17 Files, Streams and Object Serialization: Solutions

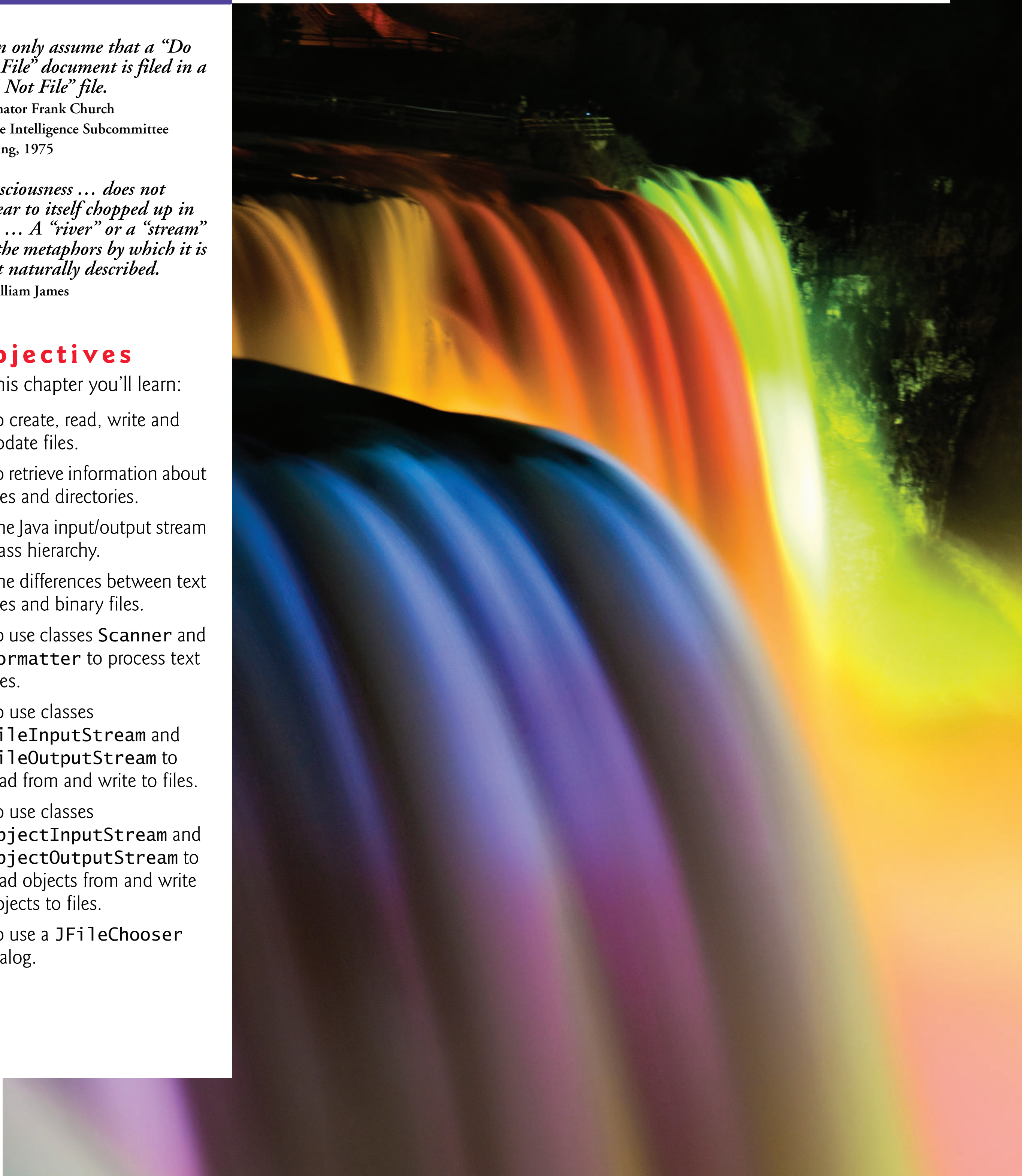
I can only assume that a "Do Not File" document is filed in a "Do Not File" file.
—Senator Frank Church
Senate Intelligence Subcommittee Hearing, 1975

Consciousness … does not appear to itself chopped up in bits. … A "river" or a "stream" are the metaphors by which it is most naturally described.
—William James

Objectives

In this chapter you'll learn:

- To create, read, write and update files.
- To retrieve information about files and directories.
- The Java input/output stream class hierarchy.
- The differences between text files and binary files.
- To use classes `Scanner` and `Formatter` to process text files.
- To use classes `FileInputStream` and `FileOutputStream` to read from and write to files.
- To use classes `ObjectInputStream` and `ObjectOutputStream` to read objects from and write objects to files.
- To use a `JFileChooser` dialog.



Self-Review Exercises

17.1 Determine whether each of the following statements is *true* or *false*. If *false*, explain why.

a) You must explicitly create the stream objects `System.in`, `System.out` and `System.err`.

ANS: False. These three streams are created for you when a Java application begins executing.

b) When reading data from a file using class `Scanner`, if you wish to read data in the file multiple times, the file must be closed and reopened to read from the beginning of the file.

ANS: True.

c) Method `exists` of class `File` returns `true` if the name specified as the argument to the `File` constructor is a file or directory in the specified path.

ANS: True.

d) Binary files are human readable in a text editor.

ANS: False. Text files are human readable in a text editor. Binary files might be human readable, but only if the bytes in the file represent ASCII characters.

e) An absolute path contains all the directories, starting with the root directory, that lead to a specific file or directory.

ANS: True.

f) Class `Formatter` contains method `printf`, which enables formatted data to be output to the screen or to a file.

ANS: False. Class `Formatter` contains method `format`, which enables formatted data to be output to the screen or to a file.

17.2 Complete the following tasks, assuming that each applies to the same program:

a) Write a statement that opens file `"oldmast.txt"` for input—use `Scanner` variable `inOldMaster`.

ANS:
```
Scanner inOldMaster = new Scanner( new File( "oldmast.txt" ) );
```

b) Write a statement that opens file `"trans.txt"` for input—use `Scanner` variable `inTransaction`.

ANS:
```
Scanner inTransaction = new Scanner( new File( "trans.txt" ) );
```

c) Write a statement that opens file `"newmast.txt"` for output (and creation)—use `formatter` variable `outNewMaster`.

ANS:
```
Formatter outNewMaster = new Formatter( "newmast.txt" );
```

d) Write the statements needed to read a record from the file `"oldmast.txt"`. Use the data to create an object of class `AccountRecord`—use `Scanner` variable `inOldMaster`. Assume that class `AccountRecord` is the same as the `AccountRecord` class in Fig. 17.5.

ANS:
```
AccountRecord account = new AccountRecord();
account.setAccount( inOldMaster.nextInt() );
account.setFirstName( inOldMaster.next() );
account.setLastName( inOldMaster.next() );
account.setBalance( inOldMaster.nextDouble() );
```

e) Write the statements needed to read a record from the file `"trans.txt"`. The record is an object of class `TransactionRecord`—use `Scanner` variable `inTransaction`. Assume that class `TransactionRecord` contains method `setAccount` (which takes an `int`) to set the account number and method `setAmount` (which takes a `double`) to set the amount of the transaction.

ANS:
```
TransactionRecord transaction = new Transaction();
transaction.setAccount( inTransaction.nextInt() );
transaction.setAmount( inTransaction.nextDouble() );
```

f) Write a statement that outputs a record to the file `"newmast.txt"`. The record is an object of type `AccountRecord`—use `Formatter` variable `outNewMaster`.

ANS:
```
outNewMaster.format( "%d %s %s %.2f\n",
account.getAccount(), account.getFirstName(),
account.getLastName(), account.getBalance() );
```

17.3 Complete the following tasks, assuming that each applies to the same program:

a) Write a statement that opens file `"oldmast.ser"` for input—use `ObjectInputStream` variable `inOldMaster` to wrap a `FileInputStream` object.

ANS:
```
ObjectInputStream inOldMaster = new ObjectInputStream(
   new FileInputStream( "oldmast.ser" ) );
```

b) Write a statement that opens file `"trans.ser"` for input—use `ObjectInputStream` variable `inTransaction` to wrap a `FileInputStream` object.

ANS:
```
ObjectInputStream inTransaction = new ObjectInputStream(
   new FileInputStream( "trans.ser" ) );
```

c) Write a statement that opens file `"newmast.ser"` for output (and creation)—use `ObjectOutputStream` variable `outNewMaster` to wrap a `FileOutputStream`.

ANS:
```
ObjectOutputStream outNewMaster = new ObjectOutputStream(
   new FileOutputStream( "newmast.ser" ) );
```

d) Write a statement that reads a record from the file `"oldmast.ser"`. The record is an object of class `AccountRecordSerializable`—use `ObjectInputStream` variable `inOldMaster`. Assume class `AccountRecordSerializable` is the same as the `AccountRecordSerializable` class in Fig. 17.16.

ANS: `accountRecord = ( AccountRecordSerializable ) inOldMaster.readObject();`

e) Write a statement that reads a record from the file `"trans.ser"`. The record is an object of class `TransactionRecord`—use `ObjectInputStream` variable `inTransaction`.

ANS: `transactionRecord = ( TransactionRecord ) inTransaction.readObject();`

f) Write a statement that outputs a record of type `AccountRecordSerializable` to the file `"newmast.ser"`—use `ObjectOutputStream` variable `outNewMaster`.

ANS: `outNewMaster.writeObject( newAccountRecord );`

17.4 Find the error in each block of code and show how to correct it.

a) Assume that `account`, `company` and `amount` are declared.

```
ObjectOutputStream outputStream;
outputStream.writeInt( account );
outputStream.writeChars( company );
outputStream.writeDouble( amount );
```

ANS: Error: The file was not opened before the attempt to output data to the stream.
Correction: Open a file for output by creating a new `ObjectOutputStream` object that wraps a `FileOutputStream` object.

b) The following statements should read a record from the file `"payables.txt"`. The `Scanner` variable `inPayable` should be used to refer to this file.

```
Scanner inPayable = new Scanner( new File( "payables.txt" ) );
PayablesRecord record = ( PayablesRecord ) inPayable.readObject();
```

ANS: Error: This example uses text files with a `Scanner`; there is no object serialization. As a result, method `readObject` cannot be used to read that data from the file. Each piece of data must be read separately, then used to create a `PayablesRecord` object.
Correction: Use `methods` of `inPayable` to read each piece of the `PayablesRecord` object.

Exercises

NOTE: Solutions to the programming exercises are located in the `ch17solutions` *folder. Each exercise has its own folder named* `ex17_##` *where* `##` *is a two-digit number representing the exercise number. For example, exercise 17.5's solution is located in the folder* `ex17_05`.