };
114         userInterface.setFieldValues( values );
115      }
116      catch ( EOFException eofex ) {
117         nextRecord.setEnabled( false );
118         JOptionPane.showMessageDialog( this,
119            "No more records in file",
120            "End of File", JOptionPane.ERROR_MESSAGE );
121      }
```

`readObject` returns an **`Object`**. Cast to **`BankAccountRecord`**.

Catch the end of file exception.

```
122     catch ( ClassNotFoundException cnfex ) {
123        JOptionPane.showMessageDialog( this,
124           "Unable to create object",
125           "Class Not Found", JOptionPane.ERROR_MESSAGE );
126     }
127     catch ( IOException ioex ) {
128        JOptionPane.showMessageDialog( this,
129           "Error during read from file",
130           "Read Error", JOptionPane.ERROR_MESSAGE );
131     }
132  }
133
134  private void closeFile()
135  {
136     try {
137        input.close();
138        System.exit( 0 );
139     }
140     catch ( IOException e ) {
141        JOptionPane.showMessageDialog( this,
142           "Error closing file",
143           "Error", JOptionPane.ERROR_MESSAGE );
144        System.exit( 1 );
145     }
146  }
147
148  public static void main( String args[] )
149  {
150     new ReadSequentialFile();
151  }
152}
```

Call **`close`** with the outer object (**`ObjectInputStream`**).



- Program Output

Updating Sequential-Access Files

- Sequential access file
  - Modification may destroy other data
  - Suppose we want to change White's name to Worthington:

```
300 White 0.00 400 Jones 32.87   (old data in file)

300 Worthington 0.00
300 White 0.00 400 Jones 32.87
300 Worthington 0.00ones 32.87
```

Data gets overwritten

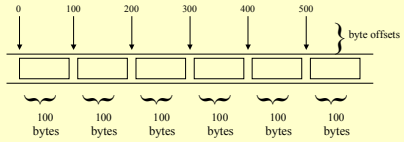
- Modification possible, but awkward

Random Access Files

- Random access files
  - Instant access to records (direct access)
    - Access records without searching through other records
  - Data can be inserted without destroying other data
  - Data previously stored can be updated or deleted without overwriting

Random Access Files

- Implemented using fixed length records
  - Like railroad train with many cars
    - Some empty, some full
  - Sequential files do not have fixed length records



Here, records are 100 bytes each (and will not overwrite each other).

Creating a Random-Access File

- **`RandomAccessFile`**
  - Have capabilities of **`DataInputStream`** and **`DataOutputStream`**
    - Can read/write raw binary data
  - When **`RandomAccessFile`** associated with a file
    - Data read/written at location of file position pointer
    - All data treated as primitive data types
      - Have fixed sizes (**`double`** is 8 bytes, **`int`** is 4, etc.)
  - This allows fixed length records
    - Class objects have same instance variables
  - In sequential file, characters written to file
    - 123456789 takes 9 bytes for sequential files
    - Takes 4 bytes for random access files (integer)

Creating a Random-Access File

- Problem statement
  - Create transaction processing program
  - Up to 100 fixed-length records
  - Each record
    - Account number (key)
    - Last name
    - First name
    - Balance
  - Update, insert, and delete accounts
  - Easiest to use random access file

Creating a Random-Access File

- Create class **`Record`**
  - Inherit from class **`BankAccount`**
    - Gets instance variables and methods

```
23   public void read( RandomAccessFile file ) throws
24   {
25      setAccount( file.readInt() );
```

  - **`RandomAccessFile`** method **`readInt`**
    - Reads and returns an integer from file
    - Also, **`readDouble`**

Creating a Random-Access File

```
31   private String padName( RandomAccessFile f )
32      throws IOException
33   {
34      char name[] = new char[ 15 ], temp;
35
36      for ( int i = 0; i < name.length; i++ ) {
37         temp = f.readChar();
38         name[ i ] = temp;
39      }
40
41      return new String( name ).replace( '\0', ' ' );
42   }
```

- Use fixed length (15 character) arrays for names
  - **`RandomAccessFile`** method **`readChar`**
- If less than 15 characters, null byte **`'\0'`** inserted
  - Swing components cannot display null bytes, so convert to spaces
  - **`replace( charToReplace, replacement )`**

Creating a Random-Access File

- Writing to a random access file

```
47      file.writeInt( getAccount() );
```

- Method **`writeInt`**, **`writeDouble`** similar
- For names
  - Need 15 characters for first and last name
  - Ensure all records have same size

```
56      StringBuffer buf = null;
59         buf = new StringBuffer( name );
63      buf.setLength( 15 );
64      f.writeChars( buf.toString() );
```

- Use method **`writeChars`**
  - Writes a **`String`** (as a sequence of **`char`**s)


```
1  // Record.java
2  // Record class for the RandomAccessFile programs.
3  package bankAccount;
4  import java.io.*;
5  import bankAccount.BankAccountRecord;
6
7  public class Record extends BankAccountRecord {
8
9     public Record()
10    {
11       this( 0, "", "", 0.0 );
12    }
13
14    public Record( int acct, String first,
15                   String last, double bal )
16    {
17
18
19       super( acct, first, last, bal );
20    }
21
22    // Read a record from the specified RandomAccessFile
23    public void read( RandomAccessFile file ) throws IOException
24    {
25       setAccount( file.readInt() );
26       setFirstName( padName( file ) );
27       setLastName( padName( file ) );
28       setBalance( file.readDouble() );
29    }
30
```

Read integer from **`RandomAccessFile`** object.


```
31   private String padName( RandomAccessFile f )
32      throws IOException
33   {
34      char name[] = new char[ 15 ], temp;
35
36      for ( int i = 0; i < name.length; i++ ) {
37         temp = f.readChar();
38         name[ i ] = temp;
39      }
40
41      return new String( name ).replace( '\0', ' ' );
42   }
43
44   // Write a record to the specified RandomAccessFile
45   public void write( RandomAccessFile file ) throws
46   {
47      file.writeInt( getAccount() );
48      writeName( file, getFirstName() );
49      writeName( file, getLastName() );
50      file.writeDouble( getBalance() );
51   }
52
```

Read name from file, ensure it has 15 characters. Replace null character with space.

Write an integer to **`RandomAccessFile`** object.

- 2.2 `padName`
- 3. `write`
- 3.1 `writeInt`

```
53    private void writeName( RandomAccessFile f, String name )
54       throws IOException
55    {
56       StringBuffer buf = null;
57
58       if ( name != null )
59          buf = new StringBuffer( name );
60       else
61          buf = new StringBuffer( 15 );
62
63       buf.setLength( 15 );
64       f.writeChars( buf.toString() );
65    }
66
67    // NOTE: This method contains a hard coded value for the
68    // size of a record of information.
69    public static int size() { return 72; }
70 }
```

Make name proper length, use **`writeChars`**.

Creating a Random-Access File

- Class **`CreateRandFile`**
  - Create an empty random access file with 100 records
    - **`0`** account number, **`null`** for names, **`0.0`** balance (default constructor for **`Record`**)

```
19       JFileChooser fileChooser = new JFileChooser();
22       int result = fileChooser.showSaveDialog( null );
28       File fileName = fileChooser.getSelectedFile();
```

  - Select file as before

```
39             file = new RandomAccessFile( fileName, "rw" );
```

  - **`RandomAccessFile`** constructor
    - Pass file name and file open mode
      - **`"r"`** - reading only
      - **`"rw"`** - reading and writing


```
1  // CreateRandFile.java
2  // This program creates a random access file sequentially
3  // by writing 100 empty records to disk.
4  import bankAccount.Record;
5  import java.io.*;
6  import javax.swing.*;
7
8  public class CreateRandomFile {
9     private Record blank;
10    private RandomAccessFile file;
11
12    public CreateRandomFile()
13    {
14       blank = new Record();
15       openFile();
16    }
17   private void openFile()
18    {
19       JFileChooser fileChooser = new JFileChooser();
20                   fileChooser.setFileSelectionMode(
21                               JFileChooser.FILES_ONLY );
22       int result = fileChooser.showSaveDialog( null );
23
24       // user clicked Cancel button on dialog
25       if ( result == JFileChooser.CANCEL_OPTION )
26          return;
27
28       File fileName = fileChooser.getSelectedFile();
29
```

Create a blank **`Record`** (default constructor), to be written to the file.


```
30       if ( fileName == null ||
31            fileName.getName().equals( "" ) )
32          JOptionPane.showMessageDialog( null,
33             "Invalid File Name",
34             "Invalid File Name",
35             JOptionPane.ERROR_MESSAGE );
36       else {
37          // Open the file
38          try {
39             file = new RandomAccessFile( fileName, "rw" );
40
41             for ( int i = 0; i < 100; i++ )
42                blank.write( file );
43
44             System.exit( 0 );
45          }
46          catch ( IOException e ) {
47             JOptionPane.showMessageDialog( null,
48                "File does not exist",
49                "Invalid File Name",
50                JOptionPane.ERROR_MESSAGE );
51             System.exit( 1 );
52          }
53       }
54    }
55
56    public static void main( String args[] )
57    {
58       new CreateRandomFile();
59    }
60 }
```

`RandomAccessFile` constructor, file opened for reading and writing.

Use our **`write`** method to write the blank **`Record`** to the file.

Writing Data Randomly to a Random-Access File

- Writing to Random-Access File
  - Must be opened in **`"rw"`** mode

```
137                   output.seek( ( accountNumber - 1 ) *
138                                Record.size() );
```

  - Method **`seek( byteLocation )`**
    - Puts file-position pointer at byte **`byteLocation`**
      - Bytes start at 0
  - Begins account 1 at location 0
    - Records evenly spaced

```
139                   record.write( output );
```

  - Once position set, write to file


```
1  // WriteRandomFile.java
2  // This program uses TextFields to get information from the
3  // user at the keyboard and writes the information to a
4  // random-access file.
5  import bankAccount.*;
6  import javax.swing.*;
7  import java.io.*;
8  import java.awt.event.*;
9  import java.awt.*;
10
11 public class WriteRandomFile extends JFrame {
12    private RandomAccessFile output;
13    private BankUI userInterface;
14    private JButton enter, open;
15
16    // Constructor -- initialize the Frame
17    public WriteRandomFile()
18    {
19       super( "Write to random access file" );
20
21       userInterface = new BankUI();
22       enter = userInterface.getDoTask();
23       enter.setText( "Enter" );
24       enter.setEnabled( false );
25       enter.addActionListener(
26          new ActionListener() {
27             public void actionPerformed( ActionEvent e )
28             {
29                addRecord();
30             }
```

```
31          }
32       );
33
34       addWindowListener(
35          new WindowAdapter() {
36             public void windowClosing( WindowEvent e )
37             {
38                if ( output != null ) {
39                   addRecord();
40                   closeFile();
41                }
42                else
43                   System.exit( 0 );
44             }
45          }
46       );
47       open = userInterface.getDoTask2();
48
49       open.setText( "Save As" );
50       open.addActionListener(
51          new ActionListener() {
52             public void actionPerformed( ActionEvent e )
53             {
54                // Open the file
55                openFile();
56             }
57          }
58       );
59       getContentPane().add( userInterface,
60                             BorderLayout.CENTER );
61
```

- 1.1 Customize button


```
62       setSize( 300, 150 );
63       show();
64    }
65
66    private void openFile()
67    {
68       JFileChooser fileChooser = new JFileChooser();
69
70       fileChooser.setFileSelectionMode(
71          JFileChooser.FILES_ONLY );
72       int result = fileChooser.showSaveDialog( this );
73
74       // user clicked Cancel button on dialog
75       if ( result == JFileChooser.CANCEL_O
76          return;
77
78       File fileName = fileChooser.getSelec
79
80       if ( fileName == null ||
81          fileName.getName().equals( "" ) )
82          JOptionPane.showMessageDialog( th
83             "Invalid File Name",
84             "Invalid File Name",
85             JOptionPane.ERROR_MESSAGE );
86
87       else {
88          // Open the file
89          try {
90             output = new RandomAccessFile( fileName, "rw" );
91             enter.setEnabled( true );
92             open.setEnabled( false );
93          }
```

- 2. Method `openFile`
- 2.1 `showSaveDialog`
- 2.2 `RandomAccessFile`

Opened with `"rw"` mode.


```
94          catch ( IOException e ) {
95             JOptionPane.showMessageDialog( this,
96                "File does not exist",
97                "Invalid File Name",
98                JOptionPane.ERROR_MESSAGE );
99          }
100      }
101   }
102
103   private void closeFile()
104   {
105      try {
106         output.close();
107         System.exit( 0 );
108      }
109      catch( IOException ex ) {
110         JOptionPane.showMessageDialog( this,
111            "Error closing file",
112            "Error", JOptionPane.ERROR_MESSAGE );
113         System.exit( 1 );
114      }
115   }
116
117   public void addRecord()
118   {
119      int accountNumber = 0;
120      String fields[] = userInterface.getFieldValues();
121      Record record = new Record();
122
```



```
122
123      if ( !fields[ BankUI.ACCOUNT ].equals( "" ) ) {
124         // output the values to the file
125         try {
126            accountNumber =
127               Integer.parseInt( fields[ BankUI.ACCOUNT ] );
128
129            if ( accountNumber > 0 && accountNumber <= 100 ) {
130               record.setAccount( accountNumber );
131
132               record.setFirstName( fields[ BankUI.FIRST ] );
133               record.setLastName( fields[ BankUI.LAST ] );
134               record.setBalance( Double.parseDouble(
135                                  fields[ BankUI.BALANCE ] ) );
136
137               output.seek( ( accountNumber - 1 ) *
138                            Record.size() );
139               record.write( output );
140            }
141
142            userInterface.clearFields();  // clear Text
143         }
144         catch ( NumberFormatException nfe ) {
145               JOptionPane.showMessageDialog( this,
146                  "Bad account number or balance",
147                  "Invalid Number Format",
148                  JOptionPane.ERROR_MESSAGE );
149         }
150         catch ( IOException io ) {
151            closeFile();
152         }
```

Set file position pointer based on **`accountNumber`** (the key), and write to file.


```
153      }
154   }
155
156   // Create a WriteRandomFile object and start the program
157   public static void main( String args[] )
158   {
159      new WriteRandomFile();
160   }
161}
```

- 5. `main`

- Program Output

Reading Data Sequentially from a Random-Access File

- Reading random-access files
  - Open for **`"r"`**
    - Principle of least privilege
    - No need to open for **`"rw"`**
- Sorting
  - Records stored in sorted order (by account number)
  - Very fast searching
    - Space/time tradeoff: quick sort, can waste space

```
1  // ReadRandomFile.java
2  // This program reads a random-access file sequentially and
3  // displays the contents one record at a time in text fields.
4  import java.io.*;
5  import java.awt.*;
6  import java.awt.event.*;
7  import java.text.DecimalFormat;
8  import javax.swing.*;
9  import bankAccount.*;
10
11 public class ReadRandomFile extends JFrame {
12    private BankUI userInterface;
13    private RandomAccessFile input;
14    private JButton next, open;
15
16    public ReadRandomFile()
17    {
18       super( "Read Client File" );
19
20       userInterface = new BankUI();
21       next = userInterface.getDoTask();
22       next.setText( "Next" );
23       next.setEnabled( false );
24       next.addActionListener(
25          new ActionListener() {
26             public void actionPerformed( ActionEvent e )
27             {
28                readRecord();
29             }
30          }
31       );
```

- 1. Constructor
- 1.1 Customize button


```
32
33       addWindowListener(
34          new WindowAdapter() {
35             public void windowClosing( WindowEvent e )
36             {
37
38                if ( input != null ) {
39                   closeFile();
40                }
41                else
42                   System.exit( 0 );
43             }
44          }
45       );
46       open = userInterface.getDoTask2();
47
48       open.setText( "Read File" );
49       open.addActionListener(
50          new ActionListener() {
51             public void actionPerformed( ActionEvent e )
52             {
53                openFile();
54             }
55          }
56       );
57       getContentPane().add( userInterface );
58
59       setSize( 300, 150 );
60       show();
61    }
62
```

- 1.2 Customize button


```
63    private void openFile()
64    {
65       JFileChooser fileChooser = new JFileChooser();
66
67       fileChooser.setFileSelectionMode(
68          JFileChooser.FILES_ONLY );
69       int result = fileChooser.showOpenDialog( this );
70
71       // user clicked Cancel button on dialog
72       if ( result == JFileChooser.CANCEL_OPTION )
73          return;
74
75       File fileName = fileChooser.getSelectedFile();
76
77       if ( fileName == null ||
78            fileName.getName().equals( "" ) )
79          JOptionPane.showMessageDialog( this,
80             "Invalid File Name",
81             "Invalid File Name",
82             JOptionPane.ERROR_MESSAGE );
83       else {
84          // Open the file
85          try {
86             input = new RandomAccessFile( fileName, "r" );
87             next.setEnabled( true );
88             open.setEnabled( false );
89          }
90
```

- 2. Method `openFile`
- 2.1 `showOpenDialog`

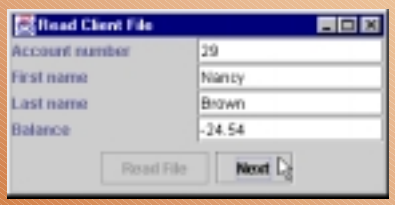

```
91          catch ( IOException e ) {
92             JOptionPane.showMessageDialog( this,
93                "File does not exist",
94                "Invalid File Name",
95                JOptionPane.ERROR_MESSAGE );
96          }
97       }
98    }
99
100   public void readRecord()
101   {
102      DecimalFormat twoDigits = new DecimalFormat( "0.00" );
103      Record record = new Record();
104
105      // read a record and display
106      try {
107         do {
108            record.read( input );
109         } while ( record.getAccount() == 0 );
110
111         String values[] = {
112            String.valueOf( record.getAccount
113            record.getFirstName(),
114            record.getLastName(),
115            String.valueOf( record.getBalance
116         userInterface.setFieldValues( values );
117      }
118      catch ( EOFException eof ) {
119         closeFile();
120      }
```

Call **`read`** (from class **`Record`**), update **`JTextField`**s. Loop skips past empty records (account **`0`**).

- 3. Method `readRecord`
- 3.1 `read`


```
121      catch ( IOException e ) {
122         JOptionPane.showMessageDialog( this,
123            "Error Reading File",
124            "Error",
125            JOptionPane.ERROR_MESSAGE );
126         System.exit( 1 );
127      }
128   }
129
130   private void closeFile()
131   {
132      try {
133         input.close();
134         System.exit( 0 );
135      }
136      catch( IOException ex ) {
137         JOptionPane.showMessageDialog( this,
138            "Error closing file",
139            "Error", JOptionPane.ERROR_MESSAGE );
140         System.exit( 1 );
141      }
142   }
143
144   public static void main( String args[] )
145   {
146      new ReadRandomFile();
147   }
148}
```

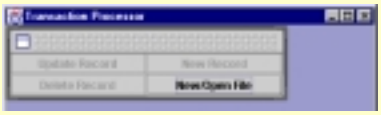
- 4. `closeFile`
- 5. `main`



- Program Output

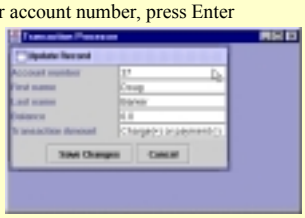
Example: A Transaction-Processing Program

- Substantial example in text
  - Uses techniques we have learned
  - Random access file
    - Update, add, delete accounts
- GUI
  - Use **`JInternalFrame`** and **`JDesktopPane`** (Chapter 13)



Example: A Transaction-Processing Program

- Update record
  - Enter account number, press Enter



Example: A Transaction-Processing Program

- Update record
  - Program validates account

```
284             if ( accountNumber < 1 || accountNumber > 100 ) {
```

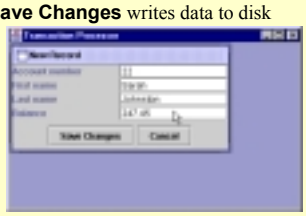
    - If valid, reads record

```
291             file.seek( ( accountNumber - 1 ) * Record.size() );
292             record.read( file );
```

  - User types transaction amount
    - Added to balance
  - Save changes
    - Calls **`addRecord`**

Example: A Transaction-Processing Program

- New record
  - Displays internal frame
  - Edit **`JTextField`**s
  - **Save Changes** writes data to disk



Example: A Transaction-Processing Program

- Delete record
  - Enter account number to delete
  - Sets account number to **`0`** (empty record)

Class `File`

- **`java.io`**
  - Many classes for I/O processing
  - Concentrated on classes for processing
    - Sequential files
      - **`FileInputStream`**, **`FileOutputStream`**
    - Object streams
      - **`ObjectInputStream`**, **`ObjectOutputStream`**
    - Random Access Files
      - **`RandomAccessFile`**
  - Discuss class **`File`**
    - Get disk information about file/directory
    - Does not process or open file

Class File

- Checking if file exists
  - Opening file for output using **FileOutputStream**
    - Discards contents without warning
  - **File** object can discover if file exists
    - Warn user, or open with **RandomAccessFile**
- Constructors
  - **File (String name)**
    - Path information, or file/directory name
    - Absolute path: starts at root directory, lists all directories leading to file
    - Relative path: subset of directories
      - Start from directory application started

Class File

- Constructors
  - **File (String pathToName, String name)**
    - **pathToName** - absolute or relative path
    - **name** - name of file/directory
  - **File (File directory, String name)**
    - **directory** - absolute or relative path
    - **name** - file or directory
- Separator character
  - Separates files and directories in path
    - **/** (Unix) or **** (Windows)
  - Java recognizes both
    - **c:\java/README** is valid
  - Use **** in string literals (remember, **** is escape character)

Class File

- Methods
  - **boolean canRead()** - readable?
  - **boolean canWrite()** - writeable?
  - **boolean exists()**
  - **boolean isFile()**
  - **boolean isDirectory()**
  - **boolean isAbsolute()** - is absolute path?
  - **String getAbsolutePath()** - returns absolute path
  - **String getName()** - name of file or directory

Class File

- Methods
  - **String getPath()**
  - **String getParent()** - returns parent directory
  - **long length()** - length of file in bytes (**0** for directory)
  - **long lastModified()** - platform dependent
  - **String[] list()** - contents of directory
  - More methods in API
- Class **RandomAccessFile**
  - Method **readLine**
    - Reads a line of text from file, returns **String**
    - Line ends when reaches **\n**, **\r** or end-of-file


```
1  // FileTest.java
2  // Demonstrating the File class.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.io.*;
6  import javax.swing.*;
7
8  public class FileTest extends JFrame
9                         implements ActionListener {
10    private JTextField enter;
11    private JTextArea output;
12
13    public FileTest()
14    {
15       super( "Testing class File" );
16
17       enter = new JTextField(
18          "Enter file or directory name here" );
19       enter.addActionListener( this );
20       output = new JTextArea();
21       Container c = getContentPane();
22       ScrollPane p = new ScrollPane();
23       p.add( output );
24       c.add( enter, BorderLayout.NORTH );
25       c.add( p, BorderLayout.CENTER );
26
27       setSize( 400, 400 );
28       show();
29    }
30
```

- 1. Constructor
- 1.1 GUI


```
31 public void actionPerformed( ActionEvent e )
32 {
33    File name = new File( e.getActionCommand() );
34
35    if ( name.exists() ) {
36       output.setText(
37          name.getName() + " exists\n" +
38          ( name.isFile() ? "is a file\n
39                          "is not a fi
40          ( name.isDirectory() ? "is a d
41                               "is not
42          ( name.isAbsolute() ? "is abso
43                              "is not
44          "Last modified: " + name.lastM
45          "\nLength: " + name.length() +
46          "\nPath: " + name.getPath() +
47          "\nAbsolute path: " + name.get
48          "\nParent: " + name.getParent(
49
50    if ( name.isFile() ) {
51       try {
52          RandomAccessFile r =
53             new RandomAccessFile( name, "r" );
54
55          StringBuffer buf = new StringBuffer();
56          String text;
57          output.append( "\n\n" );
58
59          while( ( text = r.readLine() ) != null )
60             buf.append( text + "\n" );
```

Set file/directory to what user entered in the text field.

- 2. Event handler
- 2.1 getActionCommand
- 2.2 exists
- 2.3 File methods
- 2.4 isFile
- 2.5 readLine

String as a line of text, add to buffer.

```
61
62             output.append( buf.toString() );
63          }
64          catch( IOException e2 ) {
65             JOptionPane.showMessageDialog( this,
66                "FILE ERROR",
67                "FILE ERROR", JOptionPane.ERROR_MESSA
68          }
69       }
70       else if ( name.isDirectory() ) {
71          String directory[] = name.list();
72
73          output.append( "\n\nDirectory contents:\n");
74
75          for ( int i = 0; i < directory.length; i++ )
76             output.append( directory[ i ] + "\n" );
77       }
78    }
79    else {
80       JOptionPane.showMessageDialog( this,
81          e.getActionCommand() + " Does Not Exist",
82          "FILE ERROR", JOptionPane.ERROR_MESSAGE );
83    }
84 }
85
```

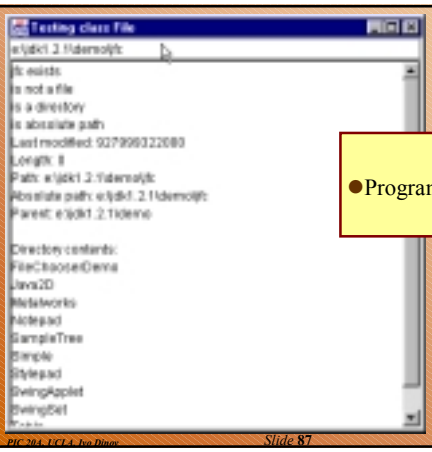
Append buffer to **output** (**JTextArea**).

Get directory contents as an array of **String**s.

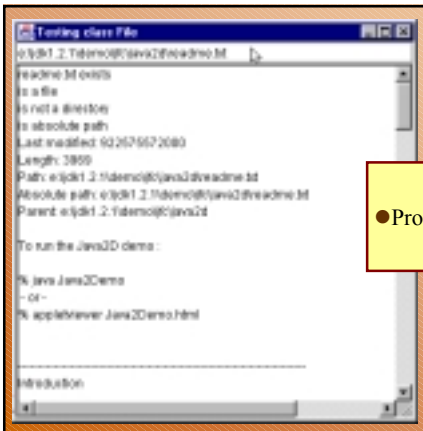
- 2.6 append
- 2.7 list


```
86 public static void main( String args[] )
87 {
88    FileTest app = new FileTest();
89
90    app.addWindowListener(
91       new WindowAdapter() {
92          public void windowClosing( WindowEvent e )
93            {
94             System.exit( 0 );
95          }
96       }
97    );
98 }
99 }
```

- 3. main



- Program Output



- Program Output