Advanced Programming

Code: SWEG2033

Chapter Three

Streams and File I/O

Introduction

- Data stored in variables, arrays, and objects are temporary.
- To permanently store the data created in a program, you need to save them in a file on a disk or a CD.
- Computers use files for long-term retention of large amounts of data.
- We refer to data maintained in files as **persistent data.**
- In this chapter, we discuss Java's powerful file-processing and stream input/output features.

Introduction

- The java.io package perform input and output (I/O) in Java.
- A stream can be defined as a sequence of data. There are two kinds of Streams.
  - **InPutStream** − The InputStream is used to read data from a source.
  - **OutPutStream** − The OutputStream is used for writing data to a destination.

Data Hierarchy

- **Bit -** smallest data item in a computer
- **character set** is the set of all the characters used to write programs and represent data items. Characters in Java are Unicode characters composed of two bytes.
- Java contains a data type, byte, that can be used to represent byte data.
- **Fields** are composed of characters or bytes(E.g. Name)
- **Record** is a group of related fields (E.g Id, name, sex, etc for employee record)
- A **file** is a group of related records. E.g all employee records of an organization.
- **Database** – group of related files.

Files and Streams

- Java views each file as a sequential stream of bytes.
- The term “stream” refers to ordered data that is read from or written to a file.
- File streams can be used to input and output data as either characters or bytes.
- Streams that input and output bytes to files are known as byte-based streams.
- Streams that input and output characters to files are known as character-based streams.
- Files that are created using byte-based streams are referred to as binary files.
- Files created using character-based streams are referred to as text files.

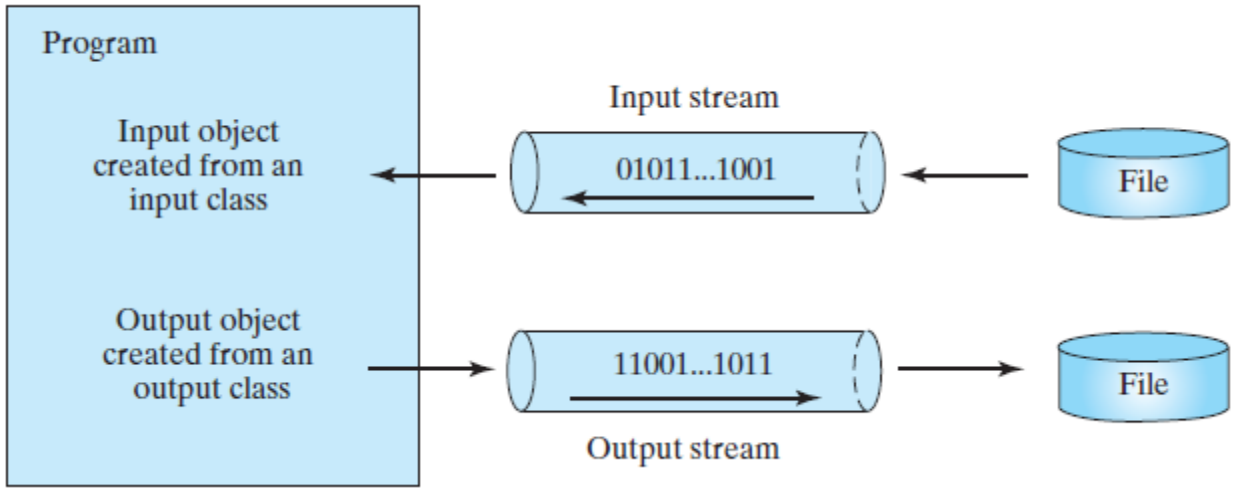
Files and Streams (cont'd)

Java's view of a file of *n bytes.*

Files and Streams (cont'd)

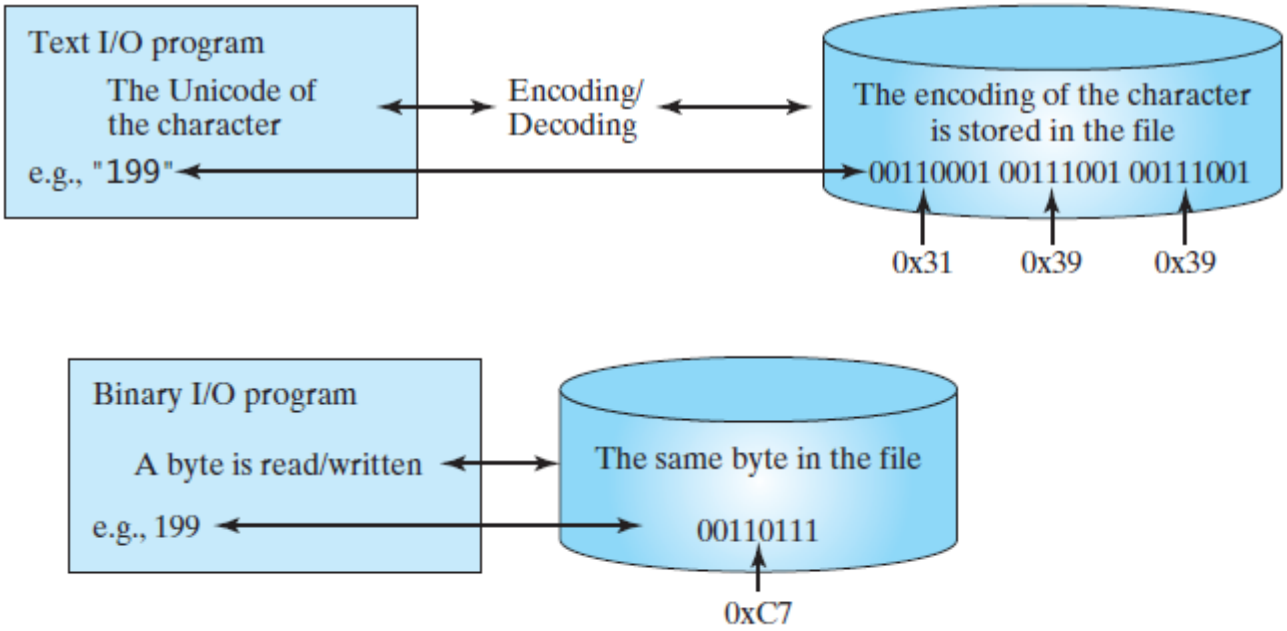
- Computers do not differentiate binary files and text files.
- All files are stored in binary format, and thus all files are essentially binary files.
- Encoding and decoding are automatically performed for text I/O.

Files and Streams (cont'd)



- The program receives data through an input object and sends data through an output object.

Files and Streams (cont'd)



Text I/O requires encoding and decoding, whereas binary I/O does not.

Files and Streams (cont'd)

- Binary I/O is more efficient than text I/O, because binary I/O does not require encoding and decoding.
- Binary files are independent of the encoding scheme on the host machine and thus are portable.
- Java programs on any machine can read a binary file created by a Java program.
- This is why Java class files are binary files. Java class files can run on a JVM on any machine.

Files and Streams (cont'd)

Byte Streams

- Java byte streams are used to perform input and output of 8-bit bytes.
- Classes : **FileInputStream** and **FileOutputStream**

```
import java.io.*;
public class CopyFile {

   public static void main(String args[]) throws IOException {
      FileInputStream in = null;
      FileOutputStream out = null;

      try {
         in = new FileInputStream("input.txt");
         out = new FileOutputStream("output.txt");

         int c;
         while ((c = in.read()) != -1) {
            out.write(c);
         }
      }finally {
         if (in != null) {
            in.close();
         }
         if (out != null) {
            out.close();
         }
      }
   }
}
```

Files and Streams (cont'd)

Character Streams

- **Character** streams are used to perform input and output for 16-bit unicode.
- Classes : **FileReader** and **FileWriter**.

```
import java.io.*;
public class CopyFile {

   public static void main(String args[]) throws IOException {
      FileReader in = null;
      FileWriter out = null;

      try {
         in = new FileReader("input.txt");
         out = new FileWriter("output.txt");

         int c;
         while ((c = in.read()) != -1) {
            out.write(c);
         }
      }finally {
         if (in != null) {
            in.close();
         }
         if (out != null) {
            out.close();
         }
      }
   }
}
```

Files and Streams (cont'd)

Standard Streams

- All the programming languages provide support for standard I/O where the user's program can take input from a keyboard and then produce an output on the computer screen.
- There are three standard streams.
  - **Standard Input** − This is used to feed the data to user's program and usually a keyboard is used as standard input stream and represented as **System.in**.
  - **Standard Output** − This is used to output the data produced by the user's program and usually a computer screen is used for standard output stream and represented as **System.out**.
  - **Standard Error** − This is used to output the error data produced by the user's program and usually a computer screen is used for standard error stream and represented as **System.err**.

Class Files

- Useful for retrieving information about files or directories from disk.
- The File class is an abstract representation of file and directory pathname.
- A pathname can be either absolute or relative.
- The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

File Input and Output

- A File object encapsulates the properties of a file or a path but does not contain the methods for creating a file or for reading/writing data from/to a file.
- In order to perform I/O, you need to create objects using appropriate Java I/O classes. The objects contain the methods for reading/writing data from/to a file.

File Input and Output

- Writing Data Using PrintWriter
  - The java.io.PrintWriter class can be used to create a file and write data to a text file.
  - First, you have to create a PrintWriter object for a text file as follows:
    - PrintWriter output = new PrintWriter(filename);
  - Then, you can invoke the print, println, and printf methods on the PrintWriter object to write data to a file.

FileInputStream/FileOutputStream

- FileInputStream/FileOutputStream is for reading/writing bytes from/to files.
- A java.io.FileNotFoundException will occur if you attempt to create a FileInputStream with a nonexistent file.

The Serializable Interface

- Serialization is the process of transforming an object into a stream of bytes.
- Deserialization is the reverse prorcess.
- Objects of classes that implement the java.io.Seralizable interface can be serializaed and deserialized.
- Serialization allows objects to be easily saved to files or sent to remote hosts over a network.
- Classes whose instances to be stored in files or sent to remote hosts should implement the java.io.Seralizable interfaces.
- ObjectInputStream/ObjectOutputStream enables you to perform I/O for objects.

Random-Access Files

- Java provides the RandomAccessFile class to allow a file to be read from and written to at random locations.
- When creating a RandomAccessFile, you can specify one of two modes
  - Mode “r” means that the stream is read-only,
  - and mode “rw” indicates that the stream allows both read and write.

Files Examples

Class File (cont'd)

Example

```
import java.io.File;
public class TestFileClass {
   public static void main(String[] args) {
         File file = new
            File("C:/Users/ABCD/Desktop/realitypod.com_files/welcome.java");
        System.out.println("Does it exist? "+ file.exists() );
        System.out.println("The file has " + file.length() + " bytes");
        System.out.println("Can it be read? " + file.canRead());
        System.out.println("Can it be written? " + file.canWrite());
        System.out.println("Is it a directory? " + file.isDirectory());
        System.out.println("Is it a file? " + file.isFile());
        System.out.println("Is it absolute? " + file.isAbsolute());
        System.out.println("Is it hidden? " + file.isHidden());
        System.out.println("Name of the file? " + file.getName());
        System.out.println("Parent Directory? " + file.getParent());
        System.out.println("Absolute path is " +
        file.getAbsolutePath());
        System.out.println("Last modified on " +
        new java.util.Date(file.lastModified()));
   }
}
```

File Input and Output (cont'd)

- **Example 1**

```
import java.io.*;
public class WriteData {
    public static void main(String[] args) throws Exception {
        File file = new File("score.txt");
        if (file.exists()) {
        System.out.println("File already exists");
        System.exit(0);
        }
         // Create a file
         PrintWriter output = new PrintWriter(file);
        //Write formatted output to the file
         output.print("John T Smith ");
         output.println(90);
         output.print("Eric K Jones ");
         output.println(85);
         // Close the file
          output.close();
      }
}
```

File Input and Output (cont'd)

Example 2

```
import java.io.File;
import java.util.Scanner;
public class ReadData {
    public static void main(String[] args) throws Exception {
        File file = new File("scores.txt");
        if (file.exists()) {
            // Create a Scanner for the file
           Scanner input = new Scanner(file);
           while (input.hasNext()) {
              String firstName = input.next();
              String mi = input.next();
              String lastName = input.next();
              int score = input.nextInt();
               System.out.println(firstName + " " + mi + " " + lastName + " " +
                score);
            }
```

File Input and Output (cont'd)

```
    //close the file
          input.close();
                  }
          else{
                System.out.println("File does not exist");
          }

      }
}
```

File Input and Output (cont'd)

- Example 2

```
import java.util.Scanner;
import javax.swing.JFileChooser;
 public class FileGUI {
    public static void main(String[] args) throws Exception {
        JFileChooser fileChooser = new JFileChooser();
        if (fileChooser.showOpenDialog(null) ==
            JFileChooser.APPROVE_OPTION) {
        // Get the selected file
        java.io.File file = fileChooser.getSelectedFile();
        // Create a Scanner for the file
        Scanner input = new Scanner(file);
        // Read text from the file
        while (input.hasNext()) {
        System.out.println(input.nextLine());
         }
         // Close the file
         input.close();
         }
    }
```

FileInputStream/FileOutputStream

- FileInputStream/FileOutputStream is for reading/writing bytes from/to files.
- A java.io.FileNotFoundException will occur if you attempt to create a FileInputStream with a nonexistent file.
- Example

```
import java.io.*;
public class TestFileStream {
  public static void main(String[] args) throws IOException {
    // Create an output stream to the file
    FileOutputStream output = new FileOutputStream("temp.txt",
    true); //If append is true, data are appended to the existing file.
```

FileInputStream/FileOutputStream(cont'd)

```
    // Output values to the file
    for (int i = 1; i <= 10; i++)
      output.write(i);
    // Close the output stream
    output.close();
    // Create an input stream for the file
    FileInputStream input = new FileInputStream("temp.txt");
     // Read values from the file
    int value;
    while ((value = input.read()) != -1)
      System.out.print(value + " ");

    // Close the output stream
    input.close();

  }
}
```

The Serializable Interface

- Serialization is the process of transforming an object into a stream of bytes.
- Deserialization is the reverse prorcess.
- Objects of classes that implement the java.io.Seralizable interface can be serializaed and deserialized.
- Serialization allows objects to be easily saved to files or sent to remote hosts over a network.
- Classes whose instances to be stored in files or sent to remote hosts should implement the java.io.Seralizable interfaces.
- ObjectInputStream/ObjectOutputStream enables you to perform I/O for objects.

Serializable(cont'd)

- Example

```
import java.io.*;
import java.io.Serializable;
public class TestObjectIOStream implements Serializable {
  public static void main(String[] args) throws ClassNotFoundException,
    IOException {
    // Create an output stream for file object.dat
    ObjectOutputStream output =
      new ObjectOutputStream(new FileOutputStream("object.dat"));
    // Write a string, double value, and object to the file

    output.writeUTF("John");
    output.writeDouble(85.5);
    output.writeObject(new java.util.Date());
    // Close output stream
    output.close();
```

Serializable(cont'd)

```
// Create an input stream for file object.dat
    ObjectInputStream input =
      new ObjectInputStream(new FileInputStream("object.dat"));

    // Write a string, double value, and object to the file

    String name = input.readUTF();
    double score = input.readDouble();
    java.util.Date date = (java.util.Date)(input.readObject());
    System.out.println(name + " " + score + " " + date);

    // Close output stream
    input.close();

  }
}
```

Random-Access Files

- Java provides the RandomAccessFile class to allow a file to be read from and written to at random locations.
- When creating a RandomAccessFile, you can specify one of two modes
  - Mode “r” means that the stream is read-only,
  - and mode “rw” indicates that the stream allows both read and write.

Random-Access Files(cont'd)

Example

```
import java.io.*;
public class TestRandomAccessFile {
  public static void main(String[] args) throws IOException {
    // Create a random access file
    RandomAccessFile inout = new RandomAccessFile("inoutdat", "rw");
    // Write new integers to the file
    for (int i = 0; i < 200; i++)
      inout.writeInt(i);
    // Display the current length of the file
    System.out.println("Current file length is " + inout.length());
    // Retrieve the first number
    inout.seek(0); // Move the file pointer to the beginning
    System.out.println("The first number is " + inout.readInt());
```

Random-Access Files(cont'd)

```
        // Retrieve the second number
        inout.seek(1*4); // Move the file pointer to the second number
        System.out.println("The second number is " + inout.readInt());
        // Retrieve the tenth number
        inout.seek(9*4); // Move the file pointer to the tenth number
        System.out.println("The tenth number is " + inout.readInt());
        // Modify the eleventh number
        inout.writeInt(555);
        // Append a new number
        inout.seek(inout.length()); // Move the file pointer to the end
        inout.writeInt(999);
            // Display the new length
        System.out.println("The new length is " + inout.length());
        // Retrieve the new eleventh number
        inout.seek(10 * 4); // Move the file pointer to the eleventh number
        System.out.println("The eleventh number is " + inout.readInt());
        inout.close();
    }
}
```