**Section 15.1 Introduction**

• Computers use files for long-term retention of large amounts of persistent data (p. [645](http://proquest.safaribooksonline.com/9780133813036/ch15_html#page_645)), even after the programs that created the data terminate.

• Computers store files on secondary storage devices (p.[645](http://proquest.safaribooksonline.com/9780133813036/ch15_html#page_645)) such as hard disks.

#### Section 15.2 Files and Streams

• Java views each file as a sequential stream of bytes (p.[645](http://proquest.safaribooksonline.com/9780133813036/ch15_html#page_645)).

• Every operating system provides a mechanism to determine the end of a file, such as an end-of-file marker (p. [645](http://proquest.safaribooksonline.com/9780133813036/ch15_html#page_645)) or a count of the total bytes in the file.

• Byte-based streams (p. [646](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec2_html#page_646)) represent data in binary format.

• Character-based streams (p. [646](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec2_html#page_646)) represent data as sequences of characters.

• Files created using byte-based streams are binary files (p. [646](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec2_html#page_646)). Files created using character-based streams are text files (p. [646](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec2_html#page_646)). Text files can be read by text editors, whereas binary files are read by a program that converts the data to a human-readable format.

• Java also can associate streams with different devices. Three stream objects are associated with devices when a Java program begins executing—System.in,System.out and System.err.

#### Section 15.3 Using NIO Classes and Interfaces to Get File and Directory Information

• A Path (p. [647](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec3_html#page_647)) represents the location of a file or directory. Path objects do not open files or provide any file-processing capabilities.

• Class Paths (p. [647](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec3_html#page_647)) is used to get a Path object representing a file or directory location.

• Class Files (p. [647](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec3_html#page_647)) provides static methods for common file and directory manipulations, including methods for copying files; creating and deleting files and directories; getting information about files and directories; reading the contents of files; getting objects that allow you to manipulate the contents of files and directories; and more.

• A DirectoryStream (p. [647](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec3_html#page_647)) enables a program to iterate through the contents of a directory.

• The static method get (p. [647](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec3_html#page_647)) of class Pathsconverts a String representing a file’s or directory’s location into a Path object.

• Character-based input and output can be performed with classes Scanner and Formatter.

• Class Formatter (p. [647](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec3_html#page_647)) enables formatted data to be output to the screen or to a file in a manner similar toSystem.out.printf.

• An absolute path (p. [647](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec3_html#page_647)) contains all the directories, starting with the root directory (p. [647](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec3_html#page_647)), that lead to a specific file or directory. Every file or directory on a disk drive has the same root directory in its path.

• A relative path (p. [647](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec3_html#page_647)) starts from the directory in which the application began executing.

• Files static method exists (p. [648](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec7_html#page_648)) receives a Pathand determines whether it exists (either as a file or as a directory) on disk.

• Path method getFileName (p. [648](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec7_html#page_648)) gets the Stringname of a file or directory without any location information.

• Files static method isDirectory (p. [648](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec7_html#page_648)) receives aPath and returns a boolean indicating whether thatPath represents a directory on disk.

• Path method isAbsolute (p. [648](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec7_html#page_648)) returns a booleanindicating whether a Path represents an absolute path to a file or directory.

• Files static method getLastModifiedTime (p. [648](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec7_html#page_648)) receives a Path and returns a FileTime (packagejava.nio.file.attribute) indicating when the file was last modified.

• Files static method size (p. [648](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec7_html#page_648)) receives a Pathand returns a long representing the number of bytes in the file or directory. For directories, the value returned is platform specific.

• Path method toString (p. [648](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec7_html#page_648)) returns a Stringrepresentation of the Path.

• Path method toAbsolutePath (p. [648](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec7_html#page_648)) converts thePath on which it’s called to an absolute path.

• Files static method newDirectoryStream (p. [648](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec7_html#page_648)) returns a DirectoryStream<Path> containing Pathobjects for a directory’s contents.

• A separator character (p. [650](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec8_html#page_650)) is used to separate directories and files in the path.

#### Section 15.4 Sequential-Access Text Files

• Java imposes no structure on a file. You must structure files to meet your application’s needs.

• To retrieve data sequentially from a file, programs normally start from the beginning of the file and read all the data consecutively until the desired information is found.

• Data in many sequential files cannot be modified without the risk of destroying other data in the file. Records in a sequential-access file are usually updated by rewriting the entire file.

#### Section 15.5 Object Serialization

• Java provides a mechanism called object serialization (p. [662](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec13_html#page_662)) that enables entire objects to be written to or read from a stream.

• A serialized object (p. [662](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec13_html#page_662)) is represented as a sequence of bytes that includes the object’s data as well as information about the object’s type and the types of data it stores.

• After a serialized object has been written into a file, it can be read from the file and deserialized (p. [662](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec13_html#page_662)) to recreate the object in memory.

• Classes ObjectInputStream (p. [662](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec13_html#page_662)) andObjectOutputStream (p. [662](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec13_html#page_662)) enable entire objects to be read from or written to a stream (possibly a file).

• Only classes that implement interface Serializable(p. [663](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec15_html#page_663)) can be serialized and deserialized.

• The ObjectOutput interface (p. [662](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec13_html#page_662)) contains methodwriteObject (p. [663](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec15_html#page_663)), which takes an Object as an argument and writes its information to anOutputStream. A class that implements this interface, such as ObjectOutputStream, would ensure that theObject is Serializable.

• The ObjectInput interface (p. [662](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec13_html#page_662)) contains methodreadObject (p. [663](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec15_html#page_663)), which reads and returns a reference to an Object from an InputStream. After an object has been read, its reference can be cast to the object’s actual type.

#### Section 15.6 Opening Files with JFileChooser

• Class JFileChooser (p. [670](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec17_html#page_670)) is used to display a dialog that enables users of a program to easily select files or directories from a GUI.

#### Section 15.7 (Optional) Additional java.ioClasses

• InputStream and OutputStream are abstract classes for performing byte-based I/O.

• Pipes (p. [673](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec7_html#page_673)) are synchronized communication channels between threads. One thread sends data via aPipedOutputStream (p. [673](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec7_html#page_673)). The target thread reads information from the pipe via a PipedInputStream (p.[673](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec7_html#page_673)).

• A filter stream (p. [674](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_674)) provides additional functionality, such as aggregating data bytes into meaningful primitive-type units. FilterInputStream(p. [674](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_674)) and FilterOutputStream are typically extended, so some of their filtering capabilities are provided by their concrete subclasses.

• A PrintStream (p. [674](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_674)) performs text output.System.out and System.err are PrintStreams.

• Interface DataInput describes methods for reading primitive types from an input stream. ClassesDataInputStream (p. [674](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_674)) and RandomAccessFile each implement this interface.

• Interface DataOutput describes methods for writing primitive types to an output stream. ClassesDataOutputStream (p. [674](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_674)) and RandomAccessFileeach implement this interface.

• Buffering is an I/O-performance-enhancement technique. Buffering reduces the number of I/O operations by combining smaller outputs together in memory. The number of physical I/O operations is much smaller than the number of I/O requests issued by the program.

• With a BufferedOutputStream (p. [674](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_674)) each output operation is directed to a buffer (p. [674](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_674)) large enough to hold the data of many output operations. Transfer to the output device is performed in one large physical output operation (p. [674](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_674)) when the buffer fills. A partially filled buffer can be forced out to the device at any time by invoking the stream object’s flush method (p. [674](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_674)).

• With a BufferedInputStream (p. [675](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_675)), many “logical” chunks of data from a file are read as one large physical input operation (p. [675](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_675)) into a memory buffer. As a program requests data, it’s taken from the buffer. When the buffer is empty, the next actual physical input operation is performed.

• A ByteArrayInputStream reads from a byte array in memory. A ByteArrayOutputStream outputs to a bytearray in memory.

• A SequenceInputStream concatenates severalInputStreams. When the program reaches the end of an input stream, that stream closes, and the next stream in the sequence opens.

• The Reader (p. [675](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_675)) and Writer (p. [675](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_675)) abstractclasses are Unicode character-based streams. Most byte-based streams have corresponding character-based concrete Reader or Writer classes.

• Classes BufferedReader (p. [675](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_675)) and BufferedWriter(p. [675](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec18_html#page_675)) buffer character-based streams.

• Classes CharArrayReader (p. [676](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec19_html#page_676)) andCharArrayWriter (p. [676](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec19_html#page_676)) manipulate char arrays.

• A LineNumberReader (p. [676](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec19_html#page_676)) is a buffered character stream that tracks the number of lines read.

• Classes FileReader (p. [676](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec19_html#page_676)) and FileWriter (p. [676](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec19_html#page_676)) perform character-based file I/O.

• Class PipedReader (p. [676](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec19_html#page_676)) and class PipedWriter (p.[676](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec19_html#page_676)) implement piped-character streams for transferring data between threads.

• Class StringReader (p. [676](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec19_html#page_676)) and StringWriter (p.[676](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec19_html#page_676)) read characters from and write characters toStrings, respectively. A PrintWriter (p. [654](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec10_html#page_654)) writes characters to a stream.

### Self-Review Exercises

[**15.1**](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec11_html#ch15ans1) Determine whether each of the following statements istrue or false. If false, explain why.

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

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.

c) Files static method exists receives a Path and determines whether it exists (either as a file or as a directory) on disk

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

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

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

[**15.2**](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec11_html#ch15ans2) 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.

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

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

d) Write the statements needed to read a record from the file "oldmast.txt". Use the data to create an object of class Account—use Scanner variableinOldMaster. Assume that class Account is the same as the Account class in [Fig. 15.9](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec16_html#ch15fig09).

e) Write the statements needed to read a record from the file "trans.txt". The record is an object of classTransactionRecord—use Scanner variableinTransaction. Assume that classTransactionRecord 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.

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

[**15.3**](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec11_html#ch15ans3) 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 inOldMasterto wrap an InputStream object.

b) Write a statement that opens file "trans.ser" for input—use ObjectInputStream variableinTransaction to wrap an InputStream object.

c) Write a statement that opens file "newmast.ser" for output (and creation)—use ObjectOutputStreamvariable outNewMaster to wrap an OutputStream.

d) Write a statement that reads a record from the file"oldmast.ser". The record is an object of classAccount—use ObjectInputStream variableinOldMaster. Assume class Account is the same as the Account class in [Fig. 15.9](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec16_html#ch15fig09)

e) Write a statement that reads a record from the file"trans.ser". The record is an object of classTransactionRecord—use ObjectInputStreamvariable inTransaction.

f) Write a statement that outputs a record of typeAccount to the file "newmast.ser"—useObjectOutputStream variable outNewMaster.

### Answers to Self-Review Exercises

[**15.1**](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec10_html#ch15que1)

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

b) True.

c) True.

d) 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) True.

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

[**15.2**](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec10_html#ch15que2)

a) Scanner inOldMaster = newScanner(Paths.get("oldmast.txt"));

b) Scanner inTransaction = newScanner(Paths.get("trans.txt"));

c) Formatter outNewMaster = newFormatter("newmast.txt");

d)

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0680pro01a)

Account account = new Account();  
account.setAccount(inOldMaster.nextInt());  
account.setFirstName(inOldMaster.next());  
account.setLastName(inOldMaster.next());  
account.setBalance(inOldMaster.nextDouble());

e)

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0680pro02a)

TransactionRecord transaction = newTransaction();  
transaction.setAccount(inTransaction.nextInt());  
transaction.setAmount(inTransaction.nextDouble());

f)

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0680pro03a)

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

[**15.3**](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec10_html#ch15que3)

a)

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0680pro04a)

ObjectInputStream inOldMaster = newObjectInputStream(  
   Files.newInputStream(Paths.get("oldmast.ser")));

b)

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0680pro05a)

ObjectInputStream inTransaction = newObjectInputStream(  
   Files.newOutputStream(Paths.get("trans.ser")));

c)

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0680pro06a)

ObjectOutputStream outNewMaster = newObjectOutputStream(  
   Files.newOutputStream(Paths.get("newmast.ser")));

d) Account = (Account) inOldMaster.readObject();

e) transactionRecord = (TransactionRecord) inTransaction.readObject();

f) outNewMaster.writeObject(newAccount);

### Exercises

**15.4 (File Matching)** Self-Review [Exercise 15.2](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec10_html#ch15que2) asked you to write a series of single statements. Actually, these statements form the core of an important type of file-processing program—namely, a file-matching program. In commercial data processing, it’s common to have several files in each application system. In an accounts receivable system, for example, there’s generally a master file containing detailed information about each customer, such as the customer’s name, address, telephone number, outstanding balance, credit limit, discount terms, contract arrangements and possibly a condensed history of recent purchases and cash payments.

As transactions occur (i.e., sales are made and payments arrive in the mail), information about them is entered into a file. At the end of each business period (a month for some companies, a week for others, and a day in some cases), the file of transactions (called "trans.txt") is applied to the master file (called "oldmast.txt") to update each account’s purchase and payment record. During an update, the master file is rewritten as the file "newmast.txt", which is then used at the end of the next business period to begin the updating process again.

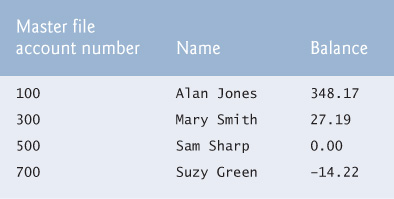
File-matching programs must deal with certain problems that do not arise in single-file programs. For example, a match does not always occur. If a customer on the master file has not made any purchases or cash payments in the current business period, no record for this customer will appear on the transaction file. Similarly, a customer who did make some purchases or cash payments could have just moved to this community, and if so, the company may not have had a chance to create a master record for this customer.

Write a complete file-matching accounts receivable program. Use the account number on each file as the record key for matching purposes. Assume that each file is a sequential text file with records stored in increasing account-number order.

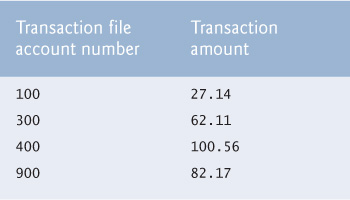
a) Define class TransactionRecord. Objects of this class contain an account number and amount for the transaction. Provide methods to modify and retrieve these values.

b) Modify class Account in [Fig. 15.9](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec16_html#ch15fig09) to include methodcombine, which takes a TransactionRecord object and combines the balance of the Account object and the amount value of the TransactionRecord object.

c) Write a program to create data for testing the program. Use the sample account data in [Figs. 15.14](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec12_html#ch15fig14)and [15.15](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec12_html#ch15fig15). Run the program to create the filestrans.txt and oldmast.txt to be used by your file-matching program.



**Fig. 15.14** | Sample data for master file.



**Fig. 15.15** | Sample data for transaction file.

d) Create class FileMatch to perform the file-matching functionality. The class should contain methods that read oldmast.txt and trans.txt. When a match occurs (i.e., records with the same account number appear in both the master file and the transactionfile), add the dollar amount in the transaction record to the current balance in the master record, and write the "newmast.txt" record. (Assume that purchases are indicated by positive amounts in the transaction file and payments by negative amounts.) When there’s a master record for a particular account, but no corresponding transaction record, merely write the master record to "newmast.txt". When there’s a transaction record, but no corresponding master record, print to a log file the message "Unmatched transaction record for account number..." (fill in the account number from the transaction record). The log file should be a text file named "log.txt".

**15.5 (File Matching with Multiple Transactions)** It’s possible (and actually common) to have several transaction records with the same record key. This situation occurs, for example, when a customer makes several purchases and cash payments during a business period. Rewrite your accounts receivable file-matching program from [Exercise 15.4](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec12_html#ch15que4) to provide for the possibility of handling several transaction records with the same record key. Modify the test data of CreateData.java to include the additional transaction records in [Fig. 15.16](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec12_html#ch15fig16).

**Fig. 15.16** | Additional transaction records.

**15.6 (File Matching with Object Serialization)** Recreate your solution for [Exercise 15.5](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec12_html#ch15que5) using object serialization. Use the statements from [Exercise 15.3](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec10_html#ch15que3) as your basis for this program. You may want to create applications to read the data stored in the .ser files—the code in [Section 15.5.2](http://proquest.safaribooksonline.com/9780133813036/ch15lev2sec17_html#ch15lev2sec17) can be modified for this purpose.

**15.7 (Telephone-Number Word Generator)** Standard telephone keypads contain the digits zero through nine. The numbers two through nine each have three letters associated with them ([Fig. 15.17](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec12_html#ch15fig17)). Many people find it difficult to memorize phone numbers, so they use the correspondence between digits and letters to develop seven-letter words that correspond to their phone numbers. For example, a person whose telephone number is 686-2377 might use the correspondence indicated in [Fig. 15.17](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec12_html#ch15fig17) to develop the seven-letter word “NUMBERS.” Every seven-letter word corresponds to exactly one seven-digit telephone number. A restaurant wishing to increase its takeout business could surely do so with the number 825-3688 (i.e., “TAKEOUT”).

**Fig. 15.17** | Telephone keypad digits and letters.

Every seven-letter phone number corresponds to many different seven-letter words, but most of these words represent unrecognizable juxtapositions of letters. It’s possible, however, that the owner of a barbershop would be pleased to know that the shop’s telephone number, 424-7288, corresponds to “HAIRCUT.” A veterinarian with the phone number 738-2273 would be pleased to know that the number corresponds to the letters “PETCARE.” An automotive dealership would be pleased to know that the dealership number, 639-2277, corresponds to “NEWCARS.”

Write a program that, given a seven-digit number, uses aPrintStream object to write to a file every possible seven-letter word combination corresponding to that number. There are 2,187 (37) such combinations. Avoid phone numbers with the digits 0 and 1.

**15.8 (Student Poll)** [Figure 7.8](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec9_html#ch07fig08) contains an array of survey responses that’s hard coded into the program. Suppose we wish to process survey results that are stored in a file. This exercise requires two separate programs. First, create an application that prompts the user for survey responses and outputs each response to a file. Use a Formatter to create a file called numbers.txt. Each integer should be written using method format. Then modify the program in [Fig. 7.8](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec9_html#ch07fig08) to read the survey responses from numbers.txt. The responses should be read from the file by using a Scanner. Use methodnextInt to input one integer at a time from the file. The program should continue to read responses until it reaches the end of the file. The results should be output to the text file "output.txt".

**15.9 (Adding Object Serialization to the** ***MyShape*Drawing Application)** Modify [Exercise 12.17](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec62_html#ch12que17) to allow the user to save a drawing into a file or load a prior drawing from a file using object serialization. Add buttons **Load** (to read objects from a file) and **Save** (to write objects to a file). Use an ObjectOutputStream to write to the file and anObjectInputStream to read from the file. Write the array ofMyShape objects using method writeObject (classObjectOutputStream), and read the array using methodreadObject (ObjectInputStream). The object-serialization mechanism can read or write entire arrays—it’s not necessary to manipulate each element of the array ofMyShape objects individually. It’s simply required that all the shapes be Serializable. For both the **Load** and **Save**buttons, use a JFileChooser to allow the user to select the file in which the shapes will be stored or from which they’ll be read. When the user first runs the program, no shapes should be displayed on the screen. The user can display shapes by opening a previously saved file or by drawing new shapes. Once there are shapes on the screen, users can save them to a file using the **Save** button.

### Making a Difference

**15.10 (Phishing Scanner)** Phishing is a form of identity theft in which, in an e-mail, a sender posing as a trustworthy source attempts to acquire private information, such as your user names, passwords, credit-card numbers and social security number. Phishing e-mails claiming to be from popular banks, credit-card companies, auction sites, social networks and online payment services may look quite legitimate. These fraudulent messages often provide links to spoofed (fake) websites where you’re asked to enter sensitive information.

Search online for phishing scams. Also check out the Anti-Phishing Working Group ([www.antiphishing.org](http://www.antiphishing.org/)), and the FBI’s Cyber Investigations website ([www.fbi.gov/about-us/investigate/cyber/cyber](http://www.fbi.gov/about-us/investigate/cyber/cyber)), where you’ll find information about the latest scams and how to protect yourself.

Create a list of 30 words, phrases and company names commonly found in phishing messages. Assign a point value to each based on your estimate of its likeliness to be in a phishing message (e.g., one point if it’s somewhat likely, two points if moderately likely, or three points if highly likely). Write an application that scans a file of text for these terms and phrases. For each occurrence of a keyword or phrase within the text file, add the assigned point value to the total points for that word or phrase. For each keyword or phrase found, output one line with the word or phrase, the number of occurrences and the point total. Then show the point total for the entire message. Does your program assign a high point total to some actual phishing e-mails you’ve received? Does it assign a high point total to some legitimate e-mails you’ve received?