Archivos, Flujos y Serialización de Objetos

1.- Introducción

Los datos almacenados en variables y arreglos son temporales: se pierden cuando una variable local se sale de alcance o cuando el programa termina.
Para la retención de datos a largo plazo, incluso después de que el programa que crea los datos termina, las computadoras usan *archivos*.
Usamos archivos cotidianamente.
Las computadoras almacenan archivos en *dispositivos de almacenamiento secundario* tales como discos duros, memorias USB, DVDs, etc.
En este capítulo explicamos cómo los programas Java crean, actualizan y procesan archivos.

Comenzamos con una discusión de la arquitectura de Java para el manejo de archivos mediante programación.
A continuación, explicamos que los datos se pueden almacenar en archivos de texto y archivos binarios, y cubrimos las diferencias entre ellos.
Demostramos recuperar información sobre archivos y directorios usando clases **Paths** y **Files** e interfaces **Path** y **DirectoryStream** (todo desde el paquete **java.nio.file**), luego consideraremos los mecanismos para escribir datos en y leer datos de archivos.

2.- Archivos y Flujos

Java ve cada archivo como un *flujo secuencial de bytes* (Fig. 15.1).
Cada sistema operativo proporciona un mecanismo para determinar el final de un archivo, como un marcador de fin de archivo o un conteo de los bytes totales en el archivo que se registra en una estructura de datos administrativos mantenidos por el sistema.
Un programa Java que procesa un flujo de bytes simplemente recibe una indicación del sistema operativo cuando llega al final del flujo: el programa no necesita saber cómo la plataforma subyacente representa los archivos o los flujos.
En algunos casos, la indicación de fin de archivo se produce como una excepción.
En otros, la indicación es un valor de retorno de un método invocado en un objeto de procesamiento de flujo.

Flujos basados en bytes y flujos basados en caracteres Los flujos de archivos se pueden utilizar para ingresar y enviar datos como bytes o caracteres.
Flujos basados en bytes datos de salida y de entrada en su formato *binario*: un **char** es dos bytes, un **int** es de cuatro bytes, un **double** es de ocho bytes, etc.
Flujos basados en caracteres datos de salida y de entrada como una secuencia de caracteres, donde cada carácter es de dos bytes.
Los archivos creados con flujos basados en bytes se conocen como *archivos binarios*, mientras que los archivos creados usando flujos basados en caracteres se denominan *archivos de texto*.
Los archivos de texto pueden leerse por editores de texto, mientras que los archivos binarios son leídos por programas que comprenden el contenido específico del archivo y su ordenamiento. Un valor numérico en un archivo binario se puede usar en los cálculos, mientras que el carácter 5 es simplemente un carácter que se puede usar en una cadena de texto.

Entrada Estándar, Salida Estándar y Flujos de Error Estándar Un programa Java abre un archivo creando un objeto y asociándolo un flujo de bytes o caracteres. El constructor del objeto interactúa con el sistema operativo para *abrir* el archivo.
Cuando un programa Java comienza a ejecutarse, crea tres objetos de flujo que están asociados con dispositivos: **System.in**, **System.out** y **System.err**.
El objeto **System.in** (flujo de entrada estándar) normalmente habilita un programa para ingresar bytes desde el teclado.
El objeto **System.out** (el objeto de flujo de salida estándar) normalmente permite que un programa envíe datos de caracteres a la pantalla.
El objeto **System.err** (el objeto de flujo de error estándar) normalmente permite que un programa genere mensajes de error basados en caracteres en la pantalla.
Cada flujo puede ser *redirigido*:
Para **System.in**, esta capacidad permite al programa leer bytes de una fuente diferente. Para **System.out** y **System.err**, permite que la salida se envíe a una ubicación diferente, como un archivo en el disco.
La clase **System** proporciona los métodos **setIn**, **setOut** y **setErr** para redirigir los flujos estándar de entrada, salida y error, respectivamente.

Los paquetes **java.io** y **java.nio** Los programas Java realizan el procesamiento basado en flujo con clases e interfaces del paquete **java.io** y los subpaquetes de **java.nio**.

La entrada y salida basadas en caracteres se pueden realizar con las clases **Scanner** y **Formatter**. Se ha utilizado ampliamente la clase **Scanner** para ingresar datos desde el teclado. **Scanner** también puede leer datos de un archivo. La clase **Formatter** permite que los datos con formato se envíen a cualquier flujo basado en texto de una manera similar al método **System.out.printf**.

3.- Uso de Clases e Interfaces **NIO** para Obtener Información de Archivos y Directorios

Interfaces **Path** y **DirectoryStream** y las clases **Paths** and **Files** (todas del paquete **java.nio.file**) son útiles para recuperar información sobre archivos y directorios en el disco:

Interfaz **Path**: Los objetos de las clases que implementan esta interfaz representan la ubicación de un archivo o directorio. Los objetos **Path** no abren archivos ni proporcionan ninguna capacidad de procesamiento de archivos.

Clase **Paths**: Proporciona métodos estáticos utilizados para obtener un objeto **Path** que representa una ubicación de directorio o archivo.

Clase **Files**: Proporciona métodos estáticos para manipulaciones comunes de archivos y directorios, como copiar archivos; creación y eliminación de archivos y directorios; obtener información sobre archivos y directorios; leer el contenido de los archivos; obtención de objetos que permiten manipular el contenido de archivos y directorios.

Interfaz **DirectoryStream**: objetos de clases que implementan esta interfaz permiten a un programa iterar a través del contenido de un directorio.

Rutas absolutas v.s. relativas La ruta de un archivo o directorio especifica su ubicación en el disco.
La ruta incluye algunos o todos los directorios que conducen al archivo o directorio.
Una *ruta absoluta* contiene todos los directorios, comenzando con el directorio raíz, que conducen a un archivo o directorio específico.
Una *ruta relativa* es "relativa" a otro directorio, por ejemplo, una ruta relativa al directorio en el que la aplicación comenzó a ejecutarse.

Obtención de objetos **Path** desde URIs Un Identificador Uniforme de Recursos (URI) es una forma más general que los Localizadores Uniformes de Recursos (URL) que se utilizan para localizar sitios web.
URIs para localizar archivos varían según los sistemas operativos.
En plataformas Windows, la URI file://C:/data.txt identifica el archivo data.txt almacenado en el directorio raíz de la unidad C:
En plataformas UNIX/Linux, la URI file:/home/student/data.txt identifica el archivo data.txt almacenado en el directorio *home* del usuario *student*.

Separador de caracteres Se utiliza para separar directorios y archivos en una ruta. En una computadora con Windows, el carácter separador es una barra invertida (\). En un sistema Linux o Mac OS X, es una barra diagonal (/).

Buena práctica de programación: Al crear cadenas que representan información de ruta, use **File.separator** para obtener el carácter separador apropiado de la computadora local en lugar de usar explícitamente / o \.

Error común de programación: Usar \ como un separador de directorio en lugar de \\ en un literal de cadena es un error lógico. Un solo \ indica que el \ seguido por el siguiente carácter representa una secuencia de escape. Use \\ para insertar un \ en un literal de cadena.

Código: Fig. 15.2

4.- Archivos de Texto de Acceso Secuencial

A continuación, creamos y manipulamos archivos de acceso secuencial en los que los registros se almacenan en orden por el campo de clave de registro.
Comenzamos con los archivos de texto, lo que permite al lector crear y editar rápidamente archivos legibles para los humanos.
Discutimos cómo crear, escribir datos, leer datos y actualizar archivos de texto de acceso secuencial.

Código: Fig. 15.3 CreateTextFile
Código: Fig. 15.6 ReadTextFile
Código: Fig. 15.7 y 15.8 MenuOption and CreditInquiry

5.- Serialización de Objetos

En la sección 4, se vió cómo escribir los campos individuales de un registro en un archivo como texto y cómo leer esos campos desde un archivo. Cuando los datos se enviaban al disco, se perdía cierta información, como el tipo de cada valor. Por ejemplo, si el valor "3" se lee de un archivo, no hay forma de saber si proviene de un **int**, un **String** o un **double**. Solo tenemos datos, no información del tipo, en un disco.

A veces queremos leer un objeto o escribir un objeto en un archivo o en una conexión de red. Java proporciona la serialización de objetos para este propósito. Un objeto serializado es un objeto representado como una secuencia de bytes que incluye los datos del objeto, así como información sobre el tipo del objeto y los tipos de datos almacenados en el objeto. Después de que un objeto serializado se haya escrito en un archivo, se puede leer del archivo y deserializar, es decir, la información de tipo y los bytes que representan el objeto y sus datos se pueden usar para recrear el objeto en la memoria.

Código: Fig. 15.9 Account
Código: Fig. 15.10 CreateSequentialFile

Código: Fig. 15.11 ReadSequentialFile