UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Archivos en Java.

2013
Transversal Programación Orientada a Objetos
Proyecto Curricular de Ingeniería de Sistemas

Introducción

Una tarea común en programación es leer y escribir archivos. La información almacenada en un archivo pueden ser datos byte o caracteres (texto), y pueden ser leídos en la misma forma. Java cuenta con varias clases para el acceso a archivos, cada una con su propio conjunto específico de métodos. Se puede elegir entre el acceso de flujo y el acceso aleatorio. Al utilizar el **acceso de flujo** en un archivo, se debe tratar como una secuencia de elementos que deben ser procesados uno tras otro, empezando con el primero. Para muchas tareas, esto es completamente inapropiado. El **acceso aleatorio** permite accesar una posición específica en el archivo. En estas aplicaciones (como las bases de datos) esto puede agilizar el procesamiento, pero también es más complicado de programar. En realidad es probable que se utilice una biblioteca de clase de bases de datos en vez de codificar el acceso de bajo nivel al disco.

1. Archivos de texto.

Si se desea procesar datos de un archivo existente, se debe:

1. Abrir el archivo
2. Leer o introducir los datos en las variables, un elemento a la vez
3. Cerrar el archivo cuando se termine de trabajar con él

Para transferir algunos datos de ciertas variables a un archivo, se debe:

1. Abrir el archivo
2. Extraer o escribir los elementos en la secuencia requerida
3. Cerrar el archivo cuando se termine de trabajar con él

Al leer un archivo, todo lo que puede hacerse es leer el siguiente elemento. Si, por ejemplo, quisiéramos examinar el último elemento, tendríamos que codificar un ciclo para leer cada uno de los elementos en turno, hasta llegar al elemento requerido. Para muchas tareas, es conveniente visualizar un archivo como una serie de líneas de texto, cada una compuesta por un número de caracteres y que termina con el carácter de fin de línea. Un beneficio de esta forma de trabajo es la facilidad de transferir archivos entre aplicaciones. Así se podría crear un archivo ejecutando un programa en Java y después cargarlo en un procesador de palabras, editor de texto o correo electrónico.
Los programas que utilizan archivos deben contener la instrucción:

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

"Búfer" significa que, el software lee un gran trozo de datos del dispositivo de almacenamiento externo y lo guarda en la RAM, de tal forma que invocaciones sucesivas de los métodos que necesitan leer una pequeña cantidad de datos del dispositivo de almacenamiento de archivos puedan obtener rápidamente los datos de la RAM. Por lo tanto, un búfer actúa como un amortiguador temporal entre el dispositivo de almacenamiento y el programa.

2. Utilizar canales para leer y escribir archivos.

Los canales de Archivos son quizás los canales más fáciles de entender. Las clases FileInputStream y FileOutputStream representan una canal de entrada (o salida) sobre un Archivo que reside en el sistema de Archivos nativo. Se puede crear un canal de Archivo desde un nombre de Archivo, un objeto File o un objeto FileDescriptor.

Este pequeño ejemplo utiliza los canales de Archivos para copiar el contenido de un Archivo en otro:

```
import java.io.*;

class FileStreamsTest {
    public static void main(String[] args) {
        try {
            File inputFile = new File("farrago.txt");
            File outputFile = new File("outagain.txt");

            FileInputStream fis = new
FileInputStream(inputFile);
            FileOutputStream fos = new
FileOutputStream(outputFile);
            int c;

            while ((c = fis.read()) != -1) {
               fos.write(c);
            }

            fis.close();
            fos.close();
        } catch (FileNotFoundException e) {
            System.err.println("FileStreamsTest: " + e);
        } catch (IOException e) {
            System.err.println("FileStreamsTest: " + e);
        }
    }
}
```

Aquí tiene el contenido del Archivo de entrada farrago.txt:

"*Considero más valiente al que conquista sus deseos que al que conquista a sus enemigos, ya que la victoria más dura es la victoria sobre uno mismo.*"

Aristoteles

Este programa crea un FileInputStream desde un objeto File con este código:

```
File inputFile = new File("farrago.txt");
FileInputStream fis = new FileInputStream(inputFile);
```

Observa el uso del objeto File **inputFile** en el constructor. **inputFile** representa el Archivo llamado **farrago.txt**, del sistema de archivos nativo. Este programa sólo utiliza **inputFile** para crear un FileInputStream sobre **farrago.txt**. Sin embargo, el programa podría utilizar **inputFile** para obtener información sobre **farrago.txt** como su path completo, por ejemplo.

Listar archivos usando la clase File.

```
import java.io.File;
public class ListaArchivos {
        public static void main(String[] args) {
                File f = new File("/home/alejo");
                String[] lista = f.list();
                for(int i=0;i<lista.length;i++){
                        System.out.println(lista[i]);
                }
        }
}
```

Ver las propiedades de un archivo.

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

public class PropiedadesArchivo {
        public static void main(String[] args) {
              File f = new File("/home/alejo/entrada.txt");
              if(f.exists()){
                 System.out.println("Nombre del archivo"+f.getName());
                  System.out.println("Camino"+f.getPath());
                  System.out.println("Camino absoluto
"+f.getAbsolutePath());
                  System.out.println("Se puede escribir"+f.canRead());
                  System.out.println("Se puede leer"+f.canWrite());
                  System.out.println("Tamaño  "+f.length());
                  }
        }
}
```

Uso de la clase Scanner para leer archivos.

```
// Lectura de todos los doubles de un Archivo
Scanner sc = Scanner.create(new File("miFich.txt"));
while (sc.hasNextDouble())
            double d = sc.nextDouble();
```

Bibliografía.