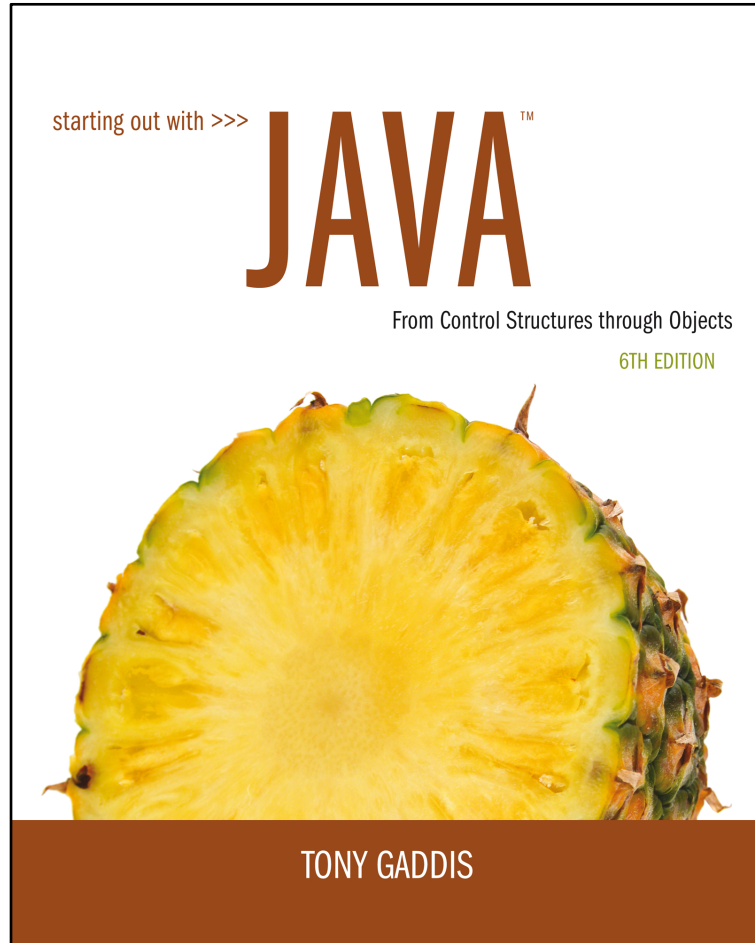


Starting Out with Java: From Control Structures Through Objects

Sixth Edition



Chapter 11

I/O

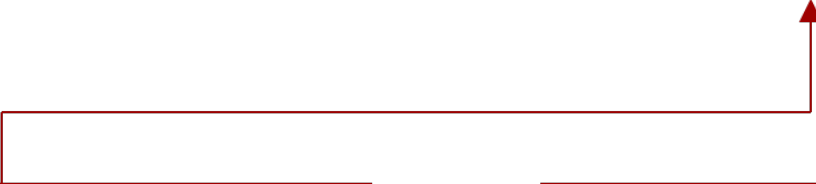
File Input and Output

- Reentering data all the time could get tedious for the user.
- The data can be saved to a file.
 - Files can be **input files** or **output files**.
- Files:
 - Files have to be opened.
 - Data is then written to the file.
 - The file must be closed prior to program termination.
- In general, there are two types of files:
 - binary
 - text

Writing Text to a File

- To open a file for text output you create an instance of the `PrintWriter` class.

```
PrintWriter outputFile = new PrintWriter("StudentData.txt");
```



Pass the name of the file that you wish to open as an argument to the `PrintWriter` constructor.

Warning: if the file already exists, it will be erased and replaced with a new file.

The `PrintWriter` Class (1 of 3)

- The `PrintWriter` class allows you to write data to a file using the `print` and `println` methods, as you have been using to display data on the screen.
- Just as with the `System.out` object, the `println` method of the `PrintWriter` class will place a newline character after the written data.
- The `print` method writes data without writing the newline character.

The PrintWriter Class (2 of 3)

Open the file.

```
PrintWriter outputFile = new PrintWriter("Names.txt");  
outputFile.println("Chris");  
outputFile.println("Kathryn");  
outputFile.println("Jean");  
outputFile.close();
```

Close the file.

Write data to the file.

The `PrintWriter` Class (3 of 3)

- To use the `PrintWriter` class, put the following `import` statement at the top of the source file:

```
import java.io.*;
```

- See example: `FileWriteDemo.java`

Exceptions (1 of 3)

- When something unexpected happens in a Java program, an **exception** is thrown.
- The method that is executing when the exception is thrown must either handle the exception or pass it up the line.
- Handling the exception will be discussed later.
- To pass it up the line, the method needs a `throws` clause in the method header.

Exceptions (2 of 3)

- To insert a `throws` clause in a method header, simply add the word **throws** and the name of the expected exception.
- `PrintWriter` objects can throw an `IOException`, so we write the `throws` clause like this:

```
public static void main(String[] args) throws IOException
```


Appending Text to a File

- To avoid erasing a file that already exists, create a `FileWriter` object in this manner:

```
FileWriter fw =  
    new FileWriter("names.txt", true);
```

- Then, create a `PrintWriter` object in this manner:

```
PrintWriter fw = new PrintWriter(fw);
```

Specifying a File Location (1 of 2)

- On a Windows computer, paths contain backslash (\) characters.
- Remember, if the backslash is used in a string literal, it is the escape character so you must use two of them:

```
PrintWriter outFile =  
    new PrintWriter("A:\\PriceList.txt");
```

Specifying a File Location (2 of 2)

- This is only necessary if the backslash is in a string literal.
- If the backslash is in a `String` object then it will be handled properly.
- Fortunately, Java allows Unix style filenames using the forward slash (/) to separate directories:

```
PrintWriter outFile = new  
    PrintWriter("/home/rharrison/names.txt");
```

Reading Data From a File (1 of 3)

- You use the `File` class and the `Scanner` class to read data from a file:

Pass the name of the file as an argument to the `File` class constructor.

```
File myFile = new File("Customers.txt");  
Scanner inputFile = new Scanner(myFile);
```

Pass the `File` object as an argument to the `Scanner` class constructor.

Reading Data From a File (2 of 3)

```
Scanner keyboard = new Scanner(System.in);  
System.out.print("Enter the filename: ");  
String filename = keyboard.nextLine();  
File file = new File(filename);  
Scanner inputFile = new Scanner(file);
```

- The lines above:
 - Creates an instance of the `Scanner` class to read from the keyboard
 - Prompt the user for a filename
 - Get the filename from the user
 - Create an instance of the `File` class to represent the file
 - Create an instance of the `Scanner` class that reads from the file

Reading Data From a File (3 of 3)

- Once an instance of `Scanner` is created, data can be read using the same methods that you have used to read keyboard input (`nextLine`, `nextInt`, `nextDouble`, etc).

```
// Open the file.  
File file = new File("Names.txt");  
Scanner inputFile = new Scanner(file);  
// Read a line from the file.  
String str = inputFile.nextLine();  
// Close the file.  
inputFile.close();
```

Exceptions (3 of 3)

- The `Scanner` class can throw an `IOException` when a `File` object is passed to its constructor.
- So, we put a `throws IOException` clause in the header of the method that instantiates the `Scanner` class.
- See Example: `ReadFirstLine.java`

Detecting the End of a File (1 of 2)

- The `Scanner` class's `hasNext()` method will return true if another item can be read from the file.

```
// Open the file.
File file = new File(filename);
Scanner inputFile = new Scanner(file);
// Read until the end of the file.
while (inputFile.hasNext())
{
    String str = inputFile.nextLine();
    System.out.println(str);
}
inputFile.close(); // close the file when done.
```


Detecting the End of a File (2 of 2)

- See example: `FileReadDemo.java`

Binary Files (1 of 6)

- The way data is stored in memory is sometimes called the **raw binary format**.
- Data can be stored in a file in its raw binary format.
- A file that contains binary data is often called a **binary file**.
- Storing data in its binary format is more efficient than storing it as text.
- There are some types of data that should only be stored in its raw binary format.

Binary Files (2 of 6)

- Binary files cannot be opened in a text editor such as Notepad.
- To write data to a binary file you must create objects from the following classes:
 - **FileOutputStream** - allows you to open a file for writing binary data. It provides only basic functionality for writing bytes to the file.
 - **DataOutputStream** - allows you to write data of any primitive type or String objects to a binary file. Cannot directly access a file. It is used in conjunction with a `FileOutputStream` object that has a connection to a file.

Binary Files (3 of 6)

- A `DataOutputStream` object is wrapped around a `FileOutputStream` object to write data to a binary file.

```
FileOutputStream fstream = new  
    FileOutputStream("MyInfo.dat");  
DataOutputStream outputFile = new  
    DataOutputStream(fstream);
```

- If the file that you are opening with the `FileOutputStream` object already exists, it will be erased and an empty file by the same name will be created.

Binary Files (4 of 6)

- These statements can be combined into one.

```
DataOutputStream outputFile = new  
DataOutputStream(new  
    FileOutputStream("MyInfo.dat"));
```

- Once the `DataOutputStream` object has been created, you can use it to write binary data to the file.
- Example: `WriteBinaryFile.java`

Binary Files (5 of 6)

- To open a binary file for input, you wrap a `DataInputStream` object around a `FileInputStream` object.

```
FileInputStream fstream = new
    FileInputStream("MyInfo.dat");
DataInputStream inputFile = new
    DataInputStream(fstream);
```

- These two statements can be combined into one

```
DataInputStream inputFile = new
    DataInputStream(new
        FileInputStream("MyInfo.dat"));
```

Binary Files (6 of 6)

- The `FileInputStream` constructor will throw a `FileNotFoundException` if the file named by the string argument cannot be found.
- Once the `DataInputStream` object has been created, you can use it to read binary data from the file.
- Example:
 - `ReadBinaryFile.java`

Writing and Reading Strings (1 of 2)

- To write a string to a binary file, use the `DataOutputStream` class's `writeUTF` method.
- This method writes its `String` argument in a format known as **UTF-8 encoding**.
 - Just before writing the string, this method writes a two-byte integer indicating the number of bytes that the string occupies.
 - Then, it writes the string's characters in Unicode. (UTF stands for Unicode Text Format.)
- The `DataInputStream` class's `readUTF` method reads from the file.

Writing and Reading Strings (2 of 2)

- To write a string to a file:

```
String name = "Chloe";  
outputFile.writeUTF(name);
```

- To read a string from a file:

```
String name = inputFile.readUTF();
```

- The `readUTF` method will correctly read a string only when the string was written with the `writeUTF` method.
- Example:
 - Write UTF.java
 - Read UTF.java

Appending Data to Binary Files

- The `FileOutputStream` constructor takes an optional second argument which must be a `boolean` value.
- If the argument is `true`, the file will not be erased if it exists; new data will be written to the end of the file.
- If the argument is `false`, the file will be erased if it already exists.

```
FileOutputStream fstream = new  
    FileOutputStream("MyInfo.dat", true);  
DataOutputStream outputFile = new  
    DataOutputStream(fstream);
```

Random Access Files (1 of 5)

- Text files and the binary files previously shown use **sequential file access**.
- With sequential access:
 - The first time data is read from the file, the data will be read from its beginning.
 - As the reading continues, the file's read position advances sequentially through the file's contents.
- Sequential file access is useful in many circumstances.
- If the file is very large, locating data buried deep inside it can take a long time.

Random Access Files (2 of 5)

- Java allows a program to perform **random file access**.
- In random file access, a program may immediately jump to any location in the file.
- To create and work with random access files in Java, you use the `RandomAccessFile` class.

```
RandomAccessFile(String filename, String mode)
```

- *filename*: the name of the file.
- *mode*: a string indicating the mode in which you wish to use the file.
 - “r” = reading
 - “rw” = for reading and writing.

Random Access Files (3 of 5)

```
// Open a file for random reading.  
RandomAccessFile randomFile = new  
    RandomAccessFile("MyData.dat", "r");  
// Open a file for random reading and writing.  
RandomAccessFile randomFile = new  
    RandomAccessFile("MyData.dat", "rw");
```

- When opening a file in “r” mode where the file does not exist, a `FileNotFoundException` will be thrown.
- Opening a file in “r” mode and trying to write to it will throw an `IOException`.
- If you open an existing file in “rw” mode, it will not be deleted and the file’s existing content will be preserved.

Random Access Files (4 of 5)

- Items in a sequential access file are accessed one after the other.
- Items in a random access file are accessed in any order.
- If you open a file in “rw” mode and the file does not exist, it will be created.
- A file that is opened or created with the `RandomAccessFile` class is treated as a binary file.

Random Access Files (5 of 5)

- The `RandomAccessFile` class has:
 - the same methods as the `DataOutputStream` class for writing data, and
 - the same methods as the `DataInputStream` class for reading data.
- The `RandomAccessFile` class can be used to sequentially process a binary file.
- Example: `WriteLetters.java`

The File Pointer (1 of 5)

- The `RandomAccessFile` class treats a file as a stream of bytes.
- The bytes are numbered:
 - the first byte is byte 0.
 - The last byte's number is one less than the number of bytes in the file.
- These byte numbers are similar to an array's subscripts, and are used to identify locations in the file.
- Internally, the `RandomAccessFile` class keeps a long integer value known as the **file pointer**.

The File Pointer (2 of 5)

- The **file pointer** holds the byte number of a location in the file.
- When a file is first opened, the file pointer is set to 0.
- When an item is read from the file, it is read from the byte that the file pointer points to.
- Reading also causes the file pointer to advance to the byte just beyond the item that was read.
- If another item is immediately read, the reading will begin at that point in the file.

The File Pointer (3 of 5)

- An `EOFException` is thrown when a read causes the file pointer to go beyond the size of the file.
- Writing also takes place at the location pointed to by the file pointer.
- If the file pointer points to the end of the file, data will be written to the end of the file.
- If the file pointer holds the number of a byte within the file, at a location where data is already stored, a write will overwrite the data at that point.

The File Pointer (4 of 5)

- The `RandomAccessFile` class lets you move the file pointer.
- This allows data to be read and written at any byte location in the file.

- The `seek` method is used to move the file pointer.

```
rndFile.seek(long position);
```

- The argument is the number of the byte that you want to move the file pointer to.

The File Pointer (5 of 5)

```
RandomAccessFile file = new  
    RandomAccessFile("MyInfo.dat", "r");  
file.seek(99);  
byte b = file.readByte();
```

- Example: ReadRandomLetters.java