Lección 16: Introducción al Manejo de Archivos

Autor: José Navarro

Referencias:

Deitel, "Java How to Program: Late Objects", Cap. 6

Horstmann, "Big Java", Cap. 16.

Objetivos:

Explicar la necesidad de manejar archivos
Crear programas para leer archivos de texto
Crear programas escribir en archivos de texto

Bosquejo

16.0 Introducción

Todos los programas que hemos visto hasta el momento tienen la característica que manejan data volátil, esto es, la data desaparece tan pronto termina la ejecución del programa. Existen situaciones, sin embargo, en que esto es inadmisible. Imagine tener un programa para manejar la nómina de un compañía de 600 empleados. La información de cada empleado debe incluir, nombre, edad, seguro social, fecha de nacimiento, etc. Evidentemente no se puede tener un programa cuya data sea volátil ya que la cantidad de información que habría que entrar cada vez que se ejecuta resulta inmanejable.

Una forma de almacenar datos permanentemente es mediante el uso de archivos. Los archivos se graban en discos o cintas magnéticas de forma permanente (o hasta que se deseen borrar). En esta lección estudiaremos como hacer uso de archivos para guardar y recuperar datos.

16.1 Acceso a los archivos de entrada

Java provee diferentes formas para manejar archivos. Una de las formas más simples de hacerlo es utilizando la clase *Scanner*. La clase *Scanner* se encuentra dentro del paquete *java.util*. Esta clase nos permitirá hacer operaciones de lectura de los archivos como si estuviéramos leyendo del teclado, tal y como hemos hecho hasta ahora. Para leer de un archivo en un disco necesitamos crear objetos de la clase *FileReader*. Esta clase está dentro del paquete *java.io*. Una forma de crear un objeto para leer de un archivo de texto es la siguiente:

```
FileReader reader = new FileReader("datos.txt");
Scanner entrada = new Scanner(reader);
```

La primera línea crea un objeto de nombre *reader* (puede ser cualquier nombre). El argumento que recibe la clase ("input.txt" en este caso) indica el nombre del archivo del cual se realizarán las operaciones de lectura. El argumento puede ser un string escrito entre comillas o puede ser una variable de clase String que contenga el nombre del archivo. Si el archivo no se encuentra en el mismo fólder en donde se encuentra el programa entonces es necesario que el nombre incluya el paso completo en donde se encuentra el archivo. Por ejemplo "c:\misProgramas\archivosDeDatos\datos.txt".

La segunda línea crea un objeto de clase *Scanner* con el nombre de *entrada*. Como argumento en la construcción del objeto se envía el objeto de clase *FileReader* (*reader* en este caso). De esta forma ahora podremos leer de él como si lo estuviésemos haciendo del teclado.

Para que estas instrucciones puedan ejecutarse correctamente el archivo "datos.txt" tiene que existir previamente. Cuando realice sus primeras pruebas puede crear el archivo con un editor de texto, un IDE como jGrasp o Netbeans u otro.

16.2 Acceso a archivos de salida

Para crear y escribir archivos de salida necesitamos la clase *PrintWriter* que se encuentra en el paquete *java.io*. Para crear un archivo de salida podemos utilizar instrucciones como las siguientes:

```
PrintWriter salida;
salida = new PrintWriter("archivoSalida.txt");
```

La primera instrucción declara a *salida* (pudo ser cualquier nombre) como un objeto de clase *PrintWriter*. La segunda instrucción indica que el archivo de salida será *archivoSalida.txt*. Los nombres y forma de manejar los nombres de los archivos de salida es igual a la de los archivos de entrada. Si el archivo existe al momento de ejecutar la segunda instrucción, que es la que crea el objeto, el programa borra el contenido del mismo para luego escribir en él. Si el archivo no existe entonces el programa lo crea en el momento.

16.3 *try … catch*

Java está diseñado para que se creen archivos robustos, o sea, que puedan manejar situaciones de errores. Es por esto que el lenguaje obliga a que cuando creamos objetos para leer o escribir en archivos Java requiere que tomemos medidas para manejar situaciones que podrían generar errores. Estas situaciones incluyen pero no están limitadas a los siguientes casos: que el archivo de entrada no exista, que el dispositivo en donde se encuentra el archivo no esté disponible, que no se pueda crear el archivo de salida y otros.

Para manejar estas situaciones tenemos que crear los objetos dentro de la sección *try* y dentro de la sección *catch* escribimos el código que manejará las situaciones de error. La estructura *try ... catch ...* tiene el siguiente formato:

```
try
{
  Instrucciones que se desean ejecutar
}
catch (condicion de error)
{
  Instrucciones que se ejecutarán si ocurre un error
}
```

16.4 Programa de ejemplo

En esta sección mostraremos un programa de ejemplo en donde se lee y escribe de archivos. El programa escribirá en el archivo "output.txt" dos líneas de texto. Note que para la escritura en el archivo cuando utilizamos nuestro objeto *fileOutput* las operaciones son similares a cuando utilizamos *System.out.* Tenemos disponibles los métodos *print(), println()* y las operaciones que podemos realizar con los mismos.

Una vez que se termina de utilizar un archivo, se cierra el mismo con el método *close()*. Es importante recordar esto porque, de no cerrarse el archivo, se puede perder parcial o completamente la data almacenada en el mismo. El programa luego leerá valores enteros de un archivo de nombre "input.txt", calculará la suma de todos ellos y la imprimirá en la pantalla. El método *hasNextLine()* devuelve *true* si queda al menos una línea de texto adicional en el archivo y *false* en caso contrario. Utilizando este método como condición en un *while* podemos leer todo el contenido de un archivo.

Cada vez que se realiza una operación de lectura de un archivo de tipo texto la operación lee un String. Es por eso que veremos que el programa hace las operaciones de lectura y guarda las líneas en la variable *strLine* que es de clase String. Luego el contenido de *strLine* se convierte en un valor entero (porque para este ejemplo se presume que cada línea representa un valor entero) y se guarda en la variable *valor.*

```
import java.io.*;
import java.io.FileReader;
import java.io.PrintWriter;
import java.util.Scanner;

public class IntroManejoArchivos
{
  public static void main(String[] args) {
    FileReader reader;
    Scanner fileInput;
    PrintWriter fileOutput;
    String strLine;
    int valor,
        suma = 0;

    try
    {
      reader = new FileReader("input.txt");
      fileInput = new Scanner (reader);
      fileOutput = new PrintWriter("output.txt");

      fileOutput.println("Primera línea en el archivo");
      fileOutput.println("Segunda línea para el archivo");
      fileOutput.close();

      while (fileInput.hasNextLine())
      {
        strLine = fileInput.next();
        valor = Integer.valueOf(strLine);
        suma = suma + valor;
      }
      fileInput.close();
      System.out.println("suma = " + suma);
    }
    catch (IOException e)
    {
      System.out.println("Error manejando los archivos" + e);
    }
  }
}
```

Luego de obtener el valor numérico representado en una línea se suma el mismo a la variable *suma* para acumular la suma total. Luego de leer todos los valores del archivo se cierra el mismo con el método *close()* y luego se muestra el resultado en pantalla.

Java es un lenguaje robusto que trata de que los programadores desarrollen un código que maneje adecuadamente situaciones de error. Es por esto que requiere que las operaciones de manejo de archivos se realicen dentro de estructuras *try … catch.* En el código de nuestro ejemplo vemos que todas las operaciones dentro de los archivos se realizaron

dentro de la sección *try* de la estructura. O sea, estamos diciendo, *intenta* (*try* en inglés) ejecutar este código. Si resulta que ocurre una situación de error asociada con las operaciones de entrada y salida (como que un archivo no existe) entonces *atrapa* (*catch* en inglés) la situación y ejecuta el código que se encuentra en la sección *catch*.
La condición (IOException e) se convierte en cierta cuando ocurre un error en el manejo de las operaciones de entrada/salida. Esto hará que se ejecute el código del *catch* que en este caso es simplemente una instrucción que imprime en mensaje de error. El argumento *e* contendrá un mensaje de error que genera Java.

Un ejemplo del contenido del archivo *input.txt* se muestra a continuación.

```
12
2
43
```

Puede escribir el contenido del archivo con el mismo editor de jGrasp, con Notepad u otro editor que guarde sólo texto sin códigos especiales de formato. Si corremos el programa el resultado en pantalla será similar al siguiente:

```
suma = 57

 ----jGRASP wedge2: exit code for process is 0.
 ----jGRASP: operation complete.
```

En adición se creará el archivo *output.txt* (si ya existía entonces se borra y se escribe nuevamente sin avisar). El contenido del archivo será el siguiente:

```
Primera línea en el archivo
Segunda línea para el archivo
```

16.5 Problemas de práctica

1. Copie el programa de ejemplo de esta lección y ejecútelo con diferentes mensajes de texto y datos en el archivo de entrada para verificar que funciona correctamente. En algunos casos no haga accesible el archivo de entrada para que se cree una condición de error y se pueda apreciar la operación del *catch*. En otros casos escriba en una línea algo distinto a un número o añada una línea en blanco al final del archivo para que vea el efecto en la ejecución del programa.

2. Escriba un programa que lea de un archivo los nombres y edades de varias personas y escriba en otro archivo el nombre de la persona de mayor edad.