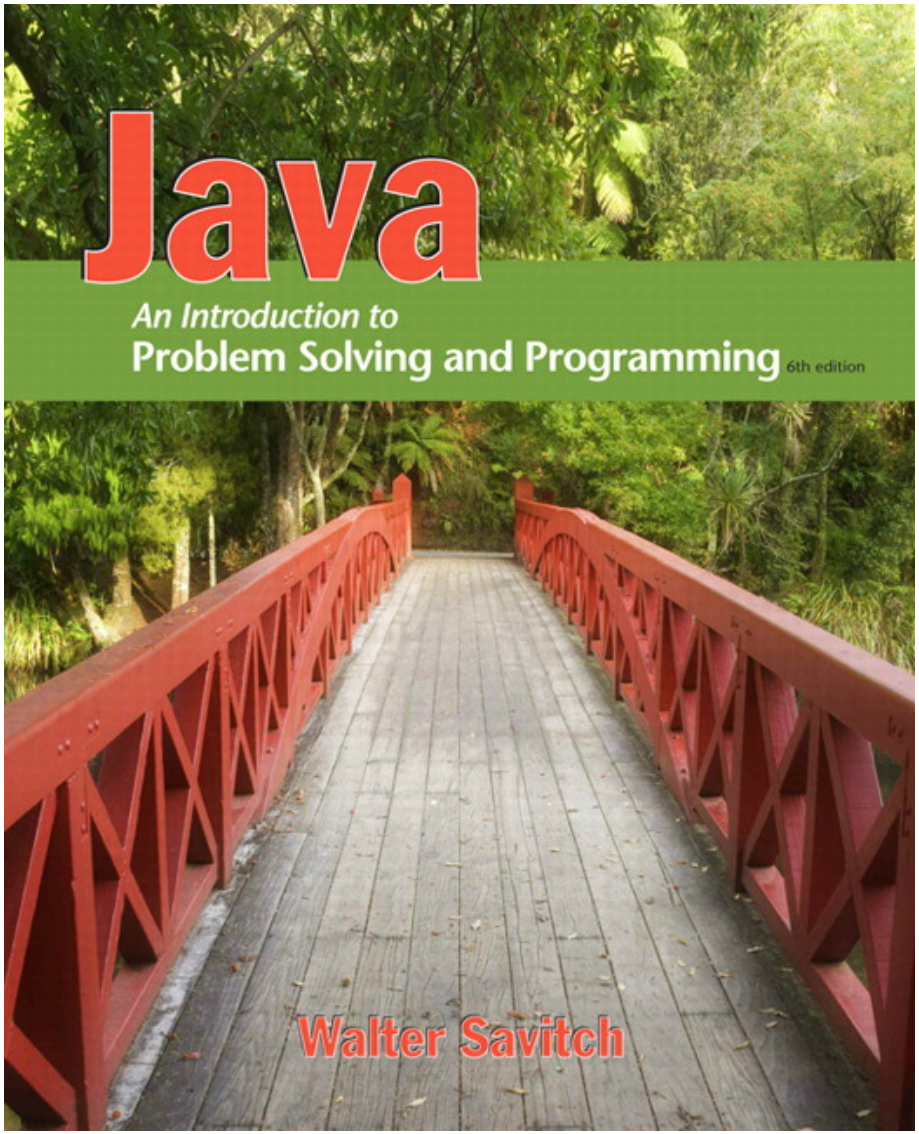



# Streams and File I/O 10

*JAVA: An Introduction to Problem Solving & Programming, 6th* Ed. By Walter Savitch
ISBN 0132162709 

**FIGURE 10.1 Input and Output Streams**

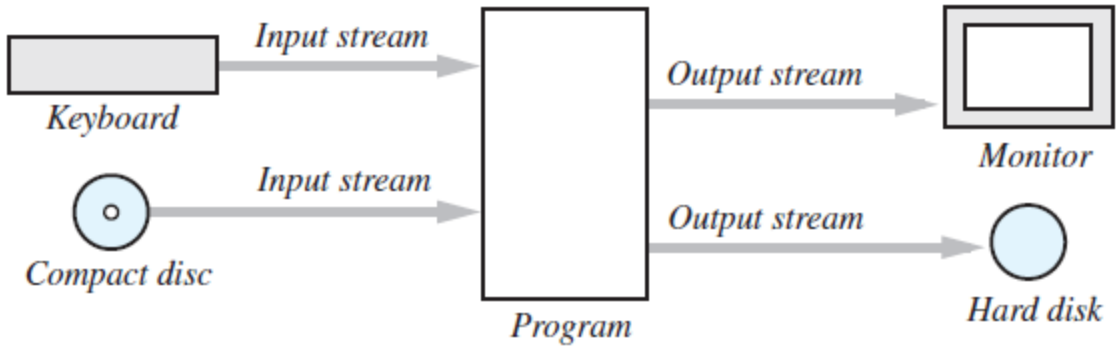


**FIGURE 10.2A Text File and a Binary File Containing the Same Values**

*A text file*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | | - | 4 | 0 | 2 | 7 | | 8 | |

...

*A binary file*

| | | |
|---|---|---|
| 12345 | -4072 | 8 |

...

## LISTING 10.1 Writing Output to a Text File *(part 1 of 2)*

```
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class TextFileOutputDemo
{
    public static void main(String[] args)
    {
        String fileName = "out.txt"; //The name could be read from
                                     //the keyboard.
        PrintWriter outputStream = null;
        try
        {
            outputStream = new PrintWriter(fileName};
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Error opening the file" +
                               fileName);
            System.exit(0);
        }
```

```
        System.out.println("Enter three lines of text:");
        Scanner keyboard = new Scanner(System.in);
        for (int count = 1; count <= 3; count++)
        {
            String line = keyboard.nextLine();
            outputStream.println(count + " " + line);
        }
        outputStream.close();
        System.out.println("Those lines were written to " +
                           fileName);
    }
}
```

### Sample Screen Output

```
Enter three lines of text:
A tall tree
in a short forest is like
a big fish in a small pond.
Those lines were written to out.txt
```

### Resulting file

```
1 A tall tree
2 in a short forest is like
3 a big fish in a small pond.
```

*You can use a text editor to read this file.*

## LISTING 10.2 Reading Data from a Text File

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;

public class TextFileInputDemo
{
    public static void main(String[] args)
    {
        String fileName = "out.txt";
        Scanner inputStream = null;
        System.out.println("The file " + fileName +
                           "\ncontains the following lines:\n");
        try
        {
            inputStream = new Scanner(new File(fileName));
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Error opening the file " +
                               fileName);
            System.exit(0);
        }
        while (inputStream.hasNextLine())
        {
            String line = inputStream.nextLine();
            System.out.println(line);
        }
        inputStream.close();
    }
}
```

### *Screen Output*

```
The file out.txt
contains the following lines;
1 A tall tree
2 in a short forest is like
3 a big fish in a small pond.
```

**FIGURE 10.3 Additional Methods in the Class `Scanner` (See also Figure 2.7)**

| |
|---|
| *Scanner_Object_Name*`.hasNext()`<br>Returns true if more input data is available to be read by the method `next`. |
| *Scanner_Object_Name*`.hasNextDouble()`<br>Returns true if more input data is available to be read by the method `nextDouble`. |
| *Scanner_Object_Name*`.hasNextInt()`<br>Returns true if more input data is available to be read by the method `nextInt`. |
| *Scanner_Object_Name*`.hasNextLine()`<br>Returns true if more input data is available to be read by the method `nextLine`. |

## LISTING 10.3 Reading a File Name and Then the File

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;

public class TextFileInputDemo2
{
    public static void main(String[] args)
    {
        System.out.print("Enter file name: ");
        Scanner keyboard = new Scanner(System.in);
        String fileName = keyboard.next();
        Scanner inputStream = null;
        System.out.println("The file " + fileName + "\n" +
                           "contains the following lines:\n");
        try
        {
            inputStream = new Scanner(new File(fileName));
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Error opening the file " +
                               fileName");
            System.exit(0).
        }
        while (inputStream.hasNextLine())
        {
            String line = inputStream.nextLine();
            System.out.println(line);
        }
        inputStream.close();
    }
}
```

### *Sample Screen Output*

```
Enter file name: out.txt
The file out.txt
contains the following lines:
1 A tall tree
2 in a short forest is like
3 a big fish in a small pond.
```

## FIGURE 10.4 Some Methods in the Class `File`

| |
|---|
| `public boolean canRead()`<br>Tests whether the program can read from the file. |
| `public boolean canWrite()`<br>Tests whether the program can write to the file. |
| `public boolean delete()`<br>Tries to delete the file. Returns true if it was able to delete the file. |
| `public boolean exists()`<br>Tests whether an existing file has the name used as an argument to the constructor when the `File` object was created. |
| `public String getName()`<br>Returns the name of the file. (Note that this name is not a path name, just a simple file name.) |
| `public String getPath()`<br>Returns the path name of the file. |
| `public long length()`<br>Returns the length of the file, in bytes. |

**LISTING 10.4 Processing a Comma-Separated Values File Containing Sales Transactions** *(part 1 of 2)*

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.File;
import java.util.Scanner;
public class TransactionReader
{
public static void main(String[] args)
    {
        String fileName = "Transactions.txt";
        try
        {
        Scanner inputStream = new Scanner(new File(fileName));
        // Skip the header line by reading and ignoring it
        String line = inputStream.nextLine();
        // Total sales
        double total = 0;
        // Read the rest of the file line by line
        while (inputStream.hasNextLine())
        {
            // Contains SKU,Quantity,Price,Description
            line = inputStream.nextLine();
```

```
                // Turn the string into an array of strings
                String[] ary = line.split(“,”);
                // Extract each item into an appropriate
                // variable
                String SKU = ary[0];
                int quantity = Integer.parseInt(ary[1]);
                double price = Double.parseDouble(ary[2]);
                String description = ary[3];
                // Output item
                System.out.printf("Sold %d of %s (SKU: %s) at "+
                    "$%1.2f each.\n",
                    quantity, description, SKU, price);
                // Compute total
                total += quantity * price;
            }
            System.out.printf("Total sales: $%1.2f\n",total);
            inputStream.close( );
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Cannot find file " + fileName);
        }
        catch(IOException e)
        {
            System.out.println("Problem with input from file " +
            fileName);
        }
    }
}
```

### *Sample Screen Output*

```
Sold 50 of SODA (SKU: 4039) at $0.99 each.
Sold 5 of T-SHIRT (SKU: 9100) at $9.50 each.
Sold 30 of JAVA PROGRAMMING TEXTBOOK (SKU: 1949) at
$110.00 each.
Sold 25 of COOKIE (SKU: 5199) at $1.50 each.
Total sales: $3434.50
```

**LISTING 10.5 Using `ObjectOutputStream` to Write to a File** *(part 1 of 2)*

```
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;

public class BinaryOutputDemo
{
    public static void main(String[] args)
    {
        String fileName = "numbers.dat";
        try
        {
            ObjectOutputStream outputStream =
            new ObjectOutputStream(new
                        FileOutputStream(fileName));
            Scanner keyboard = new Scanner(System.in);

            System.out.println("Enter nonnegative integers.");
            System.out.println("Place a negative number at the "+
                               "end.");
```

```
            int anInteger;
            do
            {
                anInteger = keyboard.nextInt();
                outputStream.writeInt(anInteger);
            } while (anInteger >= 0);

            System.out.println("Numbers and sentinel value");
            System.out.println("written to the file " + fileName);
            outputStream.close();
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Problem opening the file " +
            fileName);
        }
        catch(IOException e)
        {
            System.out.println("Problem with output to file " +
            fileName);
        }
    }
}
```

*A binary file is closed in the same way as a text file.*

### *Sample Screen Output*

```
Enter nonnegative integers.
Place a negative number at the end.
1 2 3 -1
Number and sentinel value
written to the file numbers.dat
```

The binary file after the program is run:

*This file is a binary file. You cannot read this file using a text editor.*

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | -1 |

*The **-1** in this file is a sentinel value. Ending a file with a sentinel value is not essential, as you will see later.*

**FIGURE 10.5 Some Methods in the Class `ObjectOutputStream` *(part 1 of 2)***

| |
|---|
| `public ObjectOutputStream(OutputStream streamObject)`<br>Creates an output stream that is connected to the specified binary file. There is no constructor that takes a file name as an argument. If you want to create a stream by using a file name, you write either<br>`new ObjectOutputStream(new FileOutputStream(`*File_Name*`))`<br>or, using an object of the class `File`,<br>`new ObjectOutputStream(new FileOutputStream(`<br>`new File(`*File_Name*`)))`<br>Either statement creates a blank file. If there already is a file named *File_Name*, the old contents of the file are lost.<br>The constructor for `FileOutputStream` can throw a `FileNotFoundException`. If it does not, the constructor for `ObjectOutputStream` can throw an `IOException`. |
| `public void writeInt(int n) throws IOException`<br>Writes the `int` value `n` to the output stream. |
| `public void writeLong(long n) throws IOException`<br>Writes the `long` value `n` to the output stream. |
| `public void writeDouble(double x) throws IOException`<br>Writes the `double` value `x` to the output stream. |
| `public void writeFloat(float x) throws IOException`<br>Writes the `float` value `x` to the output stream. |
| `public void writeChar(int c) throws IOException`<br>Writes a `char` value to the output stream. Note that the parameter type of `c` is `int`. However, Java will automatically convert a `char` value to an `int` value for you. So the following is an acceptable invocation of `writeChar`:<br>`outputStream.writeChar('A');` |
| `public void writeBoolean(boolean b) throws IOException`<br>Writes the `boolean` value `b` to the output stream. |
| `public void writeUTF(String aString) throws IOException`<br>Writes the string `aString` to the output stream. `UTF` refers to a particular method of encoding the string. To read the string back from the file, you should use the method `readUTF` of the class `ObjectInputStream`. These topics are discussed in the next section. |

```
public void writeObject(Object anObject) throws IOException,
                NotSerializableException, InvalidClassException
```

Writes `anObject` to the output stream. The argument should be an object of a serializable class, a concept discussed later in this chapter. Throws a `NotSerializableException` if the class of `anObject` is not serializable. Throws an `InvalidClassException` if there is something wrong with the serialization. The method `writeObject` is covered later in this chapter.

```
public void close() throws IOException
```

Closes the stream's connection to a file.

**FIGURE 10.6 Some Methods in the Class `ObjectInputStream` *(part 1 of 2)***

`ObjectInputStream(InputStream streamObject)`

Creates an input stream that is connected to the specified binary file. There is no constructor that takes a file name as an argument. If you want to create a stream by using a file name, you use either

```
new ObjectInputStream(new FileInputStream(File_Name))
```

or, using an object of the class `File`,

```
new ObjectInputStream(new FileInputStream(
                      new File(File_Name)))
```

The constructor for `FileInputStream` can throw a `FileNotFoundException`. If it does not, the constructor for `ObjectInputStream` can throw an `IOException`.

---

`public int readInt() throws EOFException, IOException`

Reads an `int` value from the input stream and returns that `int` value. If `readInt` tries to read a value from the file that was not written by the method `writeInt` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. If the read goes beyond the end of the file, an `EOFException` is thrown.

---

`public long readLong() throws EOFException, IOException`

Reads a `long` value from the input stream and returns that `long` value. If `readLong` tries to read a value from the file that was not written by the method `writeLong` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. If the read goes beyond the end of the file, an `EOFException` is thrown.

Note that you cannot write an integer using `writeLong` and later read the same integer using `readInt`, or to write an integer using `writeInt` and later read it using `readLong`. Doing so will cause unpredictable results.

```
public double readDouble() throws EOFException, IOException
```

Reads a `double` value from the input stream and returns that `double` value. If `readDouble` tries to read a value from the file that was not written by the method `writeDouble` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. If the read goes beyond the end of the file, an `EOFException` is thrown.

```
public float readFloat() throws EOFException, IOException
```

Reads a `float` value from the input stream and returns that `float` value. If `readFloat` tries to read a value from the file that was not written by the method `writeFloat` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. If the read goes beyond the end of the file, an `EOFException` is thrown.

Note that you cannot write a floating-point number using `writeDouble` and later read the same number using `readFloat`, or write a floating-point number using `writeFloat` and later read it using `readDouble`. Doing so will cause unpredictable results, as will other type mismatches, such as writing with `writeInt` and then reading with `readFloat` or `readDouble`.

*(continued)*

```
public char readChar() throws EOFException, IOException
```

Reads a `char` value from the input stream and returns that `char` value. If `readChar` tries to read a value from the file that was not written by the method `writeChar` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. If the read goes beyond the end of the file, an `EOFException` is thrown.

| |
|---|
| `public boolean readBoolean() throws EOFException, IOException`<br>Reads a `boolean` value from the input stream and returns that `boolean` value. If `readBoolean` tries to read a value from the file that was not written by the method `writeBoolean` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. If the read goes beyond the end of the file, an `EOFException` is thrown. |
| `publicString readUTF() throws IOException, UTFDataFormatException`<br>Reads a `String` value from the input stream and returns that `String` value. If `readUTF` tries to read a value from the file that was not written by the method `writeUTF` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. One of the exceptions `UTFDataFormatException` or `IOException` can be thrown. |
| `Object readObject() throws ClassNotFoundException, InvalidClassException, OptionalDataException, IOException`<br>Reads an object from the input stream. Throws a `ClassNotFoundException` if the class of a serialized object cannot be found. Throws an `InvalidClassException` if something is wrong with the serializable class. Throws an `OptionalDataException` if a primitive data item, instead of an object, was found in the stream. Throws an `IOEx-ception` if there is some other I/O problem. The method `readObject` is covered in Section 10.5. |
| `public void close() throws IOException`<br>Closes the stream's connection to a file. |

## LISTING 10.6 Using `ObjectInputStream` to Read from a File *(part 1 of 2)*

*Assumes the program in Listing 10.4 was already run.*

```
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.EOFException;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;
```

**LISTING 10.6 Using `ObjectInputStream` to Read from a File** *(part 2 of 2)*

```
public class BinaryInputDemo
{
    public static void main(String[] args)
    {
        String fileName = "numbers.dat";
        try
        {
            ObjectInputStream inputStream =
                  new ObjectInputStream(new FileInputStream(fileName));
            System.out.println("Reading the nonnegative integers");
            System.out.println("in the file " + fileName);
            int anInteger = inputStream.(readInt);
            while (anInteger >= 0)
            {
                System.out.ptintln(anInteger);
                anInteger = inputStream.readInt();
            }
            System.out.println("End of reading from file.");
            inputStream.close();
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Problem opening the file " + fileName);
        }
        catch(EOFException e)
        {
            System.out.println("Problem reading the file " + fileName);
            System.out.println("Reached end of the file.");
        }
        catch(IOException e)
        {
            System.out.println("Problem reading the file " + fileName);
        }
    }
}
```

### *Screen Output*

```
Reading the nonnegative integers
in the file number.dat
1
2
3
End of reading from file.
```

*Notice that the sentinel value* `-1` *is read from the file but is not displayed on the screen.*

## LISTING 10.7 Using EOFException *(part 1 of 2)*

*Assumes the program in Listing 10.4 was already run.*

```
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.EOFException;
import java.io.FileNotFoundException;
import java.io.IOException;

    public class EOFExceptionDemo
    {
        public static void main(String[] args)
        {

            String fileName = "numbers.dat";
```

```
        try
        {
            ObjectInputStream inputStream =
                 new ObjectInputStream(new
                 FileInputStream(fileName));
            System.out.println("Reading ALL the integers");
            System.out.printin("in the file " + fileName);
            try
            {
                while (true)
                {

                    int anInteger = inputStream.readInt();
                    System.out.println(anInteger);
                }
            }
            catch(EOFException e)
            {
                 System.out.println("End of reading from file.");
            }
            inputStream.close();
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Cannot find file " + fileName);
        }
        catch(IOException e)
        {
            System.out.println("Problem with input from file " +
            fileName);
        }
    }
}
```

*The loop ends when an exception is thrown.*

### *Screen Output*

```
Reading ALL the integers
in the file numbers.dat
1
2
3
-1
End of reading from file.
```

*When you use* `EOFException` *to end reading, you can read files that contain any kind of integers, including the* `-1` *here, which is treated just like any other integer.*

## LISTING 10.8 Processing a File of Binary Data *(part 1 of 3)*

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.EOFException;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;

public class Doubler
{
    private ObjectInputStream inputStream = null;
    private ObjectOutputStream outputStream = null;

    /**
    Doubles the integers in one file and puts them in another file.
    */
    public static void main(String[] args)
    {
        Doubler twoTimer = new Doubler();
        twoTimer.connectToInputFile();
        twoTimer.connectToOutputFile();
        twoTimer.timesTwo();
        twoTimer.closeFiles();
        System.out.println("Numbers from input file");
        System.out.println("doubled and copied to output file.");
    }
```

```java
public void connectToInputFile()
{
    String inputFileName =
        getFileName("Enter name of input file:");
    try
    {
        inputStream = new ObjectInputStream(
                          new FileInputStream(inputFileName));
    }
    catch(FileNotFoundException e)
    {
        System.out.println("File " + inputFileName +
                           " not found.");
        System.exit(0);
    }
    catch(IOException e)
    {
        System.out.println("Error opening input file" +
                           inputFileName);

         System.exit(0);
    }
}
```

```java
private String getFileName(String prompt)
{
    String fileName = null;
    System.out.println(prompt);
    Scanner keyboard = new Scanner(System.in);
    fileName = keyboard.next();

    return fileName;
}
public void connectToOutputFile()
{
    String outputFileName =
                        getFileName("Enter name of output file:");
    try
    {
        outputStream = new ObjectOutputStream(
                            new FileOutputStream(outputFileName));
    }
    catch(IOException e)
    {
        System.out.println("Erroropeningoutputfile" +
                            outputFileName);
        System.out.println(e.getMessage());
        System.exit(0);
    }
}
```

*A class used in a real-life application would usually transform the input data in a more complex way before writing it to the output file.* ***Such a class likely would have*** *additional methods.*

```
public void timesTwo()
{
    try
    {
        while (true)
        {
            int next = inputStream.readInt();
            outputStream.writeInt(2 * next);
        }
    }
```

```
    catch(EOFException e)
    {
        //Do nothing. This just ends the loop.
    }
    catch(IOException e)
    {
        System.out.println(
                        "Error: reading or writing files.");
        System.out.println(e.getMessage());
        System.exit(0);
    }
    }
    public void closeFiles()
    {
        try
        {
            inputStream.close();
            outputStream.close();
        }
        catch(IOException e)
        {
            System.out.println("Error closing files " +
                                e.getMessage());
            System.exit(0);
        }
    }
}
```

## LISTING 10.9 The Class `Species` Serialized for Binary-File I/O

*This is a new, improved definition of the class* `Species` *and replaces the* ***definition in*** *Listing 5.19 of Chapter 5.*

```
import java.io.Serializable;
import java.util.Scanner;

/**
Serialized class for data on endangered species.
*/
public class Species implements Serializable
{
    private String name;
    private int population;
    private double growthRate;

    public Species()
    {
        name = null;
        population = 0;
        growthRate = 0;
    }
```

*These two words and the* `import` *statement make this class serializable.*

```
    public Species(String initialName, int initialPopulation,
                   double initialGrowthRate)
    {
        name = initialName;
        if (initialPopulation >= 0)
            population = initialPopulation;
        else
        {
            System.out.println("ERROR: Negative population.");
            System.exit(0);
        }

        growthRate = initialGrowthRate;
    }
    public String toString()
    {
        return ("Name = " + name + "\n" +
                "Population = " + population + "\n" +
                "Growth rate = " + growthRate + "%");
    }
        <Other methods are the same as those in Listing 5.19 of Chapter 5,
        but they are not needed for the discussion in this chapter.>
}
```

## LISTING 10.10 File I/O of Class Objects *(part 1 of 3)*

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class ClassObjectIODemo
{
    public static void main(String[]args)
    {
        ObjectOutputStream outputStream = null;
        String fileName = "species.records";

        try
        {
            outputStream = new ObjectOutputStream(
                           new FileOutputStream(fileName));
        }
        catch(IOException e)
        {
            System.out.println("Error opening output file " +
                               fileName + ".");
            System.exit(0);
        }
        Species califCondor =
                    new Species("Calif. Condor", 27, 0.02);
        Species blackRhino =
                    new Species("Black Rhino", 100, 1.0);
```

```
try
{
    outputStream.writeObject(califCondor);
    outputStream.writeObject(blackRhino);
    outputStream.Close();
}
catch(IOException e)
{
    System.out.println("Error wring to file " +
                       fileName + ".");
    System.exit(0)
}

System.out.println("Records sent to file " +
                   fileName + ".");
System.out.println(
      "Now let's reopen the file and echo " +
      "the records.");
```

```
ObjectInputStream inputStream = null;

try
{
    inputStream = new ObjectInputStream(
                  new FileInputStream("species.records"));
}
catch(IOException e)
{
    System.out.println("Error opening input file " +
                       fileName + ".");
    System.exit(0);
}
Species readOne = null, readTwo = null;
```

```
        try
        {
            readOne = (Species)inputStream.readObject();
            readTwo = (Species)inputStream.readObject();
            inputStream.close();
        }

        catch(Exception e)
        {
            System.out.println("Error reading from file " +
                                fileName + ".");
            System.exit(0);
        }

        System.out.println("The following were read\n" +
                           "from the file " + fileName + ".");
        System.out.println(readOne);
        System.out.println();
        System.out.println(readTwo);

        System.out.println("End of program.");
    }
}
```

*Notice the type casts.*

*A separate* `catch` *block for each type of exception would be better. We use only one to save space.*

### *Sample Screen Output*

```
Records sent to file species.records.
Now let's reopen the file and echo the records.
The following were read
from the file species.records.
Name = Calif. Condor
Population = 27
Growth rate = 0.02%
```

```
Name = Black Rhino
Population = 100
Growth rate 1.0%
End of program.
```

## LISTING 10.11 File I/O of an Array Object *(part 1 of 2)*

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class ArrayIODemo
{
    public static void main(String[] args)
    {
        Species[] oneArray = new Species[2];
        oneArray[0] = new Species("Calif. Condor", 27, 0.02);
        oneArray[1] = new Species("Black Rhino", 100, 1.0);

        String fileName = "array.dat";

        try
        {
            ObjectOutputStream outputStream =
                  new ObjectOutputStream(
                      new FileOutputStream(fileName));
            outputStream.writeobject(oneArray);
            outputStream.close();
        }
        catch(IOException e)
        {
            System.out.println("Error writing to file " +
                                fileName + ".");
            System.exit(0);
        }
        System.out.println("Array written to file " +
                            fileName + " and file is closed.");
```

```
        System.out.println("Open the file for input and " +
                           "echo the array.")
        Species[] anotherArray = null;
        try
        {
            ObjectInputStream inputStream =
                        new objectInputStream(
                                new FileInputStream(fileName));
            anotherArray = (Species[])inputStream.readobject();
            inputStream.close();
        }
        catch(Exception e)
        {
            System.out.println("Error reading file " +
                               fileName + ".");
            System.exit(0);
        }
        System.out.println("The following were read from " +
                           "the file " + fileName + ":");
        for (int i = 0; i < anotherArray.length; i++)
        {
            System.out.println(anotherArray[i]);
            System.out.println();
        }
        System.out.println("End of program.");
    }
}
```

*Note the type cast*

*A separate* `catch` *block for each type of exception would be better. We use only one to save space.*

### *Sample Screen Output*

```
Array written to file array.dat and file is closed.
Open the file for input and echo the array.
The following were read from the file array.dat:
Name = Calif. Condor
Population = 27
Growth rate = 0.02%

Name = black Rhino
Population = 100
Growth rate = 1.0%

End of program.
```

## LISTING 10.12 A File Organizer GUI *(part 1 of 4)*

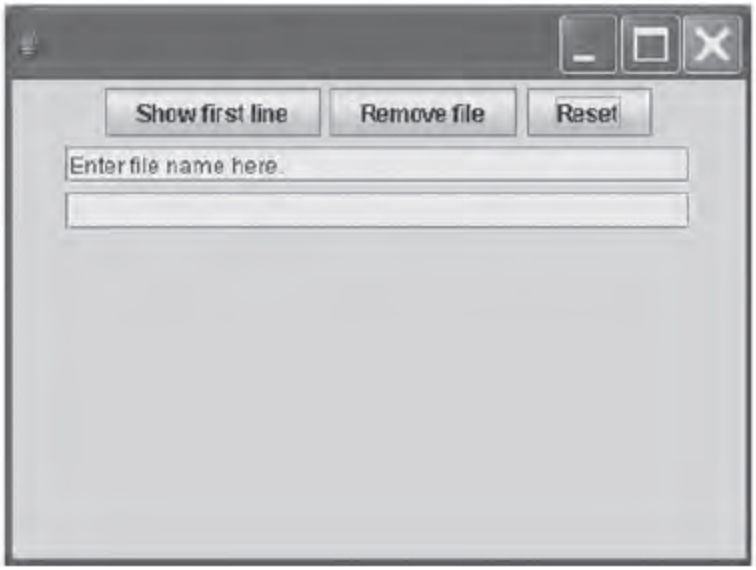
```
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTextField;
import java.awt.Color;
import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class FileOrganizer extends JFrame implements ActionListener
{
    public static final int WIDTH = 400;
    public static final int HEIGHT = 300;
    public static final int NUMBER_OF_CHAR = 30;

    private JTextField fileNameField;
    private JTextField firstLineField;
```

```
    public FileOrganizer()
    {
        setSize(WIDTH, HEIGHT);
        WindowDestroyer listener = new WindowDestroyer();
        addWindowListener(listener);
        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());

        JButton showButton = new JButton("Show first line");
        showButton.addActionListener(this);
        contentPane.add(showButton);

        JButton removeButton = new JButton("Remove file");
        removeButton.addActionListener(this);
        contentPane.add(removeButton);

        JButton resetButton = new JButton("Reset");
        resetButton.addActionListener(this);
        contentPane.add(resetButton);

        fileNameField = new JTextField(NUMBER_OF_CHAR);
        contentPane.add(fileNameField);
        fileNameField.setText("Enter file name here.");

        firstLineField = new JTextField(NUMBER_OF_CHAR);
        contentPane.add(firstLineField);
}
```

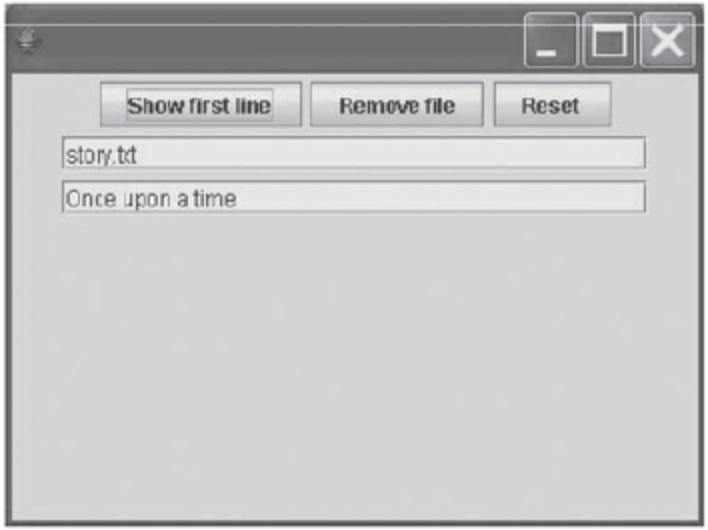
```
public void actionPerformed(ActionEvent e)
{
    String actionCommand = e.getActionCommand();
    if (actionCommand.equals("Show first line"))
        showFirstLine();
    else if (actionCommand.equals("Remove file"))
             removeFile();
    else if (actionCommand.equals("Reset"))
        resetFields();
    else
        firstLineField.setText("Unexpected error.");
}
private void showFirstLine()
{
    Scanner fileInput = null;
    String fileName = fileNameField.getText();
    File fileObject = new File(fileName);

    if (!fileObject.exists())
        firstLineField.setText("No such file");
    else if (!fileObject.canRead())
        firstLineField.setText("That file is not readable.");
    else
    {
        try
        {
            fileInput = new Scanner(fileObject);
        }
        catch(FileNotFoundException e)
        {
            firstLineField.setText("Error opening the file " +
                                   fileName);
        }
        String firstLine = fileInput.nextLine();
        firstLineField.setText(firstLine);
        fileInput.close();
    }
}
```

```
    private void resetFields()
    {
        fileNameField.setText("Enter file name here.");
        firstLineField.setText("");
    }
    private void removeFile()
    {
        Scanner fileInput = null;
        String firstLine;
        String fileName = fileNameField.getText();
        File fileObject = new File(filename);

        if (!fileObject.exists())
            firstLinefield.setText("No such file");
        else if (!fileObject.canWrite())
            firstLineField.setText("Permission denied.");
        else
        {
            if (fileObject.delete())
                firstLineField.setText("File deleted.");
            else
                firstLineField.setText("Could not delete file.");
        }
    }
    public static void main(String[] args)
    {
        FileOrganizer gui = new FileOrganizer();
        gui.setVisible(true);

    }
}
```

Screen Output Showing GUI's State Initially or After the Reset Button is Clicked

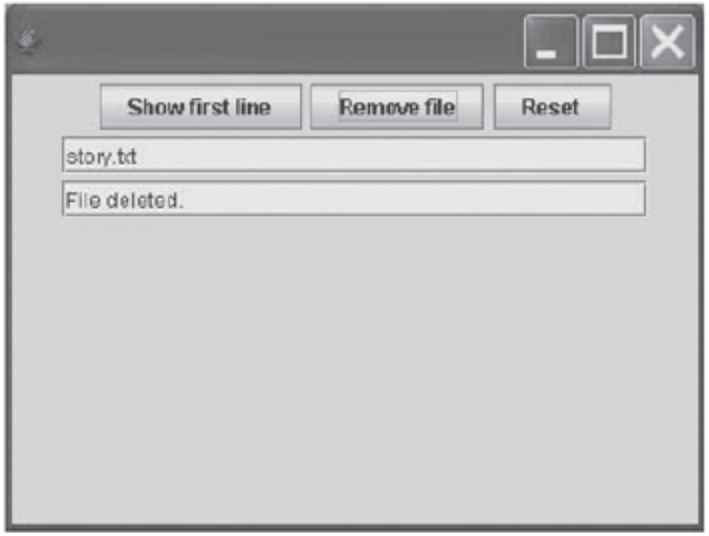


Screen Output After Entering the File Name and Clicking the Show `first line` Button

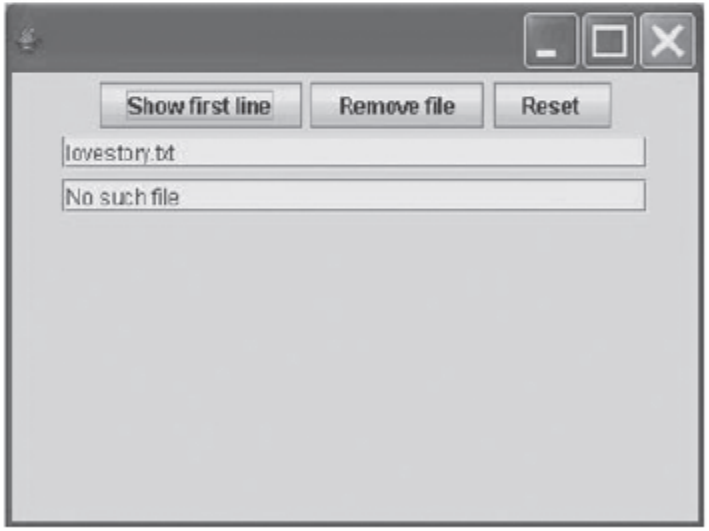



*Assumes that the first line in the file is as shown*

Screen Output After Entering the Remove `line` Button




Screen Output After Entering the File Name and Clicking the Show `first line` Button




*Assumes that the named file does not exist*

## FIGURE 10.7A GUI for Programming Project 14