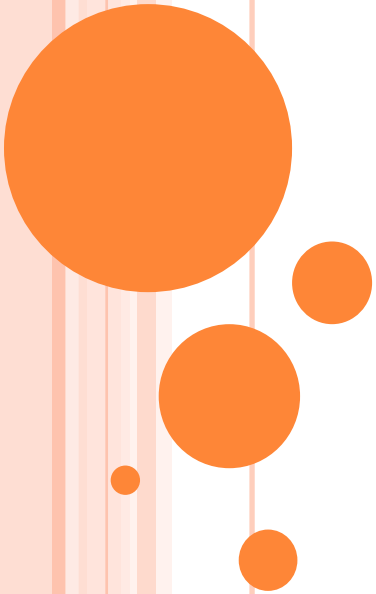
Chapter 17:
Files, Streams and Object Serialization
part 4:4



Updating Sequential-Access Files

- The data in many sequential files cannot be modified without the risk of destroying other data in the file.
- If the name "`White`" needed to be changed to "`Worthington`," the old name cannot simply be overwritten, because the new name requires more space.
- Fields in a text file—and hence records—can vary in size.
- Records in a sequential-access file are not usually updated in place. Instead, the entire file is usually rewritten.
- Rewriting the entire file is uneconomical to update just one record, but reasonable if a substantial number of records need to be updated.

OBJECT SERIALIZATION

- To read an entire object from or write an entire object to a file, Java provides **object serialization.**
- A **serialized object** is represented as a sequence of bytes that includes the object's data and its type information.
- After a serialized object has been written into a file, it can be read from the file and **deserialized** to recreate the object in memory.

Software Engineering Observation 17.1

The serialization mechanism makes exact copies of objects. This makes it a simple way to clone objects without having to override `Object` *method* `clone`.

Object Serialization (cont.)

- Classes `ObjectInputStream` and `ObjectOutputStream`, which respectively implement the **`ObjectInput`** and **`ObjectOutput`** interfaces, enable entire objects to be read from or written to a stream.
- To use serialization with files, initialize `ObjectInputStream` and `ObjectOutputStream` objects with `FileInputStream` and `FileOutputStream` objects.

Object Serialization (cont.)

- `ObjectOutput` interface method **`writeObject`** takes an `Object` as an argument and writes its information to an `OutputStream`.
- A class that implements `ObjectOuput` (such as `ObjectOutputStream`) declares this method and ensures that the object being output implements `Serializable`.
- `ObjectInput` interface method **`readObject`** 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.

Creating a Sequential-Access File Using Object Serialization

- Objects of classes that implement interface **`Serializable`** can be serialized and deserialized with `ObjectOutputStream`s and `ObjectInputStream`s.
- Interface `Serializable` is a **tagging interface**.
  - It does not contain methods.
- A class that implements `Serializable` is tagged as being a `Serializable` object.
- An `ObjectOutputStream` will not output an object unless it *is a* `Serializable` object.

```
1  // Fig. 17.15: AccountRecordSerializable.java
2  // AccountRecordSerializable class for serializable objects.
3  package com.deitel.ch17; // packaged for reuse
4
5  import java.io.Serializable;
6
7  public class AccountRecordSerializable implements Serializable
8  {
9     private int account;
10    private String firstName;
11    private String lastName;
12    private double balance;
13
14    // no-argument constructor calls other constructor with default values
15    public AccountRecordSerializable()
16    {
17       this( 0, "", "", 0.0 );
18    } // end no-argument AccountRecordSerializable constructor
19
```

Fig. 17.15 | `AccountRecordSerializable` class for serializable objects. (Part 1 of 4.)

```
20      // four-argument constructor initializes a record
21      public AccountRecordSerializable(
22         int acct, String first, String last, double bal )
23      {
24         setAccount( acct );
25         setFirstName( first );
26         setLastName( last );
27         setBalance( bal );
28      } // end four-argument AccountRecordSerializable constructor
29
30      // set account number
31      public void setAccount( int acct )
32      {
33         account = acct;
34      } // end method setAccount
35
36      // get account number
37      public int getAccount()
38      {
39         return account;
40      } // end method getAccount
41
```

Fig. 17.15 | `AccountRecordSerializable` class for serializable objects. (Part 2 of 4.)

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

Fig. 17.15 | `AccountRecordSerializable` class for serializable objects. (Part 3 of 4.)

```
60      // get last name
61      public String getLastName()
62      {
63         return lastName;
64      } // end method getLastName
65
66      // set balance
67      public void setBalance( double bal )
68      {
69         balance = bal;
70      } // end method setBalance
71
72      // get balance
73      public double getBalance()
74      {
75         return balance;
76      } // end method getBalance
77   } // end class AccountRecordSerializable
```

Fig. 17.15 | `AccountRecordSerializable` class for serializable objects. (Part 4 of 4.)

Creating a Sequential-Access File Using Object Serialization (cont.)

- In a class that implements `Serializable`, every variable must be `Serializable`.
- Any one that is not must be declared `transient` so it will be ignored during the serialization process.
- All primitive-type variables are serializable.
- For reference-type variables, check the class's documentation (and possibly its superclasses) to ensure that the type is `Serializable`.

Common Programming Error 17.2

It's a logic error to open an existing file for output when, in fact, you wish to preserve the file. Class `FileOutputStream` *provides an overloaded constructor that enables you to open a file and append data to the end of the file. This will preserve the file's contents.*

```
1  // Fig. 17.16: CreateSequentialFile.java
2  // Writing objects sequentially to a file with class ObjectOutputStream.
3  import java.io.FileOutputStream;
4  import java.io.IOException;
5  import java.io.ObjectOutputStream;
6  import java.util.NoSuchElementException;
7  import java.util.Scanner;
8
9  import com.deitel.ch17.AccountRecordSerializable;
10
11 public class CreateSequentialFile
12 {
13    private ObjectOutputStream output; // outputs data to file
14
```

Fig. 17.16 | Sequential file created using `ObjectOutputStream`. (Part 1 of 6.)

```
15      // allow user to specify file name
16      public void openFile()
17      {
18         try // open file
19         {
20            output = new ObjectOutputStream(
21               new FileOutputStream( "clients.ser" ) );
22         } // end try
23         catch ( IOException ioException )
24         {
25            System.err.println( "Error opening file." );
26         } // end catch
27      } // end method openFile
28
```

Fig. 17.16 | Sequential file created using `ObjectOutputStream`. (Part 2 of 6.)

```
29      // add records to file
30      public void addRecords()
31      {
32         AccountRecordSerializable record; // object to be written to file
33         int accountNumber = 0; // account number for record object
34         String firstName; // first name for record object
35         String lastName; // last name for record object
36         double balance; // balance for record object
37
38         Scanner input = new Scanner( System.in );
39
40         System.out.printf( "%s\n%s\n%s\n%s\n\n",
41            "To terminate input, type the end-of-file indicator ",
42            "when you are prompted to enter input.",
43            "On UNIX/Linux/Mac OS X type <ctrl> d then press Enter",
44            "On Windows type <ctrl> z then press Enter" );
45
46         System.out.printf( "%s\n%s",
47            "Enter account number (> 0), first name, last name and balance.",
48            "? " );
49
```

Fig. 17.16 | Sequential file created using `ObjectOutputStream`. (Part 3 of 6.)

```
50          while ( input.hasNext() ) // loop until end-of-file indicator
51          {
52             try // output values to file
53             {
54                accountNumber = input.nextInt(); // read account number
55                firstName = input.next(); // read first name
56                lastName = input.next(); // read last name
57                balance = input.nextDouble(); // read balance
58
59                if ( accountNumber > 0 )
60                {
61                   // create new record
62                   record = new AccountRecordSerializable( accountNumber,
63                      firstName, lastName, balance );
64                   output.writeObject( record ); // output record
65                } // end if
```

Fig. 17.16 | Sequential file created using `ObjectOutputStream`. (Part 4 of 6.)

```
66                else
67                {
68                   System.out.println(
69                      "Account number must be greater than 0." );
70                } // end else
71             } // end try
72             catch ( IOException ioException )
73             {
74                System.err.println( "Error writing to file." );
75                return;
76             } // end catch
77             catch ( NoSuchElementException elementException )
78             {
79                System.err.println( "Invalid input. Please try again." );
80                input.nextLine(); // discard input so user can try again
81             } // end catch
82
83             System.out.printf( "%s %s\n%s", "Enter account number (>0),",
84                "first name, last name and balance.", "? " );
85          } // end while
86       } // end method addRecords
87
```

Fig. 17.16 | Sequential file created using `ObjectOutputStream`. (Part 5 of 6.)

```
88       // close file and terminate application
89       public void closeFile()
90       {
91          try // close file
92          {
93             if ( output != null )
94                output.close();
95          } // end try
96          catch ( IOException ioException )
97          {
98             System.err.println( "Error closing file." );
99             System.exit( 1 );
100         } // end catch
101      } // end method closeFile
102   } // end class CreateSequentialFile
```

Fig. 17.16 | Sequential file created using `ObjectOutputStream`. (Part 6 of 6.)

```
1  // Fig. 17.17: CreateSequentialFileTest.java
2  // Testing class CreateSequentialFile.
3
4  public class CreateSequentialFileTest
5  {
6     public static void main( String[] args )
7     {
8        CreateSequentialFile application = new CreateSequentialFile();
9
10       application.openFile();
11       application.addRecords();
12       application.closeFile();
13    } // end main
14 } // end class CreateSequentialFileTest
```

Fig. 17.17 | Testing class `CreateSequentialFile`. (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. 17.17 | Testing class `CreateSequentialFile`. (Part 2 of 2.)

Reading and Deserializing Data from a Sequential-Access File

- The program in Figs. 17.18–17.19 reads records from a file created by the program in Section 17.5.1 and displays the contents.

```
1  // Fig. 17.18: ReadSequentialFile.java
2  // Reading a file of objects sequentially with ObjectInputStream
3  // and displaying each record.
4  import java.io.EOFException;
5  import java.io.FileInputStream;
6  import java.io.IOException;
7  import java.io.ObjectInputStream;
8
9  import com.deitel.ch17.AccountRecordSerializable;
10
11 public class ReadSequentialFile
12 {
13    private ObjectInputStream input;
14
```

Fig. 17.18 | Reading a file of objects sequentially with `ObjectInputStream` and displaying each record. (Part 1 of 4.)

```
15      // enable user to select file to open
16      public void openFile()
17      {
18         try // open file
19         {
20            input = new ObjectInputStream(
21               new FileInputStream( "clients.ser" ) );
22         } // end try
23         catch ( IOException ioException )
24         {
25            System.err.println( "Error opening file." );
26         } // end catch
27      } // end method openFile
28
29      // read record from file
30      public void readRecords()
31      {
32         AccountRecordSerializable record;
33         System.out.printf( "%-10s%-12s%-12s%10s\n", "Account",
34            "First Name", "Last Name", "Balance" );
35
```

Fig. 17.18 | Reading a file of objects sequentially with `ObjectInputStream` and displaying each record. (Part 2 of 4.)

```
36          try // input the values from the file
37          {
38             while ( true )
39             {
40                record = ( AccountRecordSerializable ) input.readObject();
41
42                // display record contents
43                System.out.printf( "%-10d%-12s%-12s%10.2f\n",
44                   record.getAccount(), record.getFirstName(),
45                   record.getLastName(), record.getBalance() );
46             } // end while
47          } // end try
48          catch ( EOFException endOfFileException )
49          {
50             return; // end of file was reached
51          } // end catch
52          catch ( ClassNotFoundException classNotFoundException )
53          {
54             System.err.println( "Unable to create object." );
55          } // end catch
```

Fig. 17.18 | Reading a file of objects sequentially with `ObjectInputStream` and displaying each record. (Part 3 of 4.)

```
56          catch ( IOException ioException )
57          {
58             System.err.println( "Error during read from file." );
59          } // end catch
60       } // end method readRecords
61
62       // close file and terminate application
63       public void closeFile()
64       {
65          try // close file and exit
66          {
67             if ( input != null )
68                input.close();
69          } // end try
70          catch ( IOException ioException )
71          {
72             System.err.println( "Error closing file." );
73             System.exit( 1 );
74          } // end catch
75       } // end method closeFile
76    } // end class ReadSequentialFile
```

Fig. 17.18 | Reading a file of objects sequentially with `ObjectInputStream` and displaying each record. (Part 4 of 4.)

Reading and Deserializing Data from a Sequential-Access File (cont.)

- `ObjectInputStream` method `readObject` reads an `Object` from a file.
- Method `readObject` throws an **`EOFException`** if an attempt is made to read beyond the end of the file.
- Method `readObject` throws a `ClassNotFoundException` if the class for the object being read cannot be located.

```
1  // Fig. 17.19: ReadSequentialFileTest.java
2  // Testing class ReadSequentialFile.
3
4  public class ReadSequentialFileTest
5  {
6     public static void main( String[] args )
7     {
8        ReadSequentialFile application = new ReadSequentialFile();
9
10       application.openFile();
11       application.readRecords();
12       application.closeFile();
13    } // end main
14 } // end class ReadSequentialFileTest
```

```
Account   First Name  Last Name      Balance
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. 17.19 | Testing class `ReadSequentialFile`.

Thank you