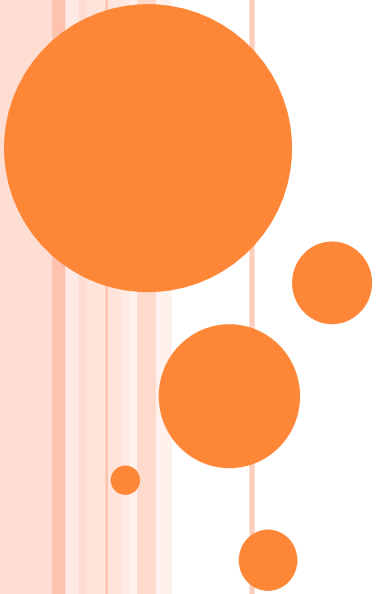
Chapter 17:
Files, Streams and Object Serialization
Part 2:4



Sequential-Access Text Files

- Sequential-access files store records in order by the record-key field.
- Text files are human-readable files.

Creating a Sequential-Access Text File

- Java imposes no structure on a file
  - Notions such as records do not exist as part of the Java language.
  - You must structure files to meet the requirements of your applications.

```
1  // Fig. 17.4: AccountRecord.java
2  // AccountRecord class maintains information for one account.
3  package com.deitel.ch17; // packaged for reuse
4
5  public class AccountRecord
6  {
7     private int account;
8     private String firstName;
9     private String lastName;
10    private double balance;
11
12    // no-argument constructor calls other constructor with default values
13    public AccountRecord()
14    {
15       this( 0, "", "", 0.0 ); // call four-argument constructor
16    } // end no-argument AccountRecord constructor
17
```

Fig. 17.4 | AccountRecord class maintains information for one account. (Part 1 of 4.)

```
18      // initialize a record
19      public AccountRecord( int acct, String first, String last, double bal )
20      {
21         setAccount( acct );
22         setFirstName( first );
23         setLastName( last );
24         setBalance( bal );
25      } // end four-argument AccountRecord constructor
26
27      // set account number
28      public void setAccount( int acct )
29      {
30         account = acct;
31      } // end method setAccount
32
33      // get account number
34      public int getAccount()
35      {
36         return account;
37      } // end method getAccount
38
```

Fig. 17.4 | `AccountRecord` class maintains information for one account. (Part 2 of 4.)

```
39      // set first name
40      public void setFirstName( String first )
41      {
42         firstName = first;
43      } // end method setFirstName
44
45      // get first name
46      public String getFirstName()
47      {
48         return firstName;
49      } // end method getFirstName
50
51      // set last name
52      public void setLastName( String last )
53      {
54         lastName = last;
55      } // end method setLastName
56
57      // get last name
58      public String getLastName()
59      {
60         return lastName;
61      } // end method getLastName
```

Fig. 17.4 | `AccountRecord` class maintains information for one account. (Part 3 of 4.)

```
62
63       // set balance
64       public void setBalance( double bal )
65       {
66          balance = bal;
67       } // end method setBalance
68
69       // get balance
70       public double getBalance()
71       {
72          return balance;
73       } // end method getBalance
74    } // end class AccountRecord
```

Fig. 17.4 | `AccountRecord` class maintains information for one account. (Part 4 of 4.)

Creating a Sequential-Access Text File (cont.)

- `Formatter` outputs formatted `String`s to the specified stream.
- The constructor with one `String` argument receives the name of the file, including its path.
  - If a path is not specified, the JVM assumes that the file is in the directory from which the program was executed.
- If the file does not exist, it will be created.
- If an existing file is opened, its contents are **truncated.**

```
1  // Fig. 17.5: CreateTextFile.java
2  // Writing data to a sequential text file with class Formatter.
3  import java.io.FileNotFoundException;
4  import java.lang.SecurityException;
5  import java.util.Formatter;
6  import java.util.FormatterClosedException;
7  import java.util.NoSuchElementException;
8  import java.util.Scanner;
9
10 import com.deitel.ch17.AccountRecord;
11
12 public class CreateTextFile
13 {
14    private Formatter output; // object used to output text to file
15
```

Fig. 17.5 | Writing data to a sequential text file with class `Formatter`. (Part 1 of 5.)

```
16      // enable user to open file
17      public void openFile()
18      {
19         try
20         {
21            output = new Formatter( "clients.txt" ); // open the file
22         } // end try
23         catch ( SecurityException securityException )
24         {
25            System.err.println(
26               "You do not have write access to this file." );
27            System.exit( 1 ); // terminate the program
28         } // end catch
29         catch ( FileNotFoundException fileNotFoundException )
30         {
31            System.err.println( "Error opening or creating file." );
32            System.exit( 1 ); // terminate the program
33         } // end catch
34      } // end method openFile
35
```

Fig. 17.5 | Writing data to a sequential text file with class `Formatter`. (Part 2 of 5.)

```
36      // add records to file
37      public void addRecords()
38      {
39         // object to be written to file
40         AccountRecord record = new AccountRecord();
41
42         Scanner input = new Scanner( System.in );
43
44         System.out.printf( "%s\n%s\n%s\n%s\n\n",
45            "To terminate input, type the end-of-file indicator ",
46            "when you are prompted to enter input.",
47            "On UNIX/Linux/Mac OS X type <ctrl> d then press Enter",
48            "On Windows type <ctrl> z then press Enter" );
49
50         System.out.printf( "%s\n%s",
51            "Enter account number (> 0), first name, last name and balance.",
52            "? " );
53
```

Fig. 17.5 | Writing data to a sequential text file with class `Formatter`. (Part 3 of 5.)

```
54          while ( input.hasNext() ) // loop until end-of-file indicator
55          {
56             try // output values to file
57             {
58                // retrieve data to be output
59                record.setAccount( input.nextInt() ); // read account number
60                record.setFirstName( input.next() ); // read first name
61                record.setLastName( input.next() ); // read last name
62                record.setBalance( input.nextDouble() ); // read balance
63
64                if ( record.getAccount() > 0 )
65                {
66                   // write new record
67                   output.format( "%d %s %s %.2f\n", record.getAccount(),
68                      record.getFirstName(), record.getLastName(),
69                      record.getBalance() );
70                } // end if
71                else
72                {
73                   System.out.println(
74                      "Account number must be greater than 0." );
75                } // end else
76             } // end try
```

Fig. 17.5 | Writing data to a sequential text file with class `Formatter`. (Part 4 of 5.)

```
77            catch ( FormatterClosedException formatterClosedException )
78            {
79               System.err.println( "Error writing to file." );
80               return;
81            } // end catch
82            catch ( NoSuchElementException elementException )
83            {
84               System.err.println( "Invalid input. Please try again." );
85               input.nextLine(); // discard input so user can try again
86            } // end catch
87
88            System.out.printf( "%s %s\n%s", "Enter account number (>0),",
89               "first name, last name and balance.", "? " );
90         } // end while
91      } // end method addRecords
92
93      // close file
94      public void closeFile()
95      {
96         if ( output != null )
97            output.close();
98      } // end method closeFile
99   } // end class CreateTextFile
```

Fig. 17.5 | Writing data to a sequential text file with class `Formatter`. (Part 5 of 5.)

Creating a Sequential-Access Text File (cont.)

- A **`SecurityException`** occurs if the user does not have permission to write data to the file.
- A **`FileNotFoundException`** occurs if the file does not exist and a new file cannot be created.
- `static` method **`System.exit`** terminates an application.
  - An argument of `0` indicates successful program termination.
  - A nonzero value, normally indicates that an error has occurred.
  - The argument is useful if the program is executed from a **batch file** on Windows or a **shell script** on UNIX/Linux/Mac OS X.

Operating system	Key combination
UNIX/Linux/Mac OS X	*<Enter> <Ctrl> d*
Windows	*<Ctrl> z*

Fig. 17.6 | End-of-file key combinations.

Creating a Sequential-Access Text File (cont.)

- `Scanner` method `hasNext` determines whether the end-of-file key combination has been entered.
- A **`NoSuchElementException`** occurs if the data being read by a `Scanner` method is in the wrong format or if there is no more data to input.
- `Formatter` method **`format`** works like `System.out.printf`
- A **`FormatterClosedException`** occurs if the `Formatter` is closed when you attempt to output.
- `Formatter` method **`close`** closes the file.
  - If method `close` is not called explicitly, the operating system normally will close the file when program execution terminates.

Creating a Sequential-Access Text File (cont.)

- Different platforms use different line-separator characters.
- On UNIX/Linux-/Mac OS X, the line separator is a newline (`\n`).
- On Windows, it is a combination of a carriage return and a line feed—represented as `\r\n`.
- You can use the **`%n`** format specifier in a format control string to output a platform-specific line separator.
- Method `System.out.println` outputs a platform-specific line separator after its argument.
- Regardless of the line separator used in a text file, a Java program can still recognize the lines of text and read them.

```
1  // Fig. 17.7: CreateTextFileTest.java
2  // Testing the CreateTextFile class.
3
4  public class CreateTextFileTest
5  {
6     public static void main( String[] args )
7     {
8        CreateTextFile application = new CreateTextFile();
9
10       application.openFile();
11       application.addRecords();
12       application.closeFile();
13    } // end main
14 } // end class CreateTextFileTest
```

Fig. 17.7 | Testing the `CreateTextFile` class. (Part 1 of 2.)

```
To terminate input, type the end-of-file indicator
when you are prompted to enter input.
On UNIX/Linux/Mac OS X type <ctrl> d then press Enter
On Windows type <ctrl> z then press Enter

Enter account number (> 0), first name, last name and balance.
? 100 Bob Jones 24.98
Enter account number (> 0), first name, last name and balance.
? 200 Steve Doe -345.67
Enter account number (> 0), first name, last name and balance.
? 300 Pam White 0.00
Enter account number (> 0), first name, last name and balance.
? 400 Sam Stone -42.16
Enter account number (> 0), first name, last name and balance.
? 500 Sue Rich 224.62
Enter account number (> 0), first name, last name and balance.
? ^Z
```

Fig. 17.7 | Testing the `CreateTextFile` class. (Part 2 of 2.)

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

Fig. 17.8 | Sample data for the program in Figs. 17.5–17.7.

Thank you

☺