Tema 10

Entrada y Salida

Sistema de Archivos

Un sistema de archivos es un medio para organizar datos que se espera que perduren después de un programa termina su ejecución, proporcionando procedimientos para almacenar, recuperar y actualizar los datos, así como organizar el espacio disponible en losdispositivos que los contienen. Un sistema de archivos organiza los datos en forma eficiente y está ajustado a las características específicas del dispositivo. Normalmente existe un fuerte acoplamiento entre el sistema operativo y el sistema de archivos.

Los sistemas de archivos se usan en dispositivos de almacenamiento de datos, como discos duros, flexibles, ópticos, memorias flash, para almacenar los archivos de computadora.

Aunque la vista de la estructura interna de un archivo como una secuencia de bytes, está relativamente estandarizada de un sistema operativo a otro, en lo que si difieren es en la metainformación asociada a ellos, como la fecha de creación del archivo, los permisos o el nombre del archivo. Esa metainformación se requiere en ciertas operaciones que manipulan a los archivos independientemente de su contenido, como crear, mover, renombrar, copiar o eliminar archivos.

Java al ser un lenguaje multiplataforma, debe lidear con esa variabilidad en los metadatos, creando un mecanismo que le oculte esas diferencias al programador.

Nombres de Archivo

Cada archivo tiene un nombre, su sintaxis depende del sistema operativo. En DOS el nombre de un archivo está formado por uno hasta 8 caracteres ASCII, seguido por una extensión de 0 a tres caracteres y separadas por un punto. Además no todos los caracteres ASCII son válidos en el nombre de un archivo. En Win32 el nombre de un archivo puede contener hasta 255 caracteres Unicode incluyendo la ruta. En Win32 también hay algunos caracteres no permitidos. De la misma manera MacOS y Unix manejan sus propias convenciones y restricciones para los nombres de archivo. Para tener mejores resultados en aplicaciones multiples plataformas considere lo siguiente al nombrar a los archivos:

- Use solamente los caracteres ASCII imprimibles (Códigos ASCII 32 a 126), puntos y guiones bajo (_).
- Evite el uso de caracteres de puntuación en especial las diagonales (/) y diagonales (invertidas).
- Nunca empiece un nombre de archivo con un punto, guiones (-) o el carácter (@).
- Evite el uso de caractere acentuados o extendidos.
- Use mayúsculas y minúsculas para facilitar la lectura. Aunque no asuma que existe diferencias entre caracteres mayúsculos y minúsculos.
- Trate de mantener la longitud de los nombres de archivo a menos de 32 caracteres.

Extensiones de los Nombres de Archivo y Tipos de Archivo

La extensión de los nombres de archivo por lo general indica el tipo de un archivo. ".java" denota un archivo texto que contiene código fuente Java. ".class" supone un archivo con código bytecode. "htm" o "html" denotan archivos que contienen código HTML. El problema es que las extensiones de archivo no garantizan el tipo de contenido. Un usuario puede cambiar la extención de un archivo o se pueden encontrar archivos de diferente contenido con la misma extensión.

Directorios

La mayoría de los sistemas operativos organizan a los archivos en una estructura jerárquica de directorios. Los directorios (llamados también carpetas) contienen cero, uno uno o más archivos o directorios. Los directorios también tienen nombres y atributos que dependiendo del sistema operativo pueden ser los mismos o no que los de los archivos.

Rutas y Separadores

Para especificar a un archivo completamente, no sólo se le da un nombre, también se debe dar el directorio en el que se encuentra el archivo. El directorio mismo puede encontrarse dentro de otro directorio, que a su vez puede estar dentro de otro directorio, hasta llegar a la raíz del sistema de archivos. La lista completa de los directorios a partir de la raíz hasta el archivo especificado, incluyendo el nombre del archivo se conoce como la **ruta absoluta** del archivo. La sintaxis de la ruta absoluta varía de sistema operativo a sistema operativo. Algunos ejemplos son los siguientes:

Sistema Operativo	Ejemplo de una Ruta Absoluta
DOS	`C: \PUBLIC\HTML\JAVAFAQ\INDEX.HTM`
Win32	`C: \public\html\javafaq\index.html`
Linux, Unix, Mac OS X	`/volumes/HD/public/html/javafaq/index.html`

En los tres casos la ruta absoluta se refiere a un archivo llamado index.html en el disco duro primario, que se ecuentra en el directorio **javafaq** que a su vez está en el directorio **html**, que a su vez está en el directorio **public**. La diferencia obvia es el carácter empleado como **separador de nombres**: '\' para DOS y Win32 o '/' para Linux, Unix o Mac OS X.

Rutas Absolutas y Relativas

Hay dos formas de referenciar a un archivo, usando una ruta absoluta o una ruta relativa. Como ya se mencióno en la sección anterior, una ruta absoluta nos da el nombre completo del archivo, empezando por el nombre de la unidad de disco o raíz del sistema de archivos. En UNIX, todas los unidades de disco que se encuentran montadas se combinan en un solo sistema de archivos virtual y su raíz es el directorio llamado '/'. Las rutas absolutas siempre empiezan con el directorio raíz '/'.

En Windows y Mac OS 9, no hay un directorio raíz. Cada unidad de disco montado es un sistema de archivos separado e independiente. En Windows, a esas unidades de disco se les asigna letras como nombre de la unidad: 'A' para el disco flexible, 'B' para el segundo disco flexible, 'C' para el disco de arranque primario, 'D', normalmente es el disco óptico, pero puedo ser otro disco o partición. Las demás letras pueden usarse para más discos, particiones o discos de red.

Una **ruta relativa** no especifica la ruta completa a un archivo, en lugar de ello específica la ruta relativa al directorio de trabajo actual. Una ruta relativa puede apuntar a un archivo dando sólo el nombre del archivo (si este se encuentra en el directorio de trabajo), otras veces puede apuntar a un archivo en un subdirectorio del directorio de trabajo dando el nombre del subdirectorio y el nombre del archivo o puede apuntar al padre del directorio de trabajo usando dos puntos (..).

Los siguientes son algunos ejemplos de rutas relativas:

Sistema Operativo	Ejemplo de una Ruta Absoluta
DOS	`HTML\JAVAFAQ\` `INDEX.HTM`
Win32	`html\javafaq\index.html` `index.html`
Linux, Unix	`html/javafaq/index.html` `index.html`
Mac OS 9	`:html:javafaq:index.html` `index.html`

Normalmente, una aplicación en ejecución identifica a un directorio como el **directorio de trabajo actual**, el directorio desde el que la aplicación fue arrancada. El directorio de trabajo queda fijo una vez que la aplicación arranca. Java no tiene forma de cambiarlo.

Clase `File`

Java trata de ocultarle al programador todas las diferencias entre los diferentes sistemas de archivos de las diferentes plataformas que soporta Java, como los atributos de archivo y los separadores de nombre y de ruta. Para ello, utiliza una abstración de los nombres de archivo y de ruta llamada **ruta abstracta**. Las instancias de la clase `java.io.File` son rutas abstractas en el sistema local, no archivos. El hecho de tener un objeto del tipo `File` no significa que el archivo exista. La clase `File` contiene métodos para obtener información acerca de los atributos de un archivo y para manipularla.

Una ruta abstracta tiene dos componentes:

1. Un prefijo que es opcional y depende del sistema, como el nombre de la unidad de disco, "`/`" para el directorio raíz en UNIX o "`\\\\`" para las rutas UNC de Microsoft Windows.
2. Una secuencia de cero o más nombres.

El primer nombre en una ruta abstracta puede ser un nombre de directorio o en el caso de las las rutas UNC de Microsoft Windows, un nombre de un huésped. Cada uno de los nombres siguientes en una ruta abstracta es un directorio; el último nombre puede ser un directorio o un archivo. Un nombre abstracto vacío no tiene prefijo y su secuencia de nombres está vacía.

La conversión entre una ruta y una ruta abstracta o viceversa es dependiente del sistema. Cuando una ruta abstracta se convierte a una ruta, cada nombre se separa del siguiente mediante una copia del carácter separador por omisión que está definido por la propiedad del sistema `file.separator` y esta disponible mediante los atributos públicos estáticos `separator` y `separatorChar` de esta clase. caundo una ruta se convierte a una ruta abstracta, los nombres pueden separarse por el carácter separador por omisión o por cualquier otro carácter separador soportado por el sistema.

Cada objeto de tipo `File` contiene un campo de tipo String llamado `path` que contiene ya sea un una ruta relativa o absoluta al archivo incluyendo el nombre del archivo o directorio. Muchos de los métodos de esta clase trabajan solamente inspeccionando esta cadena. No necesariamente inspeccionan alguna parte del sistema de archivos.

En la figura 10.1 se muestra el diagrama de la clase `File` y en las tablas 10.1 y 10.2 los atributos y algunos de sus métodos.

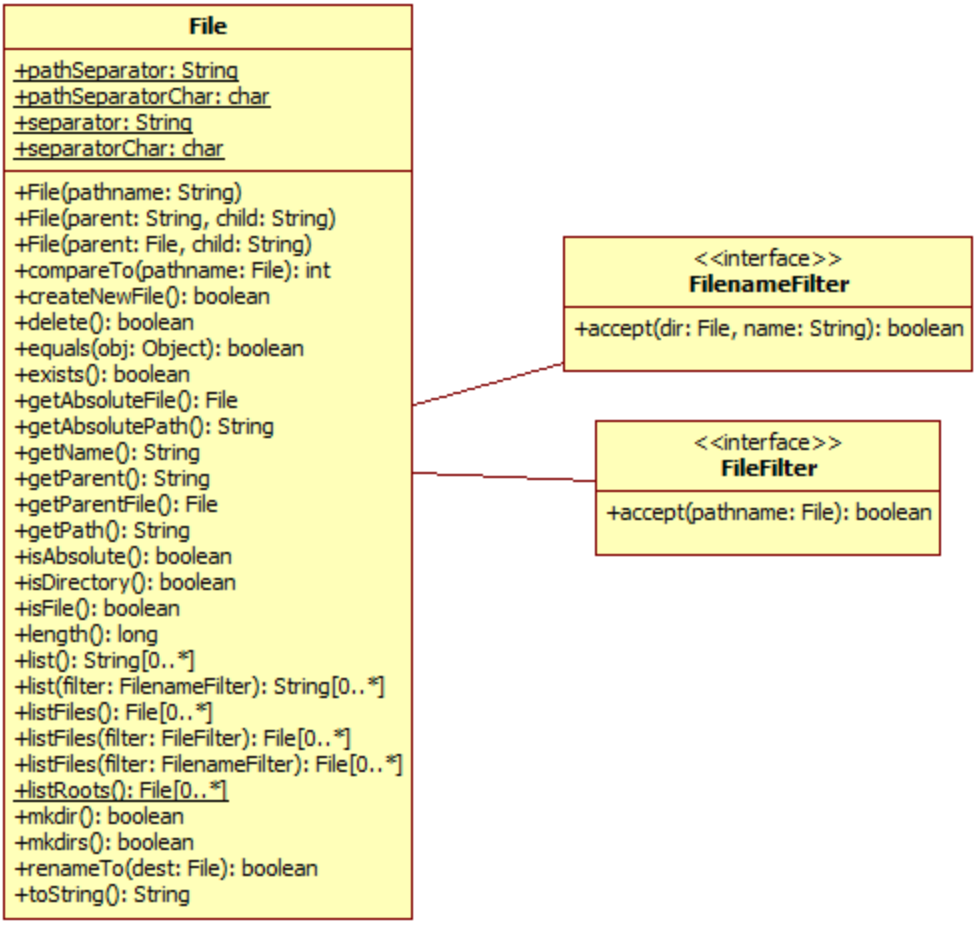


Figura 10.1 Diagrama de Clases de la Clase `File`.

Tabla 10.1. Atributos de la Clase `File`

`public static final String` **`pathSeparator`** El caracter separador de rutas, dependiente del sistema, representado como una cadena. La cadena contiene un solo carácter, el caracter separador de rutas.
`public static final char` **`pathSeparatorChar`** El caracter separador de rutas, dependiente del sistema. Contiene el primer carácter de la propiedad `pathSeparator`. Este carácter se utiliza para separar las rutas en una secuencia de archivos dada como una lista de rutas. En los sitemas UNIX el carácter es '`:`' en Windows es '`;`',
`public static final String` **`separator`** El caracter separador de nombres, dependiente del sistema, representado como una cadena. La cadena contiene un solo carácter, el caracter separador de nombres.
`public static final char` **`separatorChar`** El caracter separador de nombres, dependiente del sistema. Contiene el primer carácter de la propiedad `separator`. Este carácter se utiliza para separar los nombres de una ruta. En los sitemas UNIX el carácter es '`/`' en Windows es '`\\`',

Tabla 10.2. Métodos de la Clase `File`

`public` **`File`**`(String pathname)` Crea una nueva instancia de la clase **`File`** convirtiendo la ruta del parámetro a una ruta abstracta. La ruta del parámetro está en un formato entendible por el sistema operativo huésped. Si la ruta del parámetro es una cadena vacía, el resultado es una ruta abstracta nula. Lanza: `NullPointerException` - Si el parámetro `pathname` es `null`.
`public` **`File`**`(String parent, String child)` Crea una nueva instancia de la clase **`File`** a partir de las rutas dadas por los parámetros `parent` y `child`. Si el parámetro `parent` es null, la nueva instancia de **`File`** se crea como si se invocara al constructor con un solo argumento con el parámetro `child`. De otro modo, se toma la ruta dada por el parámetro `parent` como si fuera un directorio y la ruta dada por su parámetro `child` como si fuera un directorio o un archivo. Si la ruta dada por `child` es absoluta entonces se convierte a una ruta relativa en una forma dependiente del sistema. Si el parámetro `parent` es una cadena vacía, la nueva instancia de `File` se crea convirtiendo el parámetro `child` a una ruta abstracta y el resultado se resuelve contra el directorio por omisión dependiente del sistema. De otro modo cada ruta se convierte a una ruta abstracta y la ruta abstracta del parámetro `child` se resuelve contra la ruta del parámetro `parent`. Lanza: `NullPointerException` - Si el parámetro `child` es `null`.
`public` **`File`**`(File parent, String child)` Crea una nueva instancia de la clase **`File`** a partir de la ruta abstracta dada por el parámetro `parent` y la ruta dada por el parámetro `child`. Si el parámetro `parent` es null, la nueva instancia de **`File`** se crea como si se invocara al constructor con un solo argumento con el parámetro `child`. De otro modo, se toma la ruta abstracta dada por el parámetro `parent` como si fuera un directorio y la ruta dada por su parámetro `child` como si fuera un directorio o un archivo. Si la ruta dada por `child` es absoluta entonces se convierte a una ruta relativa en una forma dependiente del sistema. Si el parámetro `parent` es una ruta abstracta vacía, la nueva instancia de `File` se crea convirtiendo el parámetro `child` a una ruta abstracta y el resultado se resuelve contra el directorio por omisión dependiente del sistema. De otro modo cada ruta se convierte a una ruta abstracta y la ruta abstracta del parámetro `child` se resuelve contra la ruta del parámetro `parent`. Lanza: `NullPointerException` - Si el parámetro `child` es `null`.

Tabla 10.2. Métodos de la Clase `File`. Cont.

`public int compareTo(File pathname)` Compara dos rutas abstractas lexicográficamente. El ordenamiento definido por este método es dependiente del sistema. En Unix, hay distinción entre mayúsculas y minúsculas, en Windows no. Regresa cero si la ruta abstracta del parámetro es igual a esta ruta abstracta, un valor negativo si esta ruta abstracta es lexicográficamente menor que la ruta abstracta del parámetro o valor positivo si esta ruta abstracta es lexicográficamente mayor que la ruta abstracta del parámetro.
`public boolean createNewFile() throws IOException` Crea un archivo vacío nuevo en forma atómica, con el nombre de esta ruta abstracta si y sólo sí no existe ya un archivo con este nombre. Regresa `true` si se pudo crear el archivo, `false` si el archivo ya existe. Lanza: `IOException` - Si ocurre un error de E/S. `SecurityException` - Si existe un administrador de seguridad y este niega el acceso a escribir al archivo.
`public boolean` **`delete`**`()` Borra el archivo o directorio dado por esta ruta abstracta. Si esta ruta abstracta denota un directorio, entonces el directorio debe estar vacío para poder borrarlo. Regresa `true` si se pudo borrar el archivo, `false` en caso contrario. Lanza: `SecurityException` - Si existe un administrador de seguridad y este niega el acceso a borrar el archivo.
`public boolean` **`equals`**`(Object obj)` Compara esta ruta abstracta por igualdad con el objeto del parámetro. Regresa `true` si y sólo sí el parámetro no es `null` y esta ruta abstracta es el mismo archivo o directorio que esta ruta abstracta. Que dos rutas abstractas sean iguales es dependiente del sistema. En Unix, hay distinción entre mayúsculas y minúsculas, en Windows no.
`public boolean` **`exists`**`()` Regresa true si y sólo sí el archivo o directorio dado por esta ruta abstracta existe. Lanza: `SecurityException` - Si existe un administrador de seguridad y este niega el acceso a leer el archivo.
`public File` **`getAbsoluteFile`**`()` Regresa la ruta abstracta absoluta de esta ruta abstracta. Lanza: `SecurityException` - Si no se puede acceder a un valor de una propiedad del sistema.

Tabla 10.2. Métodos de la Clase `File`. Cont.

`public String` **`getAbsolutePath`**`()` Regresa una cadena con la ruta absoluta de esta ruta abstracta. Si esta ruta abstracta es ya una ruta absoluta, entonces simplemente se regresa su ruta. Si esta ruta abstracta es una ruta vacía entonces se regresa la ruta del directorio actual que está dada por la propiedad `user.dir`. De otra forma esta ruta es resuelta dependiendo del sistema. En los sistemas UNIX una ruta relativa se convierte en absoluta resolviéndola con respecto al directorio actual del usuario. En Windows se resuelve con respecto al directorio actual de la unidad de disco dada por la ruta, si la hay, o contra el directorio actual del usuario. Lanza: `SecurityException` - Si no se puede acceder a un valor de una propiedad del sistema.
`public String` **`getName`**`()` Regresa el nombre del archivo o directorio dado por esta ruta abstracta. Es el último nombre en la secuencia del nombre de la ruta. Si la secuencia de la ruta es vacía se regresa una cadena vacía.
`public String` **`getParent`**`()` Regresa el directorio padre de esta ruta abstracta o `null` si la ruta no nombra un directorio padre. El directorio padre de una ruta abstracta consiste de prefijo de la ruta, si existe, seguido de cada uno de los nombres de la ruta menos el último. Si la secuencia de la ruta está vacía entonces la ruta no nombra un directorio padre.
`public File` **`getParentFile`**`()` Regresa la ruta abstracta del directorio padre de esta ruta abstracta o `null` si la ruta no nombra un directorio padre. El directorio padre de una ruta abstracta consiste de prefijo de la ruta, si existe, seguido de cada uno de los nombres de la ruta menos el último. Si la secuencia de la ruta está vacía entonces la ruta no nombra un directorio padre.
`public String` **`getPath`**`()` Convierte esta ruta abstracta a una cadena con la ruta. La cadena resultante usa el carácter separador de nombres por ausencia para separar los nombres de la ruta.
`public boolean` **`isAbsolute`**`()` Regresa `true` si esta ruta abstracta es absoluta, `false` en caso contrario. La definición de ruta absoluta es dependiente del sistema. En UNIX la ruta es absoluta si su prefijo es `"/"`. En Windows si su prefijo es un especificador de la unidad de disco seguida por `"\\"` o si su prefijo es `"\\\\"`.
`public boolean` **`isDirectory`**`()` Regresa `true` si el archivo denotado por esta ruta abstracta existe y es un directorio, `false` en caso contrario. Lanza: `SecurityException` - Si existe un administrador de seguridad y este niega el acceso a leer el archivo.

Tabla 10.2. Métodos de la Clase `File`. Cont.

`public boolean` **`isFile`**`()` Regresa `true` si el archivo denotado por esta ruta abstracta existe y es un archivo normal, `false` en caso contrario. Un archivo es normal si no es un directorio Lanza: `SecurityException` - Si existe un administrador de seguridad y este niega el acceso a leer el archivo.
`public long` **`length`**`()` Regresa la longitud del archivo, denotado por esta ruta abstracta, en bytes. Si la ruta denota un directorio el valor regresado no está especificado. Si el archivo no existe, el método regresa `0L`.
`public String[]` **`list`**`()` Regresa un arreglo de cadenas con los nombres de los archivos y directorios dentro del directorio dado por esta ruta abstracta. Si el directorio está vacío, el arreglo estará vacío. Si esta ruta abstracta no es un directorio o si ocurre un error de E/S, el método regresa `null`. Cada cadena es sólo el nombre del archivo o directorio, no la ruta completa. No hay garantía que el arreglo de canciones con los nombres de los archivos y directorios aparezca en un orden determinado. Lanza: `SecurityException` - Si existe un administrador de seguridad y este niega el acceso a leer el directorio.
`public String[]` **`list`**`(FilenameFilter filter)` Regresa un arreglo de cadenas con los nombres de los archivos y directorios dentro del directorio dado por esta ruta abstracta y que satisfacen el filtro dado por el parámetro. Si el directorio está vacío o ninguno de los nombres satisface al filtro, el arreglo estará vacío. Si esta ruta abstracta no es un directorio o si ocurre un error de E/S, el método regresa `null`. Si el filtro del parámetro es `null`, se aceptan todos los nombres. De otra manera, un nombre satisface al filtro si y sólo el método `FilenameFilter.accept()`del filtro regresa `true` cuando se le invoca con el nombre del archivo o directorio. Cada cadena es sólo el nombre del archivo o directorio, no la ruta completa. No hay garantía que el arreglo de canciones con los nombres de los archivos y directorios aparezca en un orden determinado. Lanza: `SecurityException` - Si existe un administrador de seguridad y este niega el acceso a leer el directorio.
`public File[]` **`listFiles`**`()` Regresa un arreglo de rutas abstractas de los archivos y directorios dentro del directorio dado por esta ruta abstracta. Si el directorio está vacío, el arreglo estará vacío. Si esta ruta abstracta no es un directorio o si ocurre un error de E/S, el método regresa `null`. Si esta ruta abstracta, es absoluta, cada ruta abstracta del arreglo es absoluta. Si esta ruta abstracta es relativa, cada ruta abstracta del arreglo es relativa al mismo directorio. No hay garantía que el arreglo de canciones con los nombres de los archivos y directorios aparezca en un orden determinado. Lanza: `SecurityException` - Si existe un administrador de seguridad y este niega el acceso a leer el directorio.

Tabla 10.2. Métodos de la Clase `File`. Cont.

`public File[] listFiles(FilenameFilter filter)` Regresa un arreglo de rutas abstractas de los archivos y directorios dentro del directorio dado por esta ruta abstracta y que satisfacen el filtro dado por el parámetro. Si el directorio está vacío o ninguno de los nombres satisface al filtro, el arreglo estará vacío. Si esta ruta abstracta no es un directorio o si ocurre un error de E/S, el método regresa `null`. Si el filtro del parámetro es `null`, se aceptan todos los nombres. De otra manera, un nombre satisface al filtro si y sólo el método `FilenameFilter.accept()`del filtro regresa `true` cuando se le invoca con el nombre del archivo o directorio. Si esta ruta abstracta, es absoluta, cada ruta abstracta del arreglo es absoluta. Si esta ruta abstracta es relativa, cada ruta abstracta del arreglo es relativa al mismo directorio. No hay garantía que el arreglo de canciones con los nombres de los archivos y directorios aparezca en un orden determinado. Lanza: `SecurityException` - Si existe un administrador de seguridad y este niega el acceso a leer el directorio.
`public File[] listFiles(FileFilter filter)` Regresa un arreglo de rutas abstractas de los archivos y directorios dentro del directorio dado por esta ruta abstracta y que satisfacen el filtro dado por el parámetro. Si el directorio está vacío o ninguno de los nombres satisface al filtro, el arreglo estará vacío. Si esta ruta abstracta no es un directorio o si ocurre un error de E/S, el método regresa `null`. Si el filtro del parámetro es `null`, se aceptan todos los nombres. De otra manera, un nombre satisface al filtro si y sólo el método `FileFilter.accept()`del filtro regresa `true` cuando se le invoca con el nombre del archivo o directorio. Si esta ruta abstracta, es absoluta, cada ruta abstracta del arreglo es absoluta. Si esta ruta abstracta es relativa, cada ruta abstracta del arreglo es relativa al mismo directorio. No hay garantía que el arreglo de canciones con los nombres de los archivos y directorios aparezca en un orden determinado. Lanza: `SecurityException` - Si existe un administrador de seguridad y este niega el acceso a leer el directorio.
`public static File[] listRoots()` Lista todos los directorios raíz de los sistemas de archivos. Cada sistema de archivos tiene un directorio raíz desde el cual todos los archivos en ese directorio pueden alcanzarse. En Windows cada unidad de disco tiene un directorio raíz. En UNIX hay un solo directorio raíz. Este método regresa un arreglo de rutas abstractas, una para cada uno de los directorios del sistema de archivos. Si no sepuede determinar los directorios raíz del sistema de archivos, el método regresa `null`. Si no hay directorios raíz en el sistema de archivos, el arreglo estará vacío. Si existe un administrador de seguridad y este niega el acceso a leer un directorio raíz en particular, este directorio raíz no aparecerá en el arreglo.

Tabla 10.2. Métodos de la Clase `File`. Cont.

`public boolean` **`mkdir`**`()` Crea un directorio con el nombre de esta ruta abstracta. Regresa `true` si se pudo crear el directorio, `false` en caso contrario. Lanza: `SecurityException` - Si existe un administrador de seguridad y este niega el crear el directorio.
`public boolean` **`mkdirs`**`()` Crea un directorio con el nombre de esta ruta abstracta, incluyendo cualquier directorio padre necesario pero no existente. Aún cuando esta operación falle en crear el directorio, alguno de los directorios pudo haber sido creado. Regresa `true` si se pudo crear el directorio, junto con sus directorios padres, `false` en caso contrario. Lanza: `SecurityException` - Si existe un administrador de seguridad y este niega el crear el directorio y sus directorios padres.
`public boolean` **`renameTo`**`(File dest)` Renombra el archivo o directorio dado por esta ruta abstracta. Mucho del comportamiento de este método es dependiente de la plataforma. La operación de renombrar puede permitir mover al archivo de un sistema de archivos a otro, puede ser una operación atómica o no y puede tener éxito o no si el archivo destino ya existe. El valor de retorno debe verificarse para saber si la operación tuvo éxito o no. Regresa `true` si se pudo renombrar el archivo, `false` en caso contrario. Lanza: `SecurityException` - Si existe un administrador de seguridad y este niega el acceso a leer el archivo anterior o nuevo. `NullPointerException` – Si el parámetro `dest` es `null`.
`public String` **`toString`**`()` Regresa una cadena con la ruta de esta ruta abstracta. Es la misma cadena regresada por el método **`getPath`**`()`.

Ejemplo sobre los métodos de la Clase File

El siguiente código ilustra el uso de algunos de los métodos de la clase `File`.

PruebaFile.java

```
/*
 * PruebaFile.java
 */

package pruebas;

import java.io.File;

/**
 * Este codigo ilustra el uso de algunos de los metodos de la clase io.File
```

```
 *
 * @author mdomitsu
 */
public class PruebaFile {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        String userDir = System.getProperty("user.dir");
        String separador = System.getProperty("file.separator");
        System.out.println("Directorio del usuario: " + userDir);
        System.out.println("Separador: " + separador);

        File file1 = new File(userDir);
        System.out.println("Ruta abstracta del directorio del usuario: "
                           + file1);
        File file2 = new File("src\\pruebas\\PruebaFile.java");
        System.out.println("Caracteristicas del archivo:
src\\pruebas\\PruebaFile.java: ");
        System.out.println("Nombre: " + file2.getName());
        System.out.println(file2.getName() + (file2.exists()? " ": " no ")
                           + "existe");
        System.out.println("Tamaño " + file2.length() + " bytes");
        System.out.println("Ruta: " + file2.getPath());
        System.out.println("Ruta abstracta: " + file2);
        System.out.println("Ruta absoluta: " + file2.getAbsolutePath());
        System.out.println("Ruta abstracta absoluta: "
                           + file2.getAbsoluteFile());
        System.out.println(file2.getName() + (file2.isFile()? " ": " no ")
                           + "es un archivo");
        System.out.println(file2.getName()
                           + (file2.isDirectory()? " ": " no ")
                           + "es un directorio");
        System.out.println("\n");

        System.out.println("Caracteristicas del padre de:
src\\pruebas\\PruebaFile.java: ");
        System.out.println("Ruta: " + file2.getParent());
        System.out.println("Ruta abstracta: " + file2.getParentFile());
        System.out.println("Ruta absoluta: "
                           + file2.getParentFile().getAbsolutePath());
        System.out.println("Ruta abstracta absoluta: "
                           + file2.getParentFile().getAbsoluteFile());
        System.out.println(file2.getParent()
                           + (file2.getParentFile().isFile()? " ": " no ")
                           + "es un archivo");
        System.out.println(file2.getParent()
                           + (file2.getParentFile().isDirectory()? " ": " no
") + "es un directorio");

        File dir[] = file1.listFiles();
        System.out.println("Archivos del directorio del usuario: ");
        for(File file: dir) {
            System.out.println(file);
        }
```

```
        File raices[] = File.listRoots();
        System.out.println("Raices: ");
        for(File file: raices) {
            System.out.println(file);
        }
    }
}
```

La salida de ese programa es:

```
Directorio del usuario:
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io
Separador: \
Ruta abstracta del directorio del usuario:
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io
Caracteristicas del archivo: src\pruebas\PruebaFile.java:
Nombre: PruebaFile.java
PruebaFile.java existe
Tamaño 2603 bytes
Ruta: src\pruebas\PruebaFile.java
Ruta abstracta: src\pruebas\PruebaFile.java
Ruta absoluta:
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\src\pruebas\
PruebaFile.java
Ruta abstracta absoluta:
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\src\pruebas\
PruebaFile.java
PruebaFile.java es un archivo
PruebaFile.java no es un directorio

Caracteristicas del padre de: src\pruebas\PruebaFile.java:
Ruta: src\pruebas
Ruta abstracta: src\pruebas
Ruta absoluta:
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\src\pruebas
Ruta abstracta absoluta:
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\src\pruebas
src\pruebas no es un archivo
src\pruebas es un directorio
Archivos del directorio del usuario:
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\build
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\build.xml
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\manifest.mf
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\nbproject
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\src
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\test
Raices:
C:\
D:\
E:\
F:\
```

Filtros de Nombres de Archivo

Dos de los métodos de la clase File permiten obtener una lista de los archivos de un directorio que cumplen con cierto criterio:

- `public String[] **list**(FilenameFilter filter)`
- `public File[] **listFiles**(FilenameFilter filter)`
- `public File[] **listFiles**(FileFilter filter)`

Que regresan los nombres de los archivos (y directorios) y las rutas abstractas de los archivos (y directories) de la ruta abstracta de la clase y que satisfacen el filtro de su parámetro. Los filtros son instancias de clases que implementan ya sea la interfaz `FilenameFilter` o la interfaz `FileFilter`.

Interfaz `FilenameFilter`

Las clases que implementan esta interfaz se usan para filtrar nombres de archivos. Esta interfaz sólo declara el método `accept()` mostrado en la tabla 10.3.

Tabla 10.3. Método `accept()` de la Interfaz `FilenameFilter`

`boolean **accept**(File dir, String name)` Regresa `true` si el archivo dado por el parámetro `name` y que se encuentra en el directorio dado por el parámetro `dir`, satisface al filtro, `false` en caso contrario.

Interfaz `FileFilter`

Las clases que implementan esta interfaz se usan para filtrar rutas abstractas. Esta interfaz sólo declara el método `accept()` mostrado en la tabla 10.4.

Tabla 10.4. Método `accept()` de la Interfaz `FileFilter`

`boolean **accept**(File pathname)` Regresa `true` si la ruta abstracta dada por el parámetro `pathname` satisface al filtro, `false` en caso contrario.

Ejemplo sobre Filtros para los Directorios

El siguiente código es un filtro que acepta sólo rutas abstractas que son archivos.

FiltraArchivos.java

```
/*
 * FiltraArchivos.java
 */

package filtros;

import java.io.File;
import java.io.FileFilter;

/**
 * Este filtro sólo acepta archivos
 * @author usuario
 */
```

```
public class FiltraArchivos implements FileFilter {

    /**
     * Prueba si la ruta abstracta es un archivo
     * @param pathname ruta abstracta a probar
     * @return true si la ruta abstracta es un archivo,
     * false en caso contrario
     */
    public boolean accept(File pathname) {
        return pathname.isFile();
    }
}
```

El siguiente código es un filtro que acepta sólo rutas abstractas que son directorios.

FiltraDirs.java

```
/*
 * FiltraDirs.java
 */

package filtros;

import java.io.File;
import java.io.FileFilter;

/**
 * Este filtro sólo acepta directorios
 * @author usuario
 */
public class FiltraDirs implements FileFilter {

    /**
     * Prueba si la ruta abstracta es un directorio
     * @param pathname ruta abstracta a probar
     * @return true si la ruta abstracta es un directorio,
     * false en caso contrario
     */
    public boolean accept(File pathname) {
        return pathname.isDirectory();
    }
}
```

El siguiente código utiliza los filtros anteriores para listar los archivos y directorios, sólo los archivos y sólo los directorios.

PruebaFiltro.java

```
/*
 * PruebaFiltro.java
 */

package pruebas;

import filtros.FiltraArchivos;
import filtros.FiltraDirs;
```

```
import java.io.File;

/**
 * Este codigo ilustra el uso de los filtros para listar directorios
 *
 * @author usuario
 */
public class PruebaFiltro {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        File dir[];

        String userDir = System.getProperty("user.dir");
        String separador = System.getProperty("file.separator");
        System.out.println("Directorio del usuario: " + userDir);
        File file1 = new File(userDir);

        dir = file1.listFiles();
        System.out.println("Archivos y directorios del directorio del
usuario: ");
        for(File file: dir) {
            System.out.println(file);
        }

        dir = file1.listFiles(new FiltraArchivos());
        System.out.println("\nArchivos del directorio del usuario: ");
        for(File file: dir) {
            System.out.println(file);
        }

        dir = file1.listFiles(new FiltraDirs());
        System.out.println("\nDirectorios del directorio del usuario: ");
        for(File file: dir) {
            System.out.println(file);
        }
    }
}
```

La salida del programa anterior es:

```
Directorio del usuario:
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io
Archivos y directorios del directorio del usuario:
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\build
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\build.xml
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\manifest.mf
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\nbproject
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\src
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\test

Archivos del directorio del usuario:
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\build.xml
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\manifest.mf
```

```
Directorios del directorio del usuario:
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\build
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\nbproject
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\src
C:\Users\usuario\Documents\NetBeansProjects_Notas_Cursos\io\test
```

La Clase `JFileChooser`

La clase `JFileChooser` del paquete `swing` de la API de Java es un componente que nos permite seleccionar uno o más archivos. No es un cuadro de diálogo por lo que debe agregarse a una ventana, cuadro de diálogo u otro contenedor. Sin embargo, la clase `JFileChooser` tiene varios métodos que crean un cuadro de diálogo con una instancia de esta clase y despliegan el cuadro de diálogo.

Para usar `JFileChooser` para pedirle al usuario que seleccione un archivo se siguen los siguientes pasos:

1. Construye una instancia de la clase `JFileChooser`.
2. Establece si se desea el o los filtros de archivo deseados.
3. Despliega la instancia de la clase `JFileChooser`.
4. Obtén los archivos seleccionados por el usuario.

Para ello se usan algunos de los métodos de la clase.

La figura 10.2 muestra el diagrama de clases de la clase `JFileChooser` y las tablas10.5 y 10.6 los atributos y algunos de sus métodos.

Tabla 10.5. Atributos de la Clase `JFileChooser`

`public static final int` **`APPROVE_OPTION`** Valor regresado por la instancia de la clase `JFileChooser` si el usuario selecciona la opción aceptar.
`public static final int` **`CANCEL_OPTION`** Valor regresado por la instancia de la clase `JFileChooser` si el usuario selecciona la opción cancelar.
`public static final int` **`CUSTOM_DIALOG`** Indica que la instancia de la clase `JFileChooser` soporta una operación definida por el programador.
`public static final int` **`DIRECTORIES_ONLY`** Le indica a la instancia de la clase `JFileChooser` que solo muestre directorios.
`public static final int` **`ERROR_OPTION`** Valor regresado por la instancia de la clase `JFileChooser` si ocurrió un error.
`public static final int` **`FILES_AND_DIRECTORIES`** Le indica a la instancia de la clase `JFileChooser` que sólo muestre tanto archivos como directorios.
`public static final int` **`FILES_ONLY`** Le indica a la instancia de la clase `JFileChooser` que solo muestre archivos.

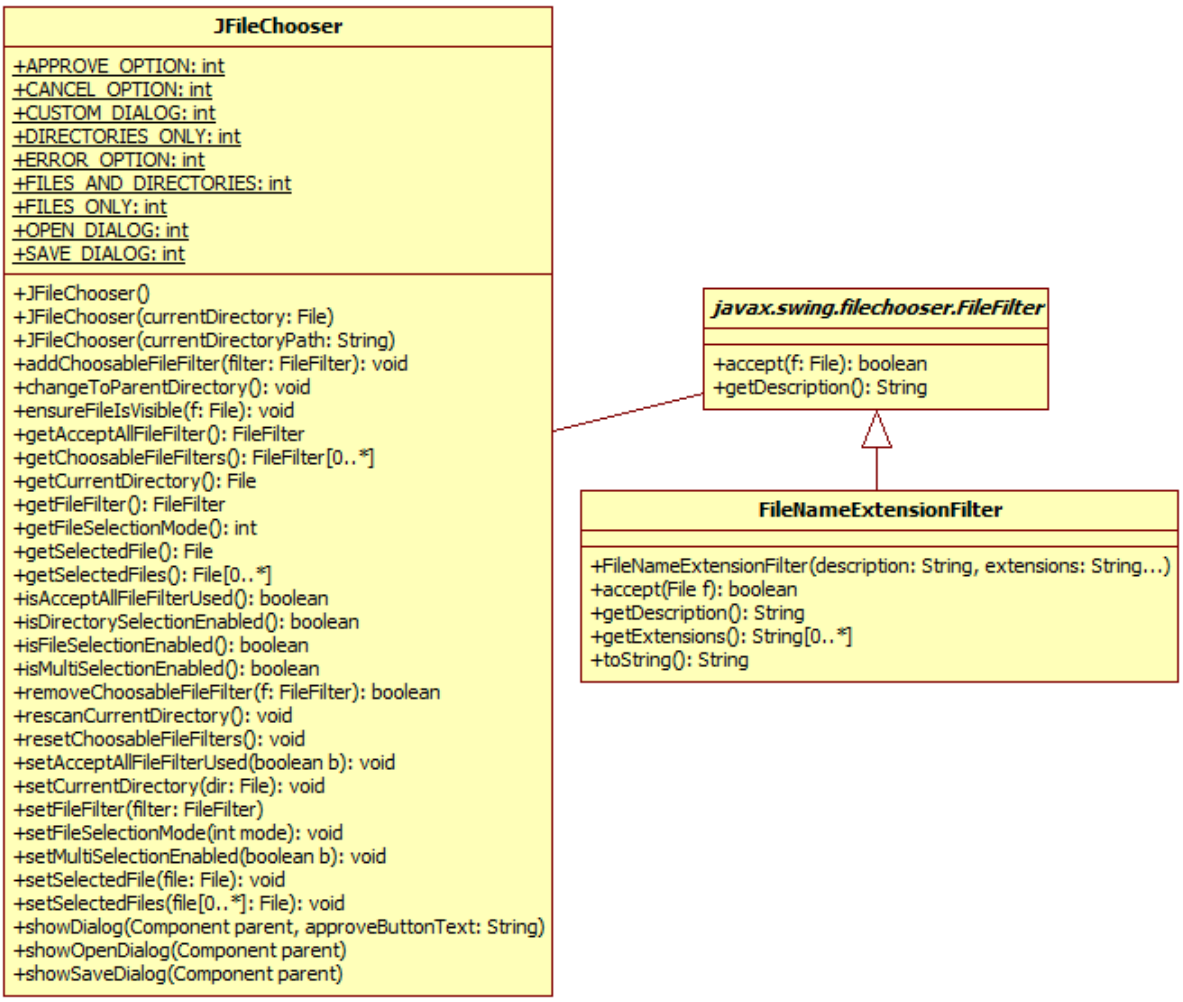


Figura 10.2 Diagrama de Clases de la Clase `JFileChooser`.

Tabla 10.5. Atributos de la Clase `JFileChooser`. Cont.

`public static final int` **`OPEN_DIALOG`** Indica que la instancia de la clase `JFileChooser` soporta una operación de abrir archivo.
`public static final int` **`SAVE_DIALOG`** Indica que la instancia de la clase `JFileChooser` soporta una operación de guardar archivo.

Tabla 10.6. Métodos de la Clase `JFileChooser`

`public` **`JFileChooser`**`()` Crea una nueva instancia de la clase `JFileChooser` apuntando al directorio del usuario por ausencia. Este directorio es dependiente del sistema. En Windows es por lo general la carpeta “Mis Documentos” y el directorio hogar del usuario en UNIX.

Tabla 10.6. Métodos de la Clase `JFileChooser`. Cont.

`public` **`JFileChooser`**`(File currentDirectory)` Crea una nueva instancia de la clase `JFileChooser` apuntando al directorio dado por la ruta abstracta del parámetro. Si el parámetro es `null`, se usa el directorio del usuario por ausencia. Este directorio es dependiente del sistema. En Windows es por lo general la carpeta “Mis Documentos” y el directorio hogar del usuario en UNIX.
`public` **`JFileChooser`**`(String currentDirectoryPath)` Crea una nueva instancia de la clase `JFileChooser` apuntando al directorio dado por la ruta del parámetro. Si el parámetro es `null`, se usa el directorio del usuario por ausencia. Este directorio es dependiente del sistema. En Windows es por lo general la carpeta “Mis Documentos” y el directorio hogar del usuario en UNIX.
`public void addChoosableFileFilter``(FileFilter filter)` Agrega un filtro a la lista de filtros seleccionables por el usuario.
`public void changeToParentDirectory()` Cambia el directorio para que sea el directorio padre del directorio actual.
`public void` **`ensureFileIsVisible`**`(File f)` Se asegura que el archivo del parámetro sea visible y no estéoculto
`public FileFilter` **`getAcceptAllFileFilter`**`()` Regresa el filtro que acepta todos los archivos. En Windows sería: `All Files (*.*)`.
`public FileFilter[]` **`getChoosableFileFilters`**`()` Regresa la lista de filtros seleccionables por el usuario.
`public File` **`getCurrentDirectory`**`()` Regresa el directorio actual.
`public FileFilter` **`getFileFilter`**`()` Regresa el filtro seleccionado actual
`public int` **`getFileSelectionMode`**`()` Regresa el tipo de archivos a ser desplegado. Puede ser uno de los siguientes: • `JFileChooser.FILES_ONLY`: Despliegua sólo archivos. • `JFileChooser.DIRECTORIES_ONLY`: Despliegua sólo directorios. • `JFileChooser.FILES_AND_DIRECTORIES`: Despliega ambos. El valor por omisión es `JFilesChooser.FILES_ONLY`.
`public File` **`getSelectedFile`**`()` Regresa la ruta abstracta del archivo seleccionado.
`public File` **`getSelectedFiles`**`()` Regresa una lista con las rutas abstractas de los archivos seleccionados si está habilitada la selección múltiple.
`public boolean` **`isAcceptAllFileFilterUsed`**`()` Regresa `true` si se esta usando el filtro que acepta todos los archivos.

Tabla 10.6. Métodos de la Clase `JFileChooser`. Cont.

`public boolean` **`isDirectorySelectionEnabled`**`()` Regresa `true` si el modo de selección permite que se muestren los directorios.
`public boolean` **`isFileSelectionEnabled`**`()` Regresa `true` si el modo de selección permite que se muestren los archivos.
`public boolean` **`isMultiSelectionEnabled`**`(` Regresa `true` si se pueden seleccionar múltiples archivos.
`public boolean` **`removeChoosableFileFilter`**`(FileFilter f)` Elimina un filtro de la lista de filtros seleccionables por el usuario.
`public void` **`rescanCurrentDirectory`**`()` Actualiza la lista de los archives desplegados
`public void` **`resetChoosableFileFilters`**`()` Restablece la lista de filtros seleccionables por el usuario a su estado original. Por lo general elimina todos los filtros agregados dejando sólo el filtro que acepta todos los archivos.
`public void` **`setAcceptAllFileFilterUsed`**`(boolean b)` Establece si el filtro que acepta todos los archivos va a estar en la lista de filtros seleccionables por el usuario. Si el valor del parámetro es `false`, el filtro se elimina de la lista de filtros. Si el valor del parámetro es `true`, el filtro se vuelve el filtro activo.
`public void` **`setCurrentDirectory`**`(File dir)` Establece el directorio actual al directorio dado por la ruta abstracta del parámetro. Si el parámetro es `null`, se usa el directorio del usuario por ausencia. Este directorio es dependiente del sistema. En Windows es por lo general la carpeta “Mis Documentos” y el directorio hogar del usuario en UNIX. Si el valor del parámetro no es un directorio, se usará el directorio padre.
`public void` **`setFileFilter`**`(FileFilter filter)` Establece el filtro actual al filtro del del parámetro.
`public void` **`setFileSelectionMode`**`(int mode)` Establece el modo de selección de la instancia de `JFilesChooser` al valor del parámetro. Los valores permitidos son: • `JFileChooser.FILES_ONLY`: Despliegua sólo archivos. • `JFileChooser.DIRECTORIES_ONLY`: Despliegua sólo directorios. • `JFileChooser.FILES_AND_DIRECTORIES`: Despliega ambos. El valor por omisión es `JFileChooser.FILES_ONLY`. Lanza: `IllegalArgumentException` – Si el valor del parámetro es un valor ilegal.
`public void` **`setMultiSelectionEnabled`**`(boolean b)` Si el valor del parámetro es `true` se permite seleccionar múltiples archivos.

Tabla 10.6. Métodos de la Clase `JFileChooser`. Cont.

`public void` **`setSelectedFile`**`(File file)` Establece el archivo seleccionado al archivo dado por la ruta abstracta del parámetro. Si el directorio padre del archivo no es el directorio actual, se cambia eldirectorio actual al directorio padre delarchivo.
`public void` **`setSelectedFiles`**`(File[] selectedFiles)` Establece los archivos seleccionados a los archivos dados por las rutas abstractas del parámetro si se permite seleccionar múltiples archivos.
`public int` **`showDialog`**`(Component parent, String approveButtonText) throws HeadlessException` Despliega un cuadro de diálogo con el texto dado por el parámetro `approveButtonText` en el botón aprobar, en lugar del texto “Save” o “Open”. El parámetro `parent` determina el contenedor sobre el que se abrirá el cuadro de diálogo. Si el parámetro `parent` es `null` el cuadro aparecerá sobre la pantalla. El método regresa el estado delcomponente al momento de cerrarse: • `JFileChooser.CANCEL_OPTION` • `JFileChooser.APPROVE_OPTION` • `JFileCHooser.ERROR_OPTION` Si ocurrió un error o se desecha el cuadro de diálogo. Lanza: `HeadlessException` – Si el ambiente no soporta un monitor, teclado o ratón.
`public int` **`showOpenDialog`**`(Component parent) throws HeadlessException` Despliega un cuadro de diálogo para abrir un archivo. El parámetro `parent` determina el contenedor sobre el que se abrirá el cuadro de diálogo. Si el parámetro `parent` es `null` el cuadro aparecerá sobre la pantalla. El método regresa el estado delcomponente al momento de cerrarse: • `JFileChooser.CANCEL_OPTION` • `JFileChooser.APPROVE_OPTION` • `JFileCHooser.ERROR_OPTION` Si ocurrió un error o se desecha el cuadro de diálogo. Lanza: `HeadlessException` – Si el ambiente no soporta un monitor, teclado o ratón.

Tabla 10.6. Métodos de la Clase `JFileChooser`. Cont.

`public int` **`showSaveDialog`**`(Component parent) throws HeadlessException` Despliega un cuadro de diálogo para guardar un archivo. El parámetro `parent` determina el contenedor sobre el que se abrirá el cuadro de diálogo. Si el parámetro `parent` es `null` el cuadro aparecerá sobre la pantalla. El método regresa el estado del componente al momento de cerrarse: • `JFileChooser.CANCEL_OPTION` • `JFileChooser.APPROVE_OPTION` • `JFileCHooser.ERROR_OPTION` Si ocurrió un error o se desecha el cuadro de diálogo. Lanza: `HeadlessException` – Si el ambiente no soporta un monitor, teclado o ratón.

Filtros de Nombres de Archivo para la clase `JFileChooser`

La clase JFileChooser nos permite manejar una lista de filtros para seleccionar los nombres de archivos que serán desplegados mediante los métodos:

- **`public void addChoosableFileFilter`**`(FileFilter filter)`
- `public FileFilter` **`getAcceptAllFileFilter`**`()`
- `public FileFilter[]` **`getChoosableFileFilters`**`()`
- `public FileFilter` **`getFileFilter`**`()`
- `public boolean` **`isAcceptAllFileFilterUsed`**`()`
- `public boolean` **`removeChoosableFileFilter`**`(FileFilter f)`
- `public void` **`resetChoosableFileFilters`**`()`
- `public void` **`setAcceptAllFileFilterUsed`**`(boolean b)`
- `public void` **`setFileFilter`**`(FileFilter filter)`

Los filtros son instancias de clases que heredan de la clase abstracta `javax.swing.filechooser.FileFilter` e implementan sus métodos. Esta clase tiene el mismo nombre que la interfaz `java.io.FileFilter` vista previamente y para evitar conflictos de nombre, si se importan ambos paquetes se deben utilizar los nombres completamente calificados.

Clase `javax.swing.filechooser.FileFilter`

Las clases que implementan los métodos de esta clase abstracta se usan para filtrar nombres de archivos. Estos métodos se muestran en la tabla 10.7.

Tabla 10.7. Métodos de la clase `javax.swing.filechooser.FileFilter`

`public abstract boolean` **`accept`**`(File f)` Regresa `true` si el archivo dado por la ruta abstracta del parámetro satisface al filtro, `false` en caso contrario.

Tabla 10.7. Métodos de la clase `javax.swing.filechooser.FileFilter`

`public abstract String` **`getDescription`**`()` Regresa una cadena con la descripción de este filtro.

Clase `FileNameExtensionFilter`

Una clase que implementa los métodos abstractos de la clase `javax.swing.filechooser.FileFilter` es la clase `FileNameExtensionFilter` y que nos permite crear filtros usando un conjunto de extensiones. La extensión de un nombre de archivo es la porción del nombre después del último punto. Los nombres de archivo que no contienen un punto no tienen extensión. Las comparaciones de las extensiones no toman en distinguen entre mayúsculas y minúsculas.

Los métodos de esta clase se muestran en la tabla 10.8.

Tabla 10.8. Métodos de la Clase `FileNameExtensionFilter`

`public abstract boolean` **`accept`**`(File f)` Regresa `true` si el archivo dado por la ruta abstracta del parámetro satisface al filtro, `false` en caso contrario. Un archivo satisface este filtro si su extensión se ajusta a una de las extensiones de este filtro o es un directorio.
`public abstract String` **`getDescription`**`()` Regresa una cadena con la descripción de este filtro.
`public String[]` **`getExtensions`**`()` Regresa un arreglo con las extensiones de este filtro.
`public String` **`toString`**`()` Regresa una cadena con la representación de este filtro.Su valor depende de la implementación.

Ejemplos sobre el uso de la clase `JFileChooser`

El siguiente código muestra el uso de una instancia de la clase JFileChooser para seleccionar el archivo con la imagen a leer. El código para leer la imagen se estudiará en el Tema 13 - Imágenes en Java.

```
  /**
     * Este metodo lee una imagen bmp, gif, jpg, png y la
     * guarda en el atributo dbi del tipo BufferedImage
     * @param frame Ventana sobre la que se despliega el cuadro
     * de dialogo JFileChooser
     */
    public void leeImagen(JFrame frame) {
        JFileChooser fc = new JFileChooser();

        // Elimina el filtro *.*
        fc.setAcceptAllFileFilterUsed(false);
```

```
        // Crea el filtro para las extenciones validas
        FileNameExtensionFilter extFiltro = new FileNameExtensionFilter(
                "Images", "bmp", "gif", "jpg", "png");

        // Establece el filtro para las extenciones validas
        fc.setFileFilter(extFiltro);

        // Despliega el cuadro de dialogo para seleccionar la imagen a abrir
        int returnVal = fc.showOpenDialog(frame);

        // Si se selecciono una imagen
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            // Obtiene el objeto File de la imagen seleccionada
            file = fc.getSelectedFile();

            // esta parte de código en la que se lee la imagen se estudiara
            // en el Tema 13 - Imágenes en Java
            try {
                // lee la imagen y la guarda en el atributo obi
                // del tipo BufferedImage
                obi = ImageIO.read(file);

                // Hace que la referencia dbi apunte a obi
                dbi = obi;
            } catch (IOException e) {
                JOptionPane.showMessageDialog(frame,
                        "Error al cargar imagen");
                return;
            }
        }
    }
```

El siguiente código muestra el uso de una instancia de la clase JFileChooser para seleccionar el archivo con la imagen a guardar. El código para guardar la imagen se estudiará en el Tema 13 - Imágenes en Java.

```
/**
     * Este metodo guarda la imagen bmp, gif, jpg, png del
     * atributo dbi del tipo BufferedImage en un archivo
     * @param frame Ventana sobre la que se despliega el cuadro
     * de dialogo JFileChooser
     */
    public void GuardaImagenComo(JFrame frame) {
        File fileSel = null;
        JFileChooser fc = new JFileChooser();

        // Elimina el filtro *.*
        fc.setAcceptAllFileFilterUsed(false);

        // Agrega varios filtros de imagenes
        fc.addChoosableFileFilter(
                new FileNameExtensionFilter("Imagen BMP", "bmp"));
        fc.addChoosableFileFilter(
                new FileNameExtensionFilter("Imagen GIF", "gif"));
```

```
        fc.addChoosableFileFilter(
                new FileNameExtensionFilter("Imagen JPG", "jpg"));
        fc.addChoosableFileFilter(
                new FileNameExtensionFilter("Imagen PNG", "png"));
        //Establece el nombre inicial de la imagen
        fc.setSelectedFile(file);

        // Despliega cuadro de dialogo para obtener el nombre
        // del archivo en el que se va a guardar la imagen
        int returnVal = fc.showSaveDialog(frame);

        // Si se selecciono un archivo
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            String nombreExt = null;

            // Obtiene el nombre del archivo seleccionado
            fileSel = fc.getSelectedFile();
            // Obtiene el nombre del filtro seleccionado
            FileNameExtensionFilter extFiltro =
               (FileNameExtensionFilter) fc.getFileFilter();
            // Obtiene la extension del nombre del filtro seleccionado
            String ext = extFiltro.getExtensions()[0];

            String path = fileSel.getPath();

            // Obtiene la extension del nombre del archivo seleccionado
            nombreExt = getExtension(fileSel);

            // Si el nombre seleccionado no corresponde a uno de imagen
            if (nombreExt != null && !esImageExtension(nombreExt)) {
                JOptionPane.showMessageDialog(frame,
                        "No es un archivo de imagen");
                return;
            }

            // Si no hay extension del nombre del archivo seleccionado
            if (nombreExt == null) {
                // Agregale la extension del nombre del filtro seleccionado
                path += "." + ext;
                fileSel = new File(path);
                nombreExt = ext;
            }

           // esta parte de código en la que se guarda la imagen se estudiara
           // en el Tema 13 - Imágenes en Java
           try {
                // Guarda la imagen
                ImageIO.write(dbi, nombreExt, fileSel);
            } catch (IOException e) {
                JOptionPane.showMessageDialog(frame,
                        "Error al guardar la imagen");
            }
        }
    }

    /**
     * Este metodo estatico obtiene la extension de un archivo
```

```
     * @param file Objeto de tipo File de la que se obtiene
     * la extension
     * @return Extension de un archivo
     */
    public static String getExtension(File file) {
        String ext = null;
        // Obtiene el nombre del archivo
        String s = file.getName();
        // busca el separador de la extension
        int pos = s.lastIndexOf('.');

        // Si hay un punto en el nombre y hay una
        // extension despues del punto
        if (pos > 0 && pos < s.length() - 1) {
            ext = s.substring(pos + 1).toLowerCase();
        }
        return ext;
    }

    /**
     * Este medodo determina si la extension del nombre de archivo
     * corresponte a una imagen
     * @param ext Extension del nombre de archivo
     * @return true si si la extension del nombre de archivo
     * corresponte a una imagen, false en caso contrario
     */
    public boolean esImageExtension(String ext) {
        String[] imagenesExt = {"bmp", "gif", "jpg", "png"};

        for (int i = 0; i < imagenesExt.length; i++) {
            if (ext.equals(imagenesExt[i])) {
                return true;
            }
        }

        return false;
    }
```

ARCHIVOS Y FLUJOS

Los datos almacenados en variables (ya sean simples o en arreglos) sólo se conservan mientras las variables están dentro de su ámbito. Esos datos se pierden cuando las variables lo abandonan o el programa termina su ejecución. Si deseamos que los datos se conserven debemos almacenar esos datos en un dispositivo de memoria secundaria: discos y cintas magnéticas, discos compactos, discos magneto-ópticos, etc. Los datos almacenados en forma permanente en un dispositivo de memoria secundaria se conocen como **datos persistentes**. Por otro lado, prácticamente todos los programas computacionales requieren comunicarse con una serie de dispositivos de entrada/salida, tales como el teclado, el monitor, los puertos serie y paralelo, etc.

Aunque cada dispositivo de memoria secundaria y cada dispositivo de entrada/salida es físicamente diferente de los otros, el sistema operativo nos oculta las diferencias entre ellos unificándolos bajo el concepto de archivo. Un **archivo** es un **dispositivo lógico** en el que podemos escribir información o del que podemos leer información. No todos los archivos tienen las mismas capacidades. Por ejemplo hay archivos a los que sólo podemos escribir: monitor y puerto paralelo. Hay otros de los que sólo podemos leer, por ejemplo el teclado. En algunos, como los archivos disco, podemos acceder la información en forma aleatoria, mientras que en otros sólo en forma secuencial, por ejemplo el teclado y los puertos serie y paralelo. Antes de poder leer o escribir en un archivo debemos de abrirlo y al terminar de trabajar con un archivo debemos cerrarlo. Al **abrir un archivo**, establecemos la comunicación entre el programa y el dispositivo y al **cerrar el archivo** destruimos esa comunicación.

Adicionalmente al concepto de archivo, muchos lenguajes de programación, incluyendo a Java, nos presentan una abstracción o interfaz común para los archivos: El **flujo (stream)**. Podemos considerar a un flujo como una corriente de bytes que se envía de nuestro programa a un archivo o que nuestro programa recibe de un archivo. Esta interfaz o intermediario nos oculta todas las diferencias que existen entre los archivos y nos permiten concentrarnos en qué datos queremos enviar o recibir de un archivo y no en cómo lo vamos a hacer. Al abrir un archivo se le asocia un flujo. Aunque un flujo es sólo una secuencia de bytes, la información que contiene puede ser interpretada por nuestro programa de dos formas: **flujos texto** y **flujos binarios**.

Flujos Texto

Si el programa interpreta los bytes que lee o escribe a un archivo como caracteres, se dice que el flujo es un flujo texto. Por ejemplo, algunos dispositivos de entrada y salida como el teclado, el monitor y el puerto paralelo le envían a, o reciben de la computadora bytes que representan caracteres. Por lo tanto, los flujos flujos asociados a esos dispositivos son flujos texto. También si deseamos guardar en disco información en forma de una secuencia de caracteres utilizaremos los flujos texto.

Un **flujo texto** es una secuencia de caracteres organizada en líneas, donde cada línea termina en un carácter de salto de línea, el carácter cuyo código ASCII es 0xA

Flujos Binarios

Si los bytes escritos o leídos de un archivo no son interpretados como caracteres, se dice que el flujo asociado al archivo es un flujo binario. En este caso los bytes pueden representar cualquier cosa: valores numéricos, objetos, imágenes, etc.

Clases de Entrada y Salida de Java

La API de Java nos proporciona un gran número de interfaces y clases que nos permite trabajar con archivos. Esas interfaces y clases se encuentran en el paquete:

`java.io`

En este tema sólo se verá una parte de las interfaces y clases del paquete `java.io` y para su estudio, se dividirán en tres grupos de acuerdo a como el programa interpreta el contenido de un archivo y a la forma de acceder a los datos almacenados en el archivo.

Archivos de Caracteres de Acceso Secuencial

En este caso, accederemos a los datos del archivo en forma secuencial, es decir del primero al último en ese orden. Para poder acceder a un dato intermedio deberemos haber leído los anteriores en orden. Además en este caso los datos del archivo serán interpretados por el programa como caracteres. En la figura 10.3 se muestra un diagrama parcial de las interfaces y clases que podemos utilizar en este tipo de problemas.

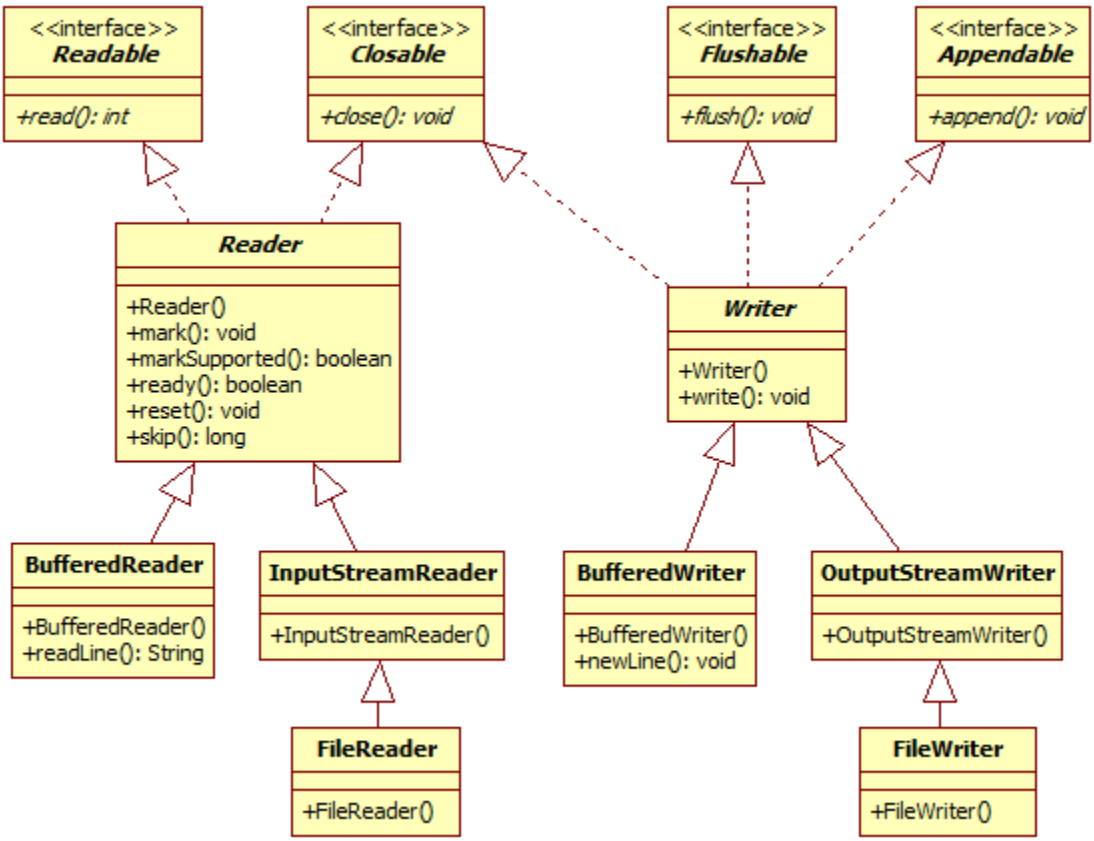


Figura 10.3 Interfaces y Clases para Archivos de Caracteres de Acceso Secuencial

- La interfaz `Readable` se encuentra en el paquete `java.lang` y representa una fuente de caracteres. Esta interfaz sólo declara un método que se muestra en la tabla 10.9.

Tabla 10.9 Método Declarado en la Interfaz `Readable`.

`int read(CharBuffer` *`cb`*`) throws IOException`
Intenta leer caracteres y los almacena en un buffer especificado. Regresa el número de caracteres agregados al buffer, -1 si la fuente de caracteres llegó a su final. **Lanza**: `IOException` – Si ocurre un error de entrada/salida `NullPointerException` – Si `cb` es nulo. `ReadOnlyBufferException` - Si `cb` es de solo lectura.

- La interfaz `Closable` representa una fuente o un destino que puede cerrarse. Esta interfaz sólo declara un método que se muestra en la tabla 10.10.

Tabla 10.10 Método Declarado en la Interfaz `Closable`.

`void close() throws IOException`
Cierra este flujo y libera los recursos del sistema asignados a él. **Lanza**: `IOException` – Si ocurre un error de entrada/salida

- La interfaz `Flushable` representa un destino que puede vaciarse. Escribe el contenido del buffer al flujo. Esta interfaz sólo declara un método que se muestra en la tabla 10.11.

Tabla 10.11 Método Declarado en la Interfaz `Flushable`.

`void flush() throws IOException`
Vacía el buffer escribiendo su contenido en el flujo. **Lanza**: `IOException` – Si ocurre un error de entrada/salida

- La interfaz `Appendable` se encuentra en el paquete `java.lang` y representa un objeto al que se le pueden agregar caracteres. Esta interfaz declara varios métodos que se muestran en la tabla 10.12.

Tabla 10.12 Métodos Declarados en la Interfaz `Appendable`.

`Appendable append(char` *`c`*`) throws IOException`
Agrega el carácter del parámetro `c` al objeto. Regresa una referencia al objeto agregable. **Lanza**: `IOException` – Si ocurre un error de entrada/salida

Tabla 10.12 Métodos Declarados en la Interfaz `Appendable`. Cont.

`Appendable append(charSecuence` *`csq`***`) throws IOException`** Agrega la secuencia de caracteres dada por el parámetro *`csq`* al objeto. Regresa una referencia al objeto agregable. **Lanza**: `IOException` **–** Si ocurre un error de entrada/salida
`Appendable append(CharSecuence` *`csq`***`, int`** *`start`***`, int`** *`end`***`) throws IOException`** Agrega la subsecuencia de caracteres del parámetro *`csq`* a partir del carácter en la posición dada por el parámetro *`start`* hasta el carácter en la posición dada por el parámetro *end*. Regresa una referencia al objeto agregable. **Lanza:** `IndexOutOfBoundsException` - Si **`start`** o **`end`** son negativos, **`start > end`,** o `end` es mayor que `csq.length()` `IOException` **–** Si ocurre un error de entrada/salida

- La clase abstracta `Reader` es para leer de flujos de caracteres. Los métodos de esta clase se muestran en la tabla 10.13.

Tabla 10.13 Métodos de la clase `Reader`.

`Reader()` Crea un flujo de caracteres de entrada.
`void mark(int` *`readAheadLimit`***`) throws IOException`** Marca la posición actual en el archivo. Las siguientes llamadas al método `reset()` posicionaran al archivo en este punto. No todos los flujos de entrada de caracteres soportan esta operación. El parámetro *`readAheadLimit`* es el límite de caracteres que pueden leerse mientras se preserva la marca. Después de leerse este número de caracteres, intentar reestablecer el flujo puede fallar. **Lanza**: `IOException` **–** Si el flujo no sopota el método `mark()` u ocurre un error de entrada/salida
`boolean markSupported()` Establece si el flujo soporta la operación `mark()`. **Regresa**: `true` si y sólo si el flujo soporta la operación `mark()`.
`int read() throws IOException` Lee un solo caracter. El método se bloquea hasta que haya un carácter, ocurra un error de entrada/salida o se llegue al final del flujo. Regresa el carácter leído como un entero en el rango de 0 a 65,535 o -1 si se ha alcanzado el final del flujo. **Lanza**: `IOException` **–** Si ocurre un error de entrada/salida

Tabla 10.13 Métodos de la clase `Reader`. Cont.

`int read(char[]` *`cbuf`*`)`** **`throws IOException`** Lee caracteres y los almacena en el arreglo dado por el parámetro *`cbuf`*. Regresa el número de caracteres leídos o -1 si se ha alcanzado el final del flujo.. **Lanza**: `IOException` **–** Si ocurre un error de entrada/salida
`abstract int read(char[]` *`cbuf`*`, int` *`off`***`, int` *`len`***`) throws IOException`** Lee hasta *`len`* caracteres y los almacena en el arreglo dado por el parámetro *`cbuf`* a partir de la posición *`off`*. Regresa el número de caracteres leídos o -1 si se ha alcanzado el final del flujo. **Lanza**: `IOException` **–** Si ocurre un error de entrada/salida
`int read(CharBuffer` *`target`*`) throws IOException`** Intenta leer caracteres en el buffer de caracteres dado por el parámetro *`target`*. Regresa el número de caracteres leídos o -1 si se ha alcanzado el final del flujo. **Lanza:** `IOException` – Si ocurre un error de entrada/salida `NullPointerException` **–** Si *`target`* es nulo. `ReadOnlyBufferException` - Si *`target`* es un buffer de sólo lectura.
`boolean ready() throws IOException` Prueba si el flujo está listo para ser leído. Regresa `true` si hay garantía que la siguiente invocación a `read()` no se bloqueará, `false` en caso contrario. **Lanza:** `IOException` **–** Si ocurre un error de entrada/salida
`void reset() throws IOException` Restablece el flujo. Si el flujo se ha marcado se intenta reposicionar en la marca. No todos los flujos de entrada de caracteres soportan esta operación. **Lanza:** `IOException` – Si el flujo no se ha marcado, si la marca se ha invalidado, si el flujo no soporta la operación `reset()` o ocurre un error de entrada/salida.
`long skip(long` *`n`*`) throws IOException`** Lee y descarta n caracteres. Este método se bloqueará hasta que haya varios caracteres, ocurra un error de entrada/salida o se llegue al final del flujo. Regresa el número de caracteres a leídos y descartados. **Lanza:** `IllegalArgumentException` – Si n es negativo. `IOException` – Si ocurre un error de entrada/salida.

- La clase `BufferedReader` es para leer de flujos de caracteres. Almacena los caracteres leídos en un buffer para eficientar la lectura de caracteres, arreglos y líneas. Se puede especificar el tamaño del buffer o se puede utilizar el valor por omisión, el cual es suficientemente grande para la mayoría de los casos.

Por lo general se aconseja envolver cualquier objeto de la jerarquía de clases de `Reader`, por ejemplo las clases `FileReader` e `InputStreamReader` con un objeto del tipo `BufferedReader` para eficientar su acceso. Por ejemplo:

```
BufferedReader in = new BufferedReader(new FileReader("datos.txt"));
```

Los métodos de esta clase se muestran en la tabla 10.14.

Tabla 10.14 Métodos de la clase `BufferedReader`.

`BufferedReader(Reader` *`in`*`)` Crea un flujo de caracteres de entrada con buffer, a partir del flujo de entrada *`in`*, el buffer de entrada es de tamaño predeterminado.
`BufferedReader(Reader` *`in`*`, int` *`sz`*`)` Crea un flujo de caracteres de entrada con buffer, a partir del flujo de entrada *`in`*, el buffer de entrada es de tamaño `sz`. **Lanza**: `IllegalArgumentException` – Si `sz` <= 0
`String readLine()throws IOException` Lee una línea de texto. Una línea es una secuencia de caracteres terminada por un carácter de salto de línea (‘\n’), un carácter de retorno de carro (‘\r’), o un carácter de salto de línea seguido por un carácter retorno de carro. Regresa una cadena con el contenido de la línea sin incluir los caracteres de terminación, o `null` si se llegó al final del flujo. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.

- La clase `InputStreamReader` actúa como un puente entre flujos de bytes y flujos de caracteres: Lee bytes y los decodifica como caracteres. Cada invocación de uno de los métodos read() de `InputStreamReader` causa que se lean uno o más bytes del flujo de entrada de bytes. Por eficiencia se aconseja envolver el objeto `InputStreamReader` dentro de un objeto `BufferedReader` para eficientar su acceso. Por ejemplo:

```
BufferedReader in
               = new BufferedReader(new InputStreamReader(System.in));
```

El constructor de esta clase se muestra en la tabla 10.15.

Tabla 10.15 Constructor de la clase `InputStreamReader`.

`InputStreamReader(InputStream` *`in`*`)` Crea un flujo del tipo `InputStreamReader` a partir del flujo de entrada *`in`*.

- La clase `FileReader` permite leer archivos de caracteres.

  El constructor de esta clase se muestra en la tabla 10.16.

Tabla 10.16 Constructor de la clase `FileReader`.

`FileReader(String` *`fileName`*`) throws FileNotFoundException` Crea un flujo del tipo `FileReader` para el archivo de nombre *`fileName`*. **Lanza**: `FileNotFoundException` - Si el archivo no existe, si el nombre es un directorio en lugar de un archivo o por si alguna otra razón no se puede abrir el archivo para lectura.

- La clase abstracta `Writer` es para escribir a flujos de caracteres. Los métodos de esta clase se muestran en la tabla 10.17.

Tabla 10.17 Métodos de la clase `Writer`.

`Writer()` Crea un flujo de caracteres de salida.
`void write(int` ***`c`***`)` Escribe un solo caracter. El carácter del parámetro *`c`*. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void Write(char[]` *`cbuf`*`) throws IOException` Escribe el arreglo de caracteres dado por el parámetro *`cbuf`*. **Lanza**: `IOException` – Si ocurre un error de entrada/salida.
`abstract void Write(char[]` *`cbuf`***`, int`** *`off`***`, int`** *`len`*`) throws IOException` Escribe *`len`* caracteres del arreglo de caracteres *`cbuf`*, a partir de la posición *`off`*. **Lanza**: `IOException` – Si ocurre un error de entrada/salida.
`void Write(String` *`str`*`) throws IOException` Escribe la cadena dada por el parámetro *`str`*. **Lanza**: `IOException` – Si ocurre un error de entrada/salida.
`void Write(String` *`str`***`, int`** ***`off`***`, int`** *`len`*`) throws IOException` Escribe *`len`* caracteres de la cadena *`str`*, a partir de la posición *`off`*. **Lanza**: `IOException` – Si ocurre un error de entrada/salida.

- La clase `BufferedWriter` es para escribir a flujos de caracteres. Utiliza un buffer para eficientar la escritura de caracteres, arreglos y líneas. Se puede especificar el tamaño del buffer o se puede utilizar el valor por omisión, el cual es suficientemente grande para la mayoría de los casos.

Por lo general se aconseja envolver cualquier objeto de la jerarquía de clases de `Writer`, por ejemplo las clases `FileWriter` e `OutputStreamWriter` `InputStreamReader` con un objeto del tipo `BufferedWriter` para eficientar su acceso. Por ejemplo:

```
PrintWriter out = new PrintWriter(new
            BufferedWriter(new FileWriter ("datos.txt")));
```

Los métodos de esta clase se muestran en la tabla 10.118.

Tabla 10.118 Métodos de la clase `BufferedWriter`.

`BufferedWriter(Writer` *`out`***`)`** Crea un flujo de caracteres de salida con buffer a partir del flujo de entrada *`out`*. Usa un buffer de tamaño predeterminado.
`BufferedWriter(Writer` *`out`***`, int`** *`sz`***`)`** Crea un flujo de caracteres de salida con buffer a partir del flujo de entrada *`out`*. Usa un buffer de tamaño `sz`. **Lanza**: `IllegalArgumentException` – Si `sz` <= 0
`void newLine()throws IOException` Escribe un separador de líneas. Se recomienda utilizar este método en lugar de escribir el carácter directamente ya que el carácter empleado para separar líneas es dependiente de la plataforma. **Lanza**: `IOException` – Si ocurre un error de entrada/salida.

- La clase `OutputStreamWriter` actúa como un puente entre flujos de caracteres y flujos de bytes: Escribe caracteres codificándolos como bytes. Cada invocación de uno de los métodos write() de `OutputStreamWriter` causa que uno o más caracteres sean codificados como bytes y se agreguen al buffer antes de ser enviados al flujo de salida de bytes. Por eficiencia se aconseja envolver el objeto `OutputStreamWriter` dentro de un objeto `BufferedWriter` para eficientar su acceso .Por ejemplo:

```
Writer out = new BufferedWriter(new
                                    OutputStreamWriter(System.out));
```

El constructor de esta clase se muestra en la tabla 10.19.

Tabla 10.19 Constructor de la clase `OutputStreamWriter`.

`OutputStreamWriter(OutputStream` *`out`***`)`** Crea un flujo del tipo `OutputStreamWriter` a partir del flujo de salida *`out`*.

- La clase `FileWriter` permite escribir a archivos de caracteres.

Los constructores de esta clase se muestran en la tabla 10.20.

Tabla 10.20 Constructores de la clase `FileWriter`.

`FileWriter(String` *`fileName`*`) throws IOException` Crea un flujo de salida del tipo `FileWriter` para el archivo de nombre *`fileName`*. **Lanza**: `IOException` - Si el archivo no existe y no puede ser escrito, si el nombre es un directorio en lugar de un archivo o por si alguna otra razón no se puede abrir el archivo para escritura.
`FileWriter(String` *`fileName,`* **`boolean`** *`append`*`) throws IOException` Crea un flujo de salida del tipo `FileWriter` para el archivo de nombre *`fileName`*. El parámetro `append` indica si los datos a escribir se agregan al final del archive o no. **Lanza**: `IOException` - Si el archivo no existe, si el nombre es un directorio en lugar de un archivo o por si alguna otra razón no se puede abrir el archivo para lectura.

Ejemplo de Archivos de Caracteres de Acceso Secuencial

El siguiente ejemplo es un programa que lee un archivo texto que contiene el código fuente de una clase de Java y cuenta el número de veces que aparecen las palabras reservadas: `if`, `switch`, `for`, `while` y `do`. Los resultados los almacena en otro archivo texto. El programa consiste de dos clases: `Estadisticas` y `PruebaEstadisticas`. La clase `Estadisticas` contiene métodos para abrir el archivo con el código fuente y leerlo por líneas, contar las ocurrencias de las palabras reservadas y escribir los resultados en otro archivo. La clase `PruebaEstadisticas` sólo sirve para probar los métodos de la clase `Estadisticas`.

Estadisticas.java

```
/*
 * Estadisticas.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */

package estadisticas;

import java.io.*;

/**
 * Este programa lee un archivo texto que contiene el código fuente de una
 * clase de Java y cuenta el número de veces que aparecen las palabras
 * reservadas: if, switch, for, while y do. Los resultados los almacena en
 * otro archivo texto.
 *
 * @author mdomitsu
```

```
 */
public class Estadisticas {
  String nomArchivoEntrada;
  String nomArchivoSalida;
  String palabras[];
  int cuentaPalabras[];

  /**
   * Este constructor inicializa los atributos de la clase e invoca a los
   * métodos que abren el archivo fuente, lo procesan y escriben el resultado
   * en otro archivo.
   * @param nomArchivoEntrada Nombre del archivo con el código fuente en java
   * @param nomArchivoSalida  Nombre del archivo con los resultados
   * @param palabras lista de las palabras reservadas a buscar.
   */
  public Estadisticas(String nomArchivoEntrada, String nomArchivoSalida,
                      String palabras[]) {
    this.nomArchivoEntrada = nomArchivoEntrada;
    this.nomArchivoSalida = nomArchivoSalida;
    this.palabras = palabras;

    // Crea e inicializa el arreglo donde se almacenarán el número de veces
    // que aparecen las palabras reservadas
    cuentaPalabras = new int[palabras.length];

    for (int i = 0; i < palabras.length; i++) cuentaPalabras[i] = 0;

    // Procesa el archivo con el código fuente
    procesaArchivo();

    // Escribe los resultados en un archivo texto
    escribeResultados();
  }

  /**
   * Este método abre el archivo con el código fuente, lo lee línea por
   * línea, cuenta las ocurrencias de cada palabra clave y las almacena en el
   * atributo cuentaPalabras.
   */
  public void procesaArchivo() {
    FileReader fileReader = null;
    BufferedReader bufferedReader;
    String linea = "";

    // Abre el archivo con el código fuente
    try {
      fileReader = new FileReader(nomArchivoEntrada);
    }
    catch(FileNotFoundException fnfe) {
      System.out.println("Error: No existe el archivo " + nomArchivoEntrada);
      return;
    }

    // Se envuelve el archivo con el código fuente con un archivo del tipo
    // BufferedReader para eficientar su acceso
    bufferedReader = new BufferedReader(fileReader);
```

```
    try {
      while(true) {
        // Obtén la siguiente línea del archivo con el código fuente
        linea = bufferedReader.readLine();

        // Si ya no hay líneas, termina el ciclo
        if (linea == null) break;

        // Cuenta y acumula las ocurrencias de cada palabra resevada en la
        // línea
        cuentaPalabrasLinea(linea);
      }
    }
    catch (IOException ioe) {
      System.out.println("Error al procesar el archivo" + nomArchivoEntrada);
      return;
    }

    // Cierra el archivo
    try {
      bufferedReader.close();
      fileReader.close();
    }
    catch (IOException ioe) {
      System.out.println("Error al cerrar el archivo" + nomArchivoEntrada);
      return;
    }
  }

  /**
   * Este método escribe en un archivo texto, las ocurrencias de cada palabra
   * reservada en un archivo con código fuente de Java
   */
  public void escribeResultados() {
    FileWriter fileWriter = null;
    BufferedWriter bufferedWriter;

    // Abre el archivo donde se escribirán los resultados
    try {
      fileWriter = new FileWriter(nomArchivoSalida);
    }
    catch(IOException ioe) {
      System.out.println("Error, no se pudo crear el archivo" +
                         nomArchivoSalida);
      return;
    }

    // Se envuelve el archivo donde se escribirán los resultados con un
    // archivo del tipo BufferedWriter para eficientar su acceso
    bufferedWriter = new BufferedWriter((OutputStreamWriter)(fileWriter));

    try {
      bufferedWriter.write("Frecuencia de las palabras reservadas");
      bufferedWriter.newLine();
      bufferedWriter.write("en la clase: " + nomArchivoEntrada);
      bufferedWriter.newLine();
      bufferedWriter.newLine();
```

```
    // Para cada palabra reservada
     for (int i = 0; i < palabras.length; i++) {
       // escribe en el archivo la palabra reservada y su ocurrencia
       bufferedWriter.write(palabras[i] + ": " + cuentaPalabras[i]);
       // Escribe en el archivo un salto de línea
       bufferedWriter.newLine();
     }
   }
   catch(IOException ioe) {
     System.out.println("Error al escribir al archivo" + nomArchivoSalida);
     return;
   }

   // Cierra el archivo
   try {
     bufferedWriter.close();
     fileWriter.close();
   }
   catch (IOException ioe) {
     System.out.println("Error al cerrar el archivo" + nomArchivoSalida);
     return;
   }
 }

 /**
  * Este método cuenta las ocurrencias de las palabras reservadas en la
  * línea dada por el parámetro y las acumula en el arreglo cuentaPalabras
  * @param linea
  */
 public void cuentaPalabrasLinea(String linea) {
   // Para cada palabra de las palabras reservadas
   for (int i = 0; i < palabras.length; i++) {
     // Cuenta el número de ocurrencias en la línea
     cuentaPalabras[i] += cuentaPalabraLinea(linea, palabras[i]);
   }
 }

 /**
  * Este método regresa el número de veces que la palabra reservada dada por
  * el parámetro ocurre en la línea dada por el parámetro
  * @param linea Línea en la que se busca la palabra reservada
  * @param palabra Palabra reservada a buscar
  * @return Número de veces que la palabra reservada dada por el parámetro
  * ocurre en la línea dada por el parámetro
  */
 public int cuentaPalabraLinea(String linea, String palabra) {
   int n = 0;
   int pos = 0;

   while(true) {
     // Busca la primer ocurrencia de la palabra en la línea a partir
     // de la posición pos
     pos = linea.indexOf(palabra, pos);


     // Si la palabra no existe, termina
```

```
      if(pos <= -1) return n;

      // Incrementa el número de ocurrencias
      n++;

      // Se prepara para continuar la búsqueda a partir de la última
      // ocurrencia
      pos += palabra.length();
    }
  }
}
```

PruebaEstadisticas.java

```
/*
 * PuebaEstadisticas.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */

package estadisticas;

/**
 * Esta clase sirve para probar la clase Estadísticas que lee un archivo
 * texto que contiene el código fuente de una clase de Java y cuenta el
 * número de veces que aparecen las palabras reservadas: if, switch, for,
 * while y do. Los resultados los almacena en otro archivo texto.
 *
 * @author mdomitsu
 */
 */
public class PuebaEstadisticas {

  /**
   * Método main(). Invoca a los métodos de la clase Estadisticas
   * @param args Argumentos en la línea de comando
   */
  public static void main(String[] args) {
    PuebaEstadisticas puebaEstadisticas1 = new PuebaEstadisticas();
    String palabrasReservadas[] = {"if", "switch", "while", "for", "do"};

    // Crea un objeto del tipo Estadisticas y le pasa los nombres de los
    // archivos con el código fuente, los resultados y la lista de palabras
    // reservadas a buscar
    Estadisticas estadisticas =
        new Estadisticas("src\\estadisticas\\Estadisticas.java",
                         "estadisticas.txt", palabrasReservadas);
  }
}
```

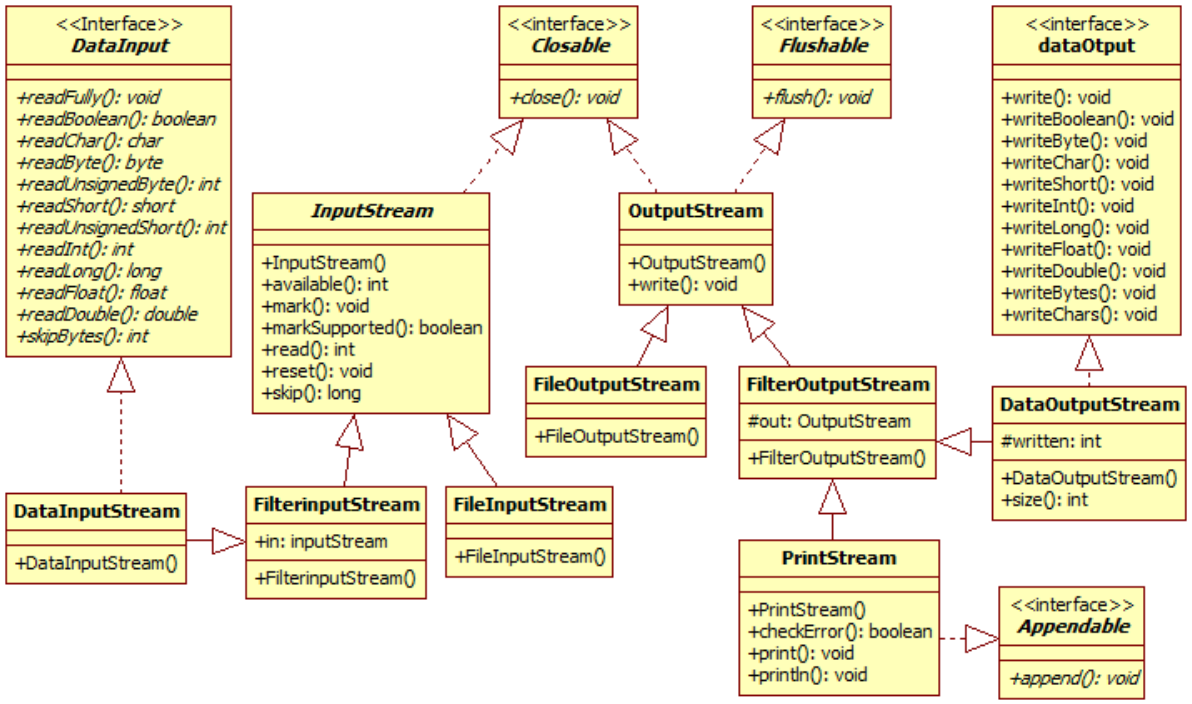
Al ejecutar el programa sobre la clase Estadisticas, tendremos el siguiente resultado:

```
Frecuencia de las palabras reservadas
en la clase: src\estadisticas\Estadisticas.java

if: 3
switch: 0
while: 3
for: 3
do: 0
```

Archivos de Acceso Secuencial de Bytes

En este caso, accederemos a los datos del archivo en forma secuencial y los datos del archivo serán interpretados por el programa como bytes. En la figura 10.4 se muestra un diagrama parcial de las interfaces y clases que podemos utilizar en este tipo de problemas.



10.4 Interfaces y Clases para Archivos de Bytes de Acceso Secuencial

- La interfaz `DataInput` se encuentra en el paquete `java.io` y provee métodos para leer bytes de un flujo binario y convertirlos a datos de los tipos primitivos.

  Los métodos declarados por esta interfaz se muestran en la tabla 10.21:

Tabla 10.21 Métodos de la Interfaz `DataInput`.

`void readFully(byte[] b) throws IOException` Lee algunos bytes del archivo y los almacena en el arreglo b. El número de bytes leídos es igual al tamaño del arreglo b. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos los bytes. `IOException` – Si ocurre un error de entrada/salida. `NullPointerException` – Si b es `null`.
`void readFully(byte[] b , int off, int len) throws IOException` Lee `len` bytes del archivo y los almacena en el arreglo b a partir de la posición `off`. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos los bytes. `IOException` – Si ocurre un error de entrada/salida. `NullPointerException` – Si b es `null`. `IndexOutOfBoundsException` – Si `off` o `len` es negativo o si `off+len` es mayor que la longitud del arreglo b.
`boolean readBoolean()throws IOException` Lee un byte de la entrada y regresa true si el byte es diferente de cero, falso en caso contrario. Este método se emplea para leer el booleano escrito por el método `writeBoolean()` de la interfaz `DataOutput`. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.
`char readChar()IOException` Lee y regresa un carácter Unicode de la entrada. Este método se emplea para leer el carácter Unicode escrito por el método `writeChar()` de la interfaz `DataOutput`. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.
`byte readByte()IOException` Lee y regresa un byte de la entrada. El byte es considerado como un valor signado en el rango -128 a 127, inclusive. Este método se emplea para leer el byte escrito por el método `writeByte()` de la interfaz `DataOutput`. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.

Tabla 10.21 Métodos de la Interfaz `DataInput`. Cont.

`int readUnsignedByte()IOException` Lee un byte de la entrada y lo regresa como un `int` en el rango de 0 a 255. Este método se emplea para leer el byte escrito por el método `writeByte()` de la interfaz `DataOutput` si el argumento del método representaba un valor en el rango de 0 a 255. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.
`short readShort()IOException` Lee dos bytes de la entrada y los regresa como un entero corto. Este método se emplea para leer el entero corto escrito por el método `writeShort()` de la interfaz `DataOutput`. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.
`int readUnsignedShort()IOException` Lee dos bytes de la entrada y lo regresa como un entero en el rango de 0 a 65535. Este método se emplea para leer el entero corto escrito por el método `writeShort()` de la interfaz `DataOutput` si el argumento del método representaba un valor en el rango de 0 a 65535. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.
`int readInt()IOException` Lee cuatro bytes de la entrada y los regresa como un entero. Este método se emplea para leer el entero escrito por el método `writeInt()` de la interfaz `DataOutput`. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.
`long readLong()IOException` Lee ocho bytes de la entrada y los regresa como un entero largo. Este método se emplea para leer el entero largo escrito por el método `writeLong()` de la interfaz `DataOutput`. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.
`float readFloat()IOException` Lee cuatro bytes de la entrada y los regresa como un flotante. Este método se emplea para leer el flotante escrito por el método `writeFloat()` de la interfaz `DataOutput`. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.

Tabla 10.21 Métodos de la Interfaz `DataInput`. Cont.

`double readDouble()IOException` Lee ocho bytes de la entrada y los regresa como un doble. Este método se emplea para leer el doble escrito por el método `writeDouble()` de la interfaz `DataOutput`. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.
`int skipBytes(int n) IOException` Intenta leer y descartar n bytes de datos del archivo de entrada. Puede leer y descartar menos bytes si se llega antes al final del archivo. El método regresa el número de bytes leídos y descartados. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.

- La interfaz `DataOutput` se encuentra en el paquete `java.io` y provee métodos para convertir datos de los tipos primitivos a bytes y escribirlos a un flujo binario. Los métodos declarados por esta interfaz se muestran en la tabla 10.22:

Tabla 10.22 Métodos de la Interfaz `DataOutput`.

`void write(int b)` Escribe al flujo de bytes, el byte dado por el parámetro *b*. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void write(byte[] b)` Escribe al flujo de bytes, todos los bytes del arreglo b. **Lanza:** `IOException` – Si ocurre un error de entrada/salida. `NullPointerException` – Si *b* es `null`.
`void write(byte[] b, int off, int len)` Escribe `len` bytes del arreglo b al flujo de bytes, a partir de la posición `off`. **Lanza:** `IOException` – Si ocurre un error de entrada/salida. `NullPointerException` – Si *b* es `null`. `IndexOutOfBoundsException` – Si `off` o `len` es negativo o si `off+len` es mayor que la longitud del arreglo b.

Tabla 10.22 Métodos de la Interfaz `DataOutput`. Cont.

`void writeBoolean(boolean` *`v`*`)` Escribe al flujo de bytes el valor booleano de v, Si v es verdadero se escribe un 1, 0 en caso contrario. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void writeByte(int` *`v`*`)` Escribe al flujo de bytes el byte del parámetro *v*. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void` **`writeChar`**`(int v)` Escribe al flujo de bytes el carácter en el parámetro *v*. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void writeShort``(int v)` Escribe al flujo de bytes el entero corto del parámetro *v*. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void` **`writeInt`**`(int v)` Escribe al flujo de bytes el entero del parámetro *v*. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void` **`writeLong`**`(long v)` Escribe al flujo de bytes el entero largo del parámetro *v*. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void` **`writeFloat`**`(float v)` Escribe al flujo de bytes el flotante del parámetro *v*. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void` **`writeDouble`**`(double v)` Escribe al flujo de bytes el doble del parámetro *v*. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.

Tabla 10.22 Métodos de la Interfaz `DataOutput`. Cont.

`void writeBytes``(String s)` Escribe al flujo de bytes la cadena *`s`*. Por cada carácter de la cadena se escribe un byte. **Lanza:** `IOException` – Si ocurre un error de entrada/salida. `NullPointerException` – Si *`s`* es `null`.
`void writeChars``(String s)` Escribe al flujo de bytes la cadena *`s`*. Por cada carácter de la cadena se escribe un caracter. **Lanza:** `IOException` – Si ocurre un error de entrada/salida. `NullPointerException` – Si *`s`* es `null`.

- La clase abstracta **`InputStream`** es la superclase de odas las clases usadas para leer de flujos de bytes. Los métodos de esta clase se muestran en la tabla 10.23.

Tabla 10.23 Métodos de la clase `InputStream`.

`InputStream()` Crea un flujo de bytes de entrada.
`int available()throws IOException` Regresa el número de bytes que pueden leerse (o leerse y descartarse) del flujo de entrada sin que el flujo se bloqueé en la siguiente invocación de un método de este flujo. Regresa **Lanza**: `IOException` **–** Si el flujo no sopota el método `mark()` u ocurre un error de entrada/salida
`void mark(int` *`readLimit`***`)`** Marca la posición actual en el archivo. Las siguientes llamadas al método `reset()` posicionaran al archivo en este punto. El parámetro *`readLimit`* es el límite de caracteres que pueden leerse mientras se preserva la marca. Después de leerse este número de caracteres, intentar reestablecer el flujo puede fallar.
`boolean markSupported()` Establece si el flujo soporta la operación `mark()`. Regresa `true` si y sólo si el flujo soporta la operación `mark()`.
`abstract int read() throws IOException` Lee el siguiente byte del flujo de entrada. El método se bloquea hasta que haya un carácter, ocurra un error de entrada/salida o se llegue al final del flujo. Regresa el byte leído como un entero en el rango de 0 a 2555 o -1 si se ha alcanzado el final del flujo. **Lanza**: `IOException` **–** Si ocurre un error de entrada/salida

Tabla 10.23 Métodos de la clase `InputStream`. Cont.

`int read(byte[]` *`b`*`) throws IOException` Lee algunos bytes y los almacena en el arreglo *b*. Regresa el número de caracteres leídos o -1 si se ha alcanzado el final del flujo. **Lanza**: `IOException` **–** Si ocurre un error de entrada/salida `NullPointerException` **–** Si *b* es nulo.
`int read(byte[]` *`b`*`, int` *`off`*`, int` *`len`*`) throws IOException` Lee hasta *`len`* bytes y los almacena en el arreglo *b*, a partir de la posición *`off`*. Regresa el número de caracteres leídos o -1 si se ha alcanzado el final del flujo.. **Lanza**: `IOException` **–** Si ocurre un error de entrada/salida `IndexOutOfBoundsException` - Si **`off`** o **`len`** son negativos, **`off + len >`** `b.length()` `NullPointerException` **–** Si *b* es nulo.
`void reset() throws IOException` Restablece el flujo. Si el flujo se ha marcado se intenta reposicionar en la marca. No todos los flujos de entrada de caracteres soportan esta operación. **Lanza:** `IOException` – Si el flujo no se ha marcado, si la marca se ha invalidado.
`long skip(long` *`n`*`) throws IOException` Lee y descarta n bytes. Este método se bloqueará hasta que haya varios caracteres, ocurra un error de entrada/salida o se llegue al final del flujo. Regresa **e**l número de caracteres a leídos y descartados. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.

- La clase `FileInputStream` permite leer archivos de bytes.

  El constructor de esta clase se muestra en la tabla 10.24.

Tabla 10.24 Constructor de la clase `FileInputStream`.

`FileInputStream(String` *`fileName`*`) throws FileNotFoundException` Crea un flujo del tipo `FileInputStream` para elarchivo de nombre *`fileName`*. **Lanza**: `FileNotFoundException` - Si el archivo no existe, si el nombre es un directorio en lugar de un archivo o por si alguna otra razón no se puede abrir el archivo para lectura.

- La clase `FilterInputStream` contiene a otro flujo de entrada, al que utiliza como su única fuente de datos, posiblemente transformando los datos o agregándoles alguna funcionalidad adicional.

  El constructor de esta clase se muestra en la tabla 10.25.

Tabla 10.25 Constructor de la clase `FilterInputStream`.

`FilterInputStream(InputStream` *`in`***`)`** Crea un flujo del tipo `FilterInputStreamReader` a partir del flujo de entrada *`in`*.

- La clase `DataInputStream` permite leer tipos de datos primitivos de Java del flujo de entrada, implementando la interfaz `DataInput`.

  El constructor de esta clase se muestra en la tabla 10.26.

Tabla 10.26 Constructor de la clase `DataInputStream`.

`DataInputStream(InputStream` *`in`*`) throws FileNotFoundException` Crea un flujo del tipo `DataInputStreamReader` a partir del flujo de entrada *`in`*.

- La clase abstracta `OutputStream` es la superclase de todas las clases para escribir a flujos de bytes. Los métodos de esta clase se muestran en la tabla 10.27.

Tabla 10.27 Métodos de la clase `OutputStream`.

`OutputStream()` Crea un flujo de bytes de salida.
`abstract void write(int` ***`c`***`)` Escribe un solo carácter al flujo de bytes de salida. El carácter a escribir está dado por el parámetro **`c`**. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void Write(byte[]` *`b`*`) throws IOException` Escribe **`b.length`** bytes del arreglo de bytes b al flujo de bytes de salida. **Lanza**: `IOException` – Si ocurre un error de entrada/salida.
`abstract void Write(byte[]` *`b`***`, int`** *`off`***`, int`** *`len`*`) throws IOException` Escribe `len` bytes bytes del arreglo de bytes *b* a partir de la posición `off`, al flujo de bytes de salida. **Lanza**: `IOException` – Si ocurre un error de entrada/salida. `IndexOutOfBoundsException` - Si **`off`** o **`len`** son negativos, **`off + len >`** `b.length()` `NullPointerException` **–** Si *b* es nulo.

- La clase `FileOutputStream` permite escribir a archivos de bytes.

  Los constructores de esta clase se muestran en la tabla 10.28.

Tabla 10.28 Constructores de la clase `FileOutputStream`.

`FileOutputStream(String` *`fileName`*`) throws IOException` Crea un flujo del tipo `FileOutputStream` para el archivo de nombre *`fileName`*. **Lanza**: `IOException` - Si el archivo no existe y no puede ser escrito, si el nombre es un directorio en lugar de un archivo o por si alguna otra razón no se puede abrir el archivo para escritura.
`FileOutputStream(String` *`filename,`* **`boolean`** *`append`*`)` **`throws IOException`** Crea un flujo del tipo `FileOutputStream` para el archivo de nombre *`fileName`*. El parámetro `append` indica si los datos a escribir se agregan al final del archive o no.. **Lanza**: `IOException` - Si el archivo no existe, si el nombre es un directorio en lugar de un archivo o por si alguna otra razón no se puede abrir el archivo para lectura.

- La clase `FilterOutputStream` es la superclase de todas las clases que filtran flujos de salida. Estos flujos envuelven a un flujo de salida existente y lo utiliza como su sumidero básico de bytes, posiblemente transformando los datos o agregándoles alguna funcionalidad adicional.

  El constructor de esta clase se muestra en la tabla 10.29.

Tabla 10.29 Constructor de la clase `FilterOutputStream`.

`FilterOutputStream(OutputStream` *`out`***`)`** Crea un flujo del tipo `FilterOutputStreamWriter` para el flujo de entrada *`out`*.

- La clase `DataOutputStream` permite que una aplicación escriba datos de tipos primitivos de Java a un flujo de salida.

  Los métodos de esta clase se muestran en la tabla 10.30.

Tabla 10.30 Métodos de la clase `DataOutputStream`.

`DataOutputStream`**`(OutputStream`** *`out`***`)`** Crea un flujo de salida de datos para el flujo de entrada *`out`*.
`final int size()` Regresa el número de bytes escritos al flujo de salida.

- La clase `PrintStream` pemite escribir la representación de varios tipos de datos a otros flujos de salida. Si un error ocurre se invoca al método interno `setError()` que establece a true el valor de una bandera interna que puede ser probada mediante el método `checkerror()`.

  Los métodos de esta clase se muestran en la tabla 10.31.

Tabla 10.31 Métodos de la clase `PrintStream`.

`PrintStream(`**`OutputStream`** *`out`*`)` Crea un flujo de salida nuevo a partir del f lujo de salida *`out`*.
`PrintStream(`**`String`** *`fileName`*`)` **`throws FileNotFoundException`** Crea un flujo de salida nuevo para el archivo de nombre *`filename`*. Si el archivo existe su longitud se trunca a cero. **Lanza:** `FileNotFoundException` – Si no se puede crear el archivo.
`boolean checkError()` Vacía el buffer de entrada del flujo y verifica el estado de error. El estado de error interno se establece a verdadero cuando el flujo lanza una excepción del tipo `IOException` excepto la excepción `InterruptedIOException`, y cuando se invoca el método `setError()`. Regresa: `true` si y sólo sí el flujo lanza una excepción del tipo `IOException` excepto la excepción `InterruptedIOException`, y cuando se invoca el método `setError()`.
`print(boolean` *`b`***`)`** **`print(char`** *`c`***`)`** **`print(int`** *`i`***`)`** **`print(long`** *`l`***`)`** **`print(float`** *`f`***`)`** **`print(double`** *`d`***`)`** **`print(Object`** *`obj`***`)`** **`print(char[]`** *`s`***`)`** **`print(String`** *`s`***`)`** Estos métodos escriben al flujo de salida, las representaciones de un byte, un caracter, un entero, un entero largo, un flotante, un doble, un objeto, arreglo de caracteres o una cadena, respectivamente. Parámetro: La representación del valor a escribirse.
`println()` **`println(boolean`** *`b`***`)`** **`println(char`** *`c`***`)`** **`println(char[]`** *`s`***`)`** **`println(double`** *`d`***`)`** **`println(float`** *`f`***`)`** **`println(int`** *`i`***`)`** **`println(long`** *`l`***`)`** **`println(Object`** *`obj`***`)`** **`println(String`** *`s`***`)`** Estos métodos escriben al flujo de salida, un separador de línea, las representaciones de un byte, un caracter, un entero, un entero largo, un flotante, un doble, un objeto, arreglo de caracteres o una cadena, respectivamente, seguidas de un separador de línea. Parámetro: La representación del valor a escribirse.

Ejemplos de Archivos de Bytes de Acceso Secuencial

1. Se desea un programa que calcule como un capital inicial depositado se va acumulando para un cierto número de meses y un cierto número de tasas de interés, con recapitalización mensual. Los valores del capital inicial, número de meses, tasas de interés y los capitales calculados se almacenarán en un archivo secuencial.

   El programa consiste de dos clases: `Capitales` y `PruebaCapitales`. La clase `Capitales` contiene métodos para calcular los capitales acumulados para los diferentes meses y tasas, crear el archivo para guardar los resultados y escribir los resultados en el archivo. La clase `PruebaCapitales` sólo sirve para probar los métodos de la clase `Capitales`.

Capitales.java

```
/*
 * Capitales.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */

package capital;
import java.io.*;

/**
 * Esta clase calcula y almacena en un archivo los capitales acumulados
 * que produce un capital inicial depositado a diferentes tasas de interes
 * y con recapitalización mensual.
 *
 * @author mdomitsu
 */
public class Capitales {
  double capitalInicial;
  int meses;
  double tasas[];
  double capitales[][];
  String nomArchivo;

  /**
   * Este constructor inicializa los atributos de la clase e invoca
   * a los métodos que calcula los capitales y los guardan en un archivo
   * @param capitalInicial Capital inicial depositado
   * @param meses Número de meses depositados
   * @param tasas Arreglo con las diferentes tasas de interes a usarse en los
   * cálculos
   * @param nomArchivo Nombre del archvo en el que se almacenarán los
   * resultados
   */
  public Capitales(double capitalInicial, int meses, double tasas[],
                   String nomArchivo) {
    this.capitalInicial = capitalInicial;
    this.meses = meses;
```

```
    this.tasas = tasas;
    this.nomArchivo = nomArchivo;

    // Crea el arreglo en el que se almacenarán los capitales calculados
    capitales = new double[meses+1][tasas.length];

    // Calcula los capitales
    calculaCapitales();

    // Almacena en el arreglo los capitales
    guardaCapitales();
  }

  /**
   * Este método calcula y almacena en un arreglo los capitales acumulados
   * que produce un capital inicial depositado a diferentes tasas de interes
   * y con recapitalización mensual
   */
  public void calculaCapitales() {
    // Inicializa los capitales del mes 0 al valor del capital inicial
    for(int j = 0; j < tasas.length; j++) capitales[0][j] = capitalInicial;

    // Para cada uno de los meses
    for(int i = 1; i <= meses; i++)
      // Para cada una de las tasas de interes
      for(int j = 0; j < tasas.length; j++) {
        // Calcula el capital acumulado a partir del capital del mes anterior
        capitales[i][j] = capitales[i-1][j] * (1 + tasas[j]/12);
      }
  }

  /**
   * Almacena los capitales en un archivo
   */
  public void guardaCapitales() {
    FileOutputStream fileOutputStream = null;
    DataOutputStream dataOutputStream;

    // Abre el archivo donde se escribirán los resultados
    try {
      fileOutputStream = new FileOutputStream(nomArchivo);
    }
    catch (IOException ioe) {
      System.out.println("Error, no se pudo crear el archivo" +
                         nomArchivo);
      return;
    }

    // Se envuelve el archivo donde se escribirán los resultados con un
    // archivo del tipo DataOutputStream para escribir valores de datos de
    // tipos primitivos
    dataOutputStream = new DataOutputStream(fileOutputStream);

    try {
      // Escribe el capital inicial
      dataOutputStream.writeDouble(capitalInicial);
```

```
      // Escribe el número de meses
      dataOutputStream.writeInt(meses);

      // Escribe el número de tasas de interés
      dataOutputStream.writeInt(tasas.length);

      // Escribe las tasas de interés
      for(int i = 0; i < tasas.length; i++)
        dataOutputStream.writeDouble(tasas[i]);

      // Para cada mes
      for(int i = 1; i <= meses; i++)
        // Para cada tasa de interes
        for (int j = 0; j < tasas.length; j++)
          // Escribe el capital acumulado
          dataOutputStream.writeDouble(capitales[i][j]);
    }
    catch(IOException ioe) {
      System.out.println("Error al escribir al archivo" + nomArchivo);
      return;
    }

    // Cierra el archivo
    try {
      dataOutputStream.close();
      fileOutputStream.close();
    }
    catch (IOException ioe) {
      System.out.println("Error al cerrar el archivo" + nomArchivo);
      return;
    }
  }

  public void tabulaCapitales() {
    System.out.println("           Tasas");
    System.out.print("Meses ");
    for(int j = 0; j < tasas.length; j++)
      System.out.print("      " + tasas[j]);
    System.out.println();

    for(int i = 1; i <= meses; i++) {
      System.out.print("   " + i);
      for (int j = 0; j < tasas.length; j++) {
        System.out.print("  " + capitales[i][j]);
      }
      System.out.println();
    }
  }
}
```

PruebaCapitales.java

```
/*
 * PruebaCapitales.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */
```

```
package capital;

/**
 * Esta clase sirve para probar la clase Capitales que calcula y almacena
 * en un archivo los capitales acumulados que produce un capital inicial
 * depositado a diferentes tasas de interes y con recapitalización mensual.
 *
 * @author mdomitsu
 */
public class PruebaCapitales {
  /**
   * Método main(). Invoca a los métodos de la clase Capitales
   * @param args Argumentos en la línea de comando
   */
  public static void main(String[] args) {
    PruebaCapitales pruebaCapitales1 = new PruebaCapitales();

    // Arreglo con las tasas de interés a usar
    double tasas[] = {0.1, 0.15, 0.2};

    // Crea un objeto del tipo Capitales y le pasa el capital inicial,
    // el número de meses a tabular, el arreglo con las tasas de interés
    // y el nombre del archivo en el que se guardarán los resultados
    Capitales capitales = new Capitales(100.0, 20, tasas, "capitales.dat");
  }
}
```

2. Se desea un programa que lea los capitales almacenados en el archivo del ejemplo anterior y los tabule.

   El programa consiste de dos clases: `TabulaCapitales` y `PruebaTabulaCapitales`. La clase `TabulaCapitales` contiene métodos para leer los capitales acumulados para los diferentes meses y tasas del archivo y para tabular los resultados. La clase `PruebaTabulaCapitales` sólo sirve para probar los métodos de la clase `TabulaCapitales`.

TabulaCapitales.java

```
/*
 * Tabula**Capitales.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */

package capital;

import java.io.*;
import java.text.DecimalFormat;

/**
 * Esta clase lee de un archivo los capitales acumulados que produce
 * un capital inicial depositado a diferentes tasas de interes
 * y con recapitalización mensual y los tabula.
 *
```

```
 * @author mdomitsu
 */
public class TabulaCapitales {
  String nomArchivo;
  double capitalInicial;
  int meses;
  double tasas[];
  double capitales[][];

  /**
   * Este constructor inicializa los atributos de la clase e invoca
   * a los métodos que lee los capitales y los tabula
   * @param nomArchivo Nombre del archivo con los capitales
   */
  public TabulaCapitales(String nomArchivo) {
    this.nomArchivo = nomArchivo;

    // lee los capitales del archivo
    leeCapitales();

    // Tabula los capitales
    tabulaCapitales();
  }

  /**
   * Este método lee los capitales de un archivo
   */
  public void leeCapitales() {
    FileInputStream fileInputStream = null;
    DataInputStream dataInputStream;
    int numTasas;

    // Abre el archivo con los capitales
    try {
      fileInputStream = new FileInputStream(nomArchivo);
    }
    catch (FileNotFoundException fnfe) {
      System.out.println("Error: No existe el archivo " + nomArchivo);
      return;
    }

    // Se envuelve el archivo con los capitales con un archivo del tipo
    // DataInputStream para leer valores de tipos de datos primitivos
    dataInputStream = new DataInputStream(fileInputStream);

    try {
      // Lee el capital inicial
      capitalInicial = dataInputStream.readDouble();

      // Lee el número de meses
      meses = dataInputStream.readInt();

      // Lee el número de tasas de interés
      numTasas = dataInputStream.readInt();

      // Crea el arreglo para almacenar las tasas de interes
      tasas = new double[numTasas];
```

```
      // Crea el arreglo para almacenar los capitales
      capitales = new double[meses][numTasas];

      // Lee las tasas de interés
      for(int i = 0; i < numTasas; i++)
        tasas[i] = dataInputStream.readDouble();

      // Lee los capitales
      // Para cada mes
      for(int i = 0; i < meses; i++)
        // Para cada tasa de interes
        for (int j = 0; j < tasas.length; j++)
          // Lee el capital acumulado
          capitales[i][j] = dataInputStream.readDouble();
    }
    catch(IOException ioe) {
      System.out.println("Error al leer del archivo" + nomArchivo);
      return;
    }

    // Cierra el archivo
    try {
      dataInputStream.close();
      fileInputStream.close();
    }
    catch (IOException ioe) {
      System.out.println("Error al cerrar el archivo" + nomArchivo);
      return;
    }
  }

  /**
   * Este méodo tabula los capitales
   */
  public void tabulaCapitales() {
    // Formate para escribir los meses
    DecimalFormat tresDigitos = new DecimalFormat("000");

    // Formato para escribir los capitales
    DecimalFormat dosDecimales = new DecimalFormat("#,##0.00");

    // Escribe el encabezado de la tabla
    System.out.println("                      Tasas");
    System.out.print("Meses");
    for(int j = 0; j < tasas.length; j++)
      System.out.print("       " + dosDecimales.format(tasas[j]));
    System.out.println();

    // Escribe los renglones de la tabla
    for(int i = 0; i < meses; i++) {
      // Escribe el mes
      System.out.print("  " + tresDigitos.format(i+1));

      // Escribe los capitales de ese mes
      for (int j = 0; j < tasas.length; j++) {
        System.out.print("     " + dosDecimales.format(capitales[i][j]));
```

```
      }
      System.out.println();
    }
  }
}
```

PruebaTabulaCapitales.java

```
/*
 * Prueba**Tabula**Capitales.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */

package capital;

/**
 * Esta clase sirve para probar la clase TabulaCapitales lee
 * de un archivo los capitales acumulados que produce un capital inicial
 * depositado a diferentes tasas de interes y con recapitalización mensual
 * y los tabula
 *
 * @author Manuel Domitsu Kono
 */
public class PruebaTabulaCapitales {

  /**
   * Método main(). Invoca a los métodos de la clase TabulaCapitales
   * @param args Argumentos en la línea de comando
   */
  public static void main(String[] args) {
    PruebaTabulaCapitales pruebaTabulaCapitales1 =
                                          new PruebaTabulaCapitales();

    // Crea un objeto del tipo TabulaCapitales y le pasa el nombre del
    // archivo en el que están guardados los resultados
    TabulaCapitales tabulaCapitales = new TabulaCapitales("capitales.dat");
  }
}
```

La corrida del programa anterior se muestra a continuación:

```
                        Tasas
      Meses       0.10       0.15       0.20
        001     100.83     101.25     101.67
        002     101.67     102.52     103.36
        003     102.52     103.80     105.08
        004     103.38     105.09     106.84
        005     104.24     106.41     108.62
      ...
```

Archivos de Acceso Aleatorio

En este caso, accederemos a los datos del archivo en forma aleatorial y los datos del archivo serán interpretados por el programa como bytes. Por acceso aleatorio

entendemos que podemos posicionarnos en cualquier lugar del archivo en forma directa sin tener que leer y descartar los bytes anteriores del archivo. En la figura 10.3 se muestra un diagrama parcial de las interfaces y clases que podemos utilizar en este tipo de problemas.

- Si el archivo se crea en el modo de lectura/escritura, las operaciones de escritura también están disponibles; las operaciones de escritura escriben bytes a partir del apuntador de archivo y adelantan ese apuntador de archivo para que apunten al siguiente byte después del último byte escrito. Las operaciones que escriben más allá del fin actual del archivo extendiendo el tamaño del archivo. El valor del apuntador de archivo puede leerse mediante el método `getFilePointer()` y establecerse mediante el método `seek()`.

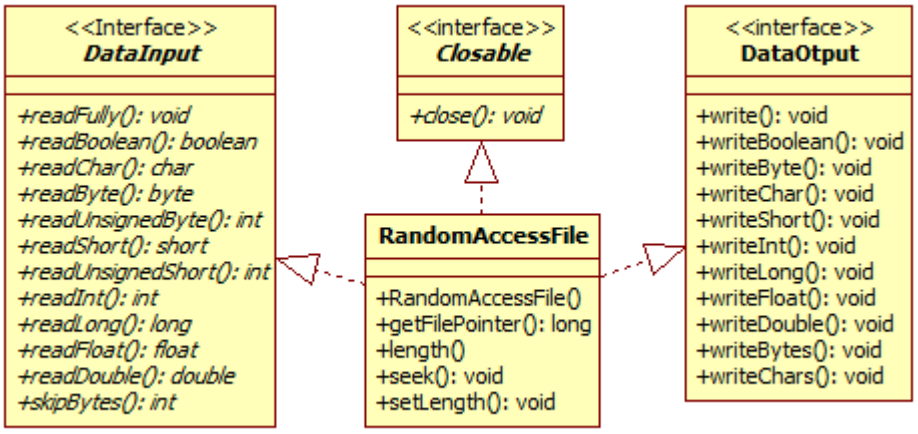


Figura 10.5 Interfaces y Clases para Archivos de Acceso Aleatorio

Los métodos de esta clase se muestran en la tabla 10.32.

Tabla 10.32 Métodos de la clase `RandomAccessFile`.

`RandomAccessFile(String` *`fileName`*`, String` *`mode`*`) throws FileNotFoundException`
Crea un flujo de acceso aleatorio para el archivo de nombre *`fileName`* y modo de acceso *`mode`*. Los valores permitidos para el método de acceso son: o `"r"` – Abre el archivo para lectura solamente. o `"rw"` – Abre el archivo para lectura y escritura. Si el archivo no existe se intenta crear uno. **Lanza**: `FileNotFoundException` – Si el modo de apertura es "r" y el nombre de archivo no corresponde a un archivo regular. Si el modo de apertura es `"rw"` y el nombre de archivo no corresponde a un archivo escribible existente y no puede crearse uno nuevo o si ocurre un error de entrada/salida. `IllegalArgumentException` – Si el modo de acceso no corresponde a ningulo de los modos válidos.

Tabla 10.32 Métodos de la clase `RandomAccessFile`.

`long getFilePointer()throws IOException` Regresa el valor actual del apuntador de archivo. El desplazamiento con respecto al inicio del archivo en bytes de la posición en la que ocurrirá la siguiente lectura o escritura. Lanza: `IOException` - Si ocurre un error de entrada/salida.
`long length()throws IOException` Regresa el tamaño de este archivo en bytes. Lanza: `IOException` - Si ocurre un error de entrada/salida.
`void seek(long` *`pos`*`) IOException` Establece el desplazamiento *`pos`* con respecto al inicio del archivo en bytes de la posición en la que ocurrirá la siguiente lectura o escritura. El desplazamiento puede estar más allá del final del archivo. En este caso el tamaño del archivo no cambia hasta que ocurra una escritura en esta posición. Lanza: `IOException` - Si *`pos`* es menor que cero u ocurre un error de entrada/salida.
`void setLeght(long length) IOException` Establece el tamaño del archivo al valor de **`length`**. Si el tamaño actual del archivo, como lo reporta el método `lengh()`, es mayor que el valor deseado el archivo será truncado. En este caso, si el apuntador de archivo es mayor que el nuevo tamaño del archivo, el valor del apuntador de archivo será igual al tamaño del archivo después del truncamiento. Si el tamaño actual del archivo, como lo reporta el método `lengh()`, es menor que el valor deseado el archivo será extendido. en este caso, el contenido de la porción extendida no está definida. Lanza: `IOException` - Si ocurre un error de entrada/salida.

Ejemplo de Archivos de Acceso Aleatorio

En el Tema 6: Colecciones se implementó un mecanismo de persistencia basado en arreglos para almacenar los datos de las canciones y películas del programa sobre el amante de la música y el cine. Sin embargo, este mecanismo no es en realidad un mecanismo de persistencia ya que los datos se almacenan en arreglos y estos datos se pierden en cuanto el programa termina su ejecución. En este ejemplo implantaremos un verdadero mecanismo de persistencia basado en archivos. Tendremos tres archivos de acceso aleatorio llamados **canciones.dat, peliculas.dat** y **generos.dat** para almacenar los datos de las canciones, películas y géneros. Los métodos para agregar, actualizar y borrar canciones, películas y géneros, así como para realizar consultas estarán en las clases `Canciones`,`Peliculas` y `Generos`.

Cada uno de los registros de los archivos **canciones.dat, peliculas.dat** y **generos.dat** contendrá los datos de una canción, de una película o de un género. A fin de poder

acceder a los registros de las canciones o de las películas en forma aleatoria, es necesario que todos los registros de las canciones sean del mismo tamaño y lo mismo debemos tener con los registros de las películas y de los géneros. Por tal motivo los datos a almacenar que sean cadenas deberán convertirse a arreglos de tamaño fijo al momento de almacenarse y reconvertirse a cadenas al momento de recuperarse de los archivos. Los métodos que nos permiten guardar y recuperar cadenas en un archivo como arreglos de tamaño fijo, así como los métodos que nos permiten guardar y recuperar fechas y borrar registros se implementan en la clase `AccesoAleatorio` la cual será la superclase de las clases `Canciones`, `Peliculas` y `Generos`. El diagrama de clases de las clases `AccesoAleatorio`, `Canciones`, `Peliculas` y `Generos` se muestra en la figura 10.4.

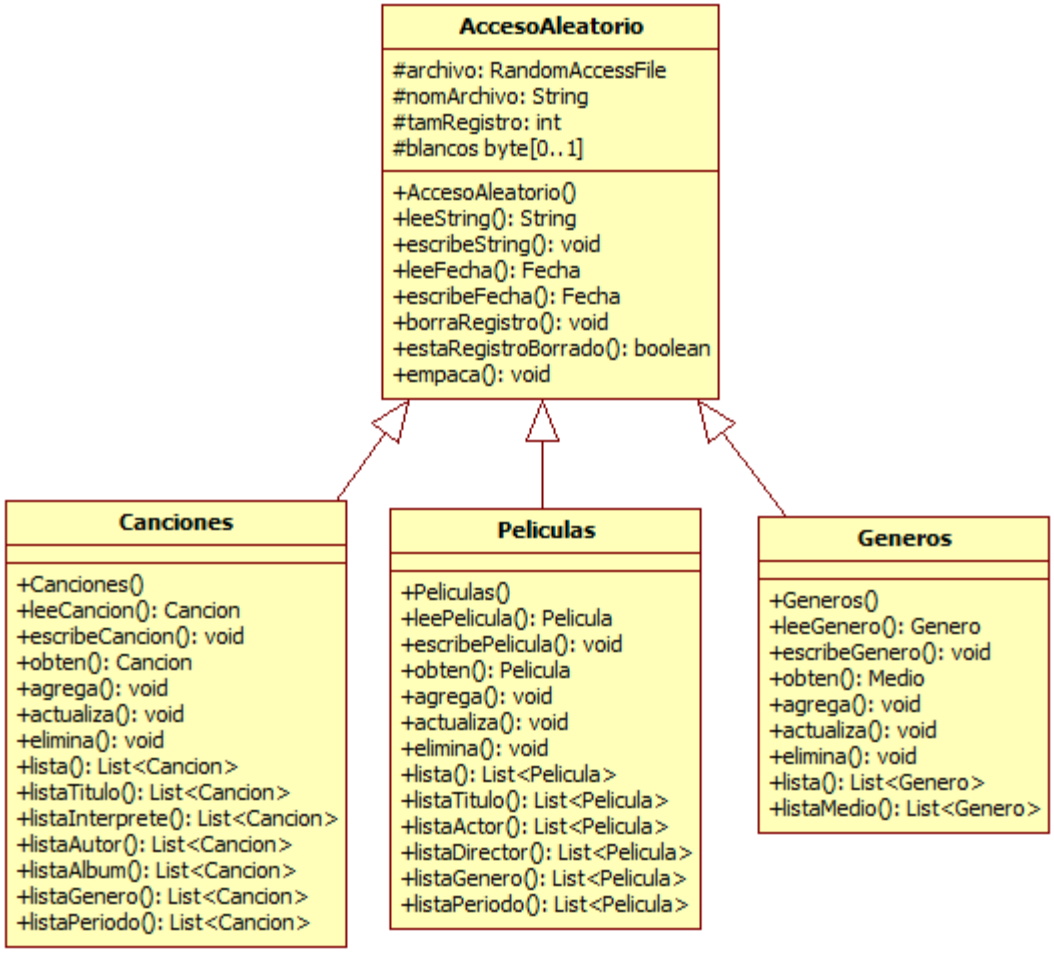


Figura 10.6 Diagrama de Clases del Mecanismo de Persistencia Basado en Archivos del Programa Amante Música.

El listado de la clase `AccesoAleatorio` es el siguiente:

AccesoAleatorio.java

```
/*
 * AccesoAleatorio.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */

package dao;

import java.io.*;

import objetosServicio.*;

/**
 * Esta clase contiene rutinas para trabajar con archivos de acceso aleatorio
 *
 * @author mdomitsu
 */
public class AccesoAleatorio {
  protected RandomAccessFile archivo;
  protected String nomArchivo;
  protected int tamRegistro;
  protected byte blancos[];

  /**
   * Constructor de la clase. Establece el nombre del archivo, el tamaño de
   * cada uno de los registros del archivo y crea un arreglo de bytes en
   * ceros para borrar un registro
   * @param nomArchivo Nombre del archivo
   * @param tamRegistro Tamaño del registro
   */
  public AccesoAleatorio(String nomArchivo, int tamRegistro) {
    this.nomArchivo = nomArchivo;
    this.tamRegistro = tamRegistro;

    // Crea un arreglo de bytes en ceros
    blancos = new byte[tamRegistro];
    for(int i = 0; i < tamRegistro; i++) blancos[i] = 0;
  }

  /**
   * Lee una secuencia de caracteres de un archivo de acceso aleatorio y los
   * regresa en un String
   * @param tam Número de caracteres a leer del archivo
   * @return Un String con los caracteres leídos
   * @throws IOException Si hay un error de entrada o salida.
   */
  public String leeString(int tam) throws IOException {
    char cadena[] = new char[tam];

    // Lee tam caracteres del archivo y los almacena en un arreglo
    for(int i = 0; i < tam; i++) cadena[i] = archivo.readChar();

    // Convierte el arreglo en un String
    String sCadena = new String(cadena);
```

```
    // Reemplaza los caracteres '\u0000' por espacios
    sCadena = sCadena.replace('\u0000', ' ');

    // Elimina los espacios en blanco
    sCadena = sCadena.trim();

    return sCadena;
  }

  /**
   * Escribe una secuencia de caracteres en un archivo. El número de
   * caracteres a escribir está dado por tam y los caracteres están en
   * sCadena.
   * @param sCadena String con los caracteres a escribir
   * @param tam Número de caracteres a escribir.
   * @throws IOException Si hay un error de entrada o salida.
   */
  public void escribeString(String sCadena, int tam) throws IOException {
    StringBuffer cadena;

    if(sCadena != null) {
      // Crea un StringBuffer a partir de la cadena
      cadena = new StringBuffer(sCadena);
    }
    else
      // Crea una cadena vacía de tamaño tam
      cadena = new StringBuffer(tam);

    // Hace el StringBuffer de tamaño tam
    cadena.setLength(tam);

    // Convierte el stringbuffer a una cadena
    sCadena = cadena.toString();

    // escribe la cadena al archivo
    archivo.writeChars(sCadena);
  }

  /**
   * Lee una fecha de un archivo como tres enteros: día, mes, año
   * @return La fecha leída
   * @throws IOException Si hay un error de entrada o salida.
   */
  public Fecha leeFecha() throws IOException {
    // Lee el día del archivo
    int dia = archivo.readInt();

    // Lee el mes del archivo
    int mes = archivo.readInt();

    // Lee año del archivo
    int anho = archivo.readInt();

    // Crea una fecha a partir del día, mes y año
    Fecha fecha = new Fecha(dia, mes, anho);

    return fecha;
```

```
  }

  /**
   * Escribe una fecha a un archivo como tres enteros: día, mes, año
   * @param fecha Fecha a escribir
   * @throws IOException Si hay un error de entrada o salida.
   */
  public void escribeFecha(Fecha fecha) throws IOException {
    if(fecha != null) {
      // Escribe el día
      archivo.writeInt(fecha.getDia());

      // Escribe el mes
      archivo.writeInt(fecha.getMes());

      // Escribe el año
      archivo.writeInt(fecha.getAnho());
    }
    else {
      archivo.writeInt(0);
      archivo.writeInt(0);
      archivo.writeInt(0);
    }
  }

  /**
   * Escribe un registro con tamRegistro bytes en 0 en el archivo
   * @throws IOException Si hay un error de entrada o salida.
   */
  public void borraRegistro() throws IOException {
    // Escribe en el archivo tamRegistro bytes en 0
    archivo.write(blancos);
  }

  /**
   * Regresa true si el arreglo de bytes dado por registro contiene puros
   * ceros
   * @param registro Arreglo a probar
   * @return true si el arreglo contiene puros ceros, false en caso
   * contrario.
   */
  public boolean estaRegistroBorrado(byte registro[]) {
    // regresa false al primer byte diferente de ceros
    for(int i = 0; i < tamRegistro; i++)
      if(registro[i] != 0) return false;

    // Si son puros ceros regresa true
    return true;
  }

  /**
   * Este método elimina fisicamente los registros borrados de un archivo
   * @throws IOException
   */
  public void empaca() throws IOException {
    byte registro[] = new byte[tamRegistro];
    int registrosBorrados = 0;
```

```
    // Calcula el número de registros en el archivo
    int numRegistros = (int)archivo.length()/tamRegistro;

    // Para cada registro del archivo
    for(int i=0; i < numRegistros; i++) {
      // Posiciona en el i-esimo registro del archivo
      archivo.seek(i * tamRegistro);
      // lee el registro
      archivo.read(registro);

      // Si el registro está borrado
      if(estaRegistroBorrado(registro)) {
        // Recorre todas los registros una posicion hacia
        // arriba para recuperar el espacio
        for(int j=i; j < numRegistros-1; j++) {
          // Posiciona en el j-esimo + 1 registro del archivo
          archivo.seek( (j + 1) * tamRegistro);
          // lee el registro
          archivo.read(registro);

          // Posiciona en el j-esimo lugar del archivo
          archivo.seek(j * tamRegistro);
          // escribe el registro
          archivo.write(registro);
        }
        // Decrementa el número de registros al eliminar un registro
        numRegistros--;
        // Contabiliza el número de registros eliminados
        registrosBorrados++;
      }
    }

    // Trunca el archivo para eliminar el espacio liberado por los registros
    // eliminados
    archivo.setLength(archivo.length() - registrosBorrados * tamRegistro);
  }
}
```

El siguiente es el listado parcial de la clase `Canciones` que implementa los métodos que nos permiten obtener, consultar, insertar, actualizar y eliminar registros en el archivo **canciones.dat**. Cada registro proviene de o se convierte en un objeto del tipo `Cancion`. Los métodos de esta clase lanzan excepciones del tipo `DAOException`, la cual fue vista en el Tema 6: Colecciones.

Canciones.java

```
/*
 * Canciones.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */

package dao;

import java.io.*;
```

```
import java.util.List;
import java.util.ArrayList;

import objetosServicio.*;
import objetosNegocio.*;
import excepciones.DAOException;

/**
 * Esta clase permite agregar, actualizar, eliminar y consultar canciones
 * del programa AmanteMusica en su versión que usa archivos
 *
 * @author mdomitsu
 */
public class Canciones extends AccesoAleatorio {

  // Tamaño de un registro (datos de una canción)
  //   clave        7 caracteres 14 bytes
  //   titulo      35 caracteres 70 bytes
  //   cveGenero    7 caracteres 14 bytes
  //   interprete  35 caracteres 70 bytes
  //   autor       35 caracteres 70 bytes
  //   album       35 caracteres 70 bytes
  //   duracion    int            4 bytes
  //   fecha       3 int         12 bytes
  //  Total                     324 bytes

  public Canciones(String nomArchivo) {
    super(nomArchivo, 324);
  }

  /**
   * Lee una canción de un archivo
   * @return La canción leída
   * @throws IOException Si hay un error de entrada / salida.
   */
  private Cancion leeCancion() throws IOException {
    Cancion cancion = new Cancion();

    // Lee de el archivo cada unos de los atributos de la canción
    cancion.setClave(leeString(7));
    cancion.setTitulo(leeString(35));
    Genero genero = new Genero(leeString(7));
    cancion.setGenero(genero);
    cancion.setInterprete(leeString(35));
    cancion.setAutor(leeString(35));
    cancion.setAlbum(leeString(35));
    cancion.setDuracion(archivo.readInt());
    cancion.setFecha(leeFecha());

    return cancion;
  }

  /**
   * Escribe una canción a un archivo
   * @param cancion Canción a escribir
   * @throws IOException Si hay un error de entrada / salida.
   */
```

```
  private void escribeCancion(Cancion cancion) throws IOException {
   escribeString(cancion.getClave(), 7);
   escribeString(cancion.getTitulo(), 35);
   escribeString(cancion.getGenero().getCveGenero(), 7);
   escribeString(cancion.getInterprete(), 35);
   escribeString(cancion.getAutor(), 35);
   escribeString(cancion.getAlbum(), 35);
   archivo.writeInt(cancion.getDuracion());
   escribeFecha(cancion.getFecha());
  }

  /**
   * Regresa la cancion del arreglo que coincida con la cancion del
   * parametro.
   * Las claves de las canciones del arreglo y del parametro deben coincidir
   * @param cancion Objeto de tipo Cancion con la clave de la canción a
   * buscar
   * @return La Cancion si la encuentra. null en caso contrario.
   * @throws DAOException Si hay un error de entrada / salida
   * o el archivo no existe.
   */
  public Cancion obten(Cancion cancion) throws DAOException {
    Cancion cancionLeida;

    // Abre el archivo de sólo lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya canciones en el archivo
      while(true) {
        // Lee una canción
          cancionLeida = leeCancion();
        // Si es la canción buscada, regrésala
        if(cancion.equals(cancionLeida)) {
          return cancionLeida;
        }
      }
    }
    // Si se llegó al final del archivo sin encontrar la canción
    catch (EOFException eofe) {
      return null;
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
```

```
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite agregar una canción a un archivo.
   * @param cancion Cancion a agregar en la tabla canciones
   * @throws DAOException Si hay un error de entrada / salida,
   * el archivo no existe.
   */
  public void agrega(Cancion cancion) throws DAOException {
    // Abre el archivo de escritura/lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Se posiciona al final del archivo
      archivo.seek(archivo.length());

      // Escribe la canción
      escribeCancion(cancion);
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Actualiza la canción del archivo que coincida con la cancion del
   * parametro.
   * Las claves de las canciones del archivo y del parametro deben coincidir
   * @param cancion La canción a modificar
   * @throws DAOException Si hay un error de entrada / salida,
   * el archivo no existe o no se puede actualizar la canción.
   */
  public void actualiza(Cancion cancion) throws DAOException {
    Cancion cancionLeida;

    // Abre el archivo de escritura/lectura
    try {
```

```
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya canciones en el archivo
      while(true) {
        // Lee una canción
        cancionLeida = leeCancion();

        // Si es la canción buscada
        if(cancion.equals(cancionLeida)) {
          // Se posiciona al principio del registro
          archivo.seek(archivo.getFilePointer() - tamRegistro);

          // Escribe la canción modificada
          escribeCancion(cancion);

          // Termina la búsqueda
          break;
        }
      }
    }
    // Si se llegó al final del archivo
    catch (EOFException eofe) {
      throw new DAOException("La canción no existe");
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Elimina la canción del archivo que coincida con la cancion del
   * parametro.
   * Las claves de las canciones del archivo y del parametro deben coincidir
   * Este método permite borrar una canción del archivo canciones
   * @param cancion Cancion a borrar
   * @throws DAOException Si hay un error de entrada / salida,
   * el archivo no existe o no se puede eliminar la canción.
   */
  public void elimina(Cancion cancion) throws DAOException {
    Cancion cancionLeida;
```

```
    // Abre el archivo de escritura/lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya canciones en el archivo
      while(true) {
        // Lee una canción
        cancionLeida = leeCancion();

        // Si es la canción buscada
        if(cancion.equals(cancionLeida)) {
          // Se posiciona al principio del registro
          archivo.seek(archivo.getFilePointer() - tamRegistro);

          // Escribe un registro en blanco y empaca
          borraRegistro();
          empaca();

          // Termina la búsqueda
          break;
        }
      }
    }
    // Si se llegó al final del archivo
    catch (EOFException eofe) {
      throw new DAOException("La canción no existe");
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite consultar las canciones del archivo canciones.
   * @return Una lista de las canciones del archivo canciones
   * @throws DAOException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public List<Cancion> lista() throws DAOException {
    List<Cancion> lista = new ArrayList<Cancion>();
    Cancion cancion;
```

```
    // Abre el archivo de sólo lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya canciones en el archivo
      while (true) {
        // Lee una canción
        cancion = leeCancion();

        // Agrega la canción a la lista de canciones
        lista.add(cancion);
      }
    }
    // Si se llegó al final del archivo
    catch (EOFException eofe) {
      // Regresa la lista de canciones
      return lista;
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite consultar las canciones del archivo canciones
   * y que tienen el mismo título.
   * @param titulo Título de la canción a buscar
   * @return Una lista de las canciones del archivo canciones con el mismmo
   * titulo
   * @throws DAOException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public List<Cancion> listaTitulo(String titulo)
    throws DAOException {
    List<Cancion> lista = new ArrayList<Cancion>();
     Cancion cancion;

     // Abre el archivo de sólo lectura
     try {
       archivo = new RandomAccessFile(nomArchivo, "r");
```

```
      }
      catch(FileNotFoundException fnfe) {
        throw new DAOException("Archivo inexistente");
      }

      try {
        // Mientras haya canciones en el archivo
        while(true) {
          // Lee una canción
          cancion = leeCancion();

          // Si es la canción buscada
          if(titulo.equals(cancion.getTitulo()))
            // Agrega la canción a la lista de canciones
            lista.add(cancion);
        }
      }
      // Si se llegó al final del archivo
      catch (EOFException eofe) {
        // Regresa la lista de canciones
        return lista;
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al acceder al archivo");
      }
      finally {
        try {
          // Cierra el archivo
          archivo.close();
        }
        // Si ocurrió un error de entrada salida
        catch (IOException eofe) {
          throw new DAOException("Error al cerrar el archivo");
        }
      }
  }

...

  /**
   * Este método permite consultar las canciones del archivo canciones
   * y que tienen el mismo genero.
   * @param cveGenero Clave del género de la canción a buscar
   * @return Una lista de las canciones del archivo canciones del mismo
   * genero
   * @throws DAOException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public List<Cancion> listaGenero(String cveGenero)
    throws DAOException {
    List<Cancion> lista = new ArrayList<Cancion>();
     Cancion cancion;

     // Abre el archivo de sólo lectura
     try {
       archivo = new RandomAccessFile(nomArchivo, "r");
```

```
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya canciones en el archivo
      while(true) {
        // Lee una canción
        cancion = leeCancion();

        // Si es la canción buscada
        if(cveGenero.equals(cancion.getGenero().getCveGenero()))
          // Agrega la canción a la lista de canciones
          lista.add(cancion);
      }
    }
    // Si se llegó al final del archivo
    catch (EOFException eofe) {
      // Regresa la lista de canciones
      return lista;
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite consultar las canciones del archivo canciones
   * y que tienen el mismo periodo.
   * @param periodo Periodo de la canción a buscar
   * @return Una lista de las canciones del archivo canciones del mismo
   * periodo
   * @throws DAOException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public List<Cancion> listaPeriodo(Periodo periodo)
    throws DAOException {
    List<Cancion> lista = new ArrayList<Cancion>();
     Cancion cancion;

    // Abre el archivo de sólo lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
```

```
        throw new DAOException("Archivo inexistente");
      }

      try {
        // Mientras haya canciones en el archivo
        while(true) {
          // Lee una canción
          cancion = leeCancion();

          // Si es la canción buscada
          if(periodo.contiene(cancion.getFecha()))
            // Agrega la canción a la lista de canciones
            lista.add(cancion);
        }
      }
      // Si se llegó al final del archivo
      catch (EOFException eofe) {
        // Regresa la lista de canciones
        return lista;
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al acceder al archivo");
      }
      finally {
        try {
          // Cierra el archivo
          archivo.close();
        }
        // Si ocurrió un error de entrada salida
        catch (IOException eofe) {
          throw new DAOException("Error al cerrar el archivo");
        }
      }
    }
}
```

El siguiente es el listado parcial de la clase `Peliculas` que implementa los métodos que nos permiten obtener, consultar, insertar, actualizar y eliminar registros en el archivo **peliculas.dat**. Cada registro proviene de o se convierte en un objeto del tipo **Pelicula**. Los métodos de esta clase lanzan excepciones del tipo `DAOException`, la cual fue vista en el Tema 6: Colecciones.

Peliculas

```
/*
 * Peliculas.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */

package dao;

import java.io.*;
import java.util.List;
import java.util.ArrayList;
```

```
import objetosServicio.*;
import objetosNegocio.*;
import excepciones.DAOException;

/**
 * Esta clase permite agregar, actualizar, eliminar y consultar películas
 * del programa AmanteMusica en su versión que usa archivos
 *
 * @author mdomitsu
 */
public class Peliculas extends AccesoAleatorio {

  // Tamaño de un registro (datos de una película)
  //   clave       7 caracteres 14 bytes
  //   titulo     35 caracteres 70 bytes
  //   cveGenero   7 caracteres 14 bytes
  //   actor1     35 caracteres 70 bytes
  //   actor2     35 caracteres 70 bytes
  //   director   35 caracteres 70 bytes
  //   duracion   int            4 bytes
  //   fecha      3 int         12 bytes
  //  Total                    324 bytes

  public Peliculas(String nomArchivo) {
    super(nomArchivo, 324);
  }

  /**
   * Lee una película de un archivo
   * @return La película leída
   * @throws IOException Si hay un error de entrada / salida.
   */
  private Pelicula leePelicula() throws IOException {
    Pelicula pelicula = new Pelicula();

    // Lee de el archivo cada unos de los atributos de la película
    pelicula.setClave(leeString(7));
    pelicula.setTitulo(leeString(35));
    pelicula.setGenero(new Genero(leeString(7)));
    pelicula.setActor1(leeString(35));
    pelicula.setActor2(leeString(35));
    pelicula.setDirector(leeString(35));
    pelicula.setDuracion(archivo.readInt());
    pelicula.setFecha(leeFecha());

    return pelicula;
  }

  /**
   * Escribe una película a un archivo
   * @param pelicula Canción a escribir
   * @throws IOException Si hay un error de entrada / salida.
   */
  private void escribePelicula(Pelicula pelicula) throws IOException {
   escribeString(pelicula.getClave(), 7);
   escribeString(pelicula.getTitulo(), 35);
```

```
    escribeString(pelicula.getGenero().getCveGenero(), 7);
    escribeString(pelicula.getActor1(), 35);
    escribeString(pelicula.getActor2(), 35);
    escribeString(pelicula.getDirector(), 35);
    archivo.writeInt(pelicula.getDuracion());
    escribeFecha(pelicula.getFecha());
  }

  /**
   * Regresa la pelicula del arreglo que coincida con la pelicula del
   * parametro.
   * Las claves de las peliculas del arreglo y del parametro deben coincidir
   * @param pelicula Objeto de tipo Pelicula con la clave de la película a
   * buscar
   * @return La Pelicula si la encuentra. null en caso contrario.
   * @throws DAOException Si hay un error de entrada / salida
   * o el archivo no existe.
   */
  public Pelicula obten(Pelicula pelicula) throws DAOException {
    Pelicula peliculaLeida;

    // Abre el archivo de sólo lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya películas en el archivo
      while(true) {
        // Lee una película
        peliculaLeida = leePelicula();

        // Si es la película buscada
        if(pelicula.equals(peliculaLeida)) {
          // Regresa la película buscada
          return peliculaLeida;
        }
      }
    }
    // Si se llegó al final del archivo sin encontrar la película
    catch (EOFException eofe) {
      return null;
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
```

```
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite agregar una película al archivo peliculas.
   * @param pelicula Pelicula a agregar en la tabla películas.
   * @throws DAOException Si hay un error de entrada / salida,
   * el archivo no existe.
   */
  public void agrega(Pelicula pelicula) throws DAOException {
    // Abre el archivo de escritura/lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Se posiciona al final del archivo
      archivo.seek(archivo.length());

      // Escribe la película
      escribePelicula(pelicula);
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Actualiza la película del archivo que coincida con la película del
   * parametro.
   * Las claves de las películas del archivo y del parametro deben coincidir
   * @param pelicula La película a modificar
   * @throws DAOException Si hay un error de entrada / salida,
   * el archivo no existe o no se puede actualizar la película.
   */
  public void actualiza(Pelicula pelicula) throws DAOException {
    Pelicula peliculaLeida;

    // Abre el archivo de lectura/escritura
    try {
      archivo = new RandomAccessFile(nomArchivo, "rw");
```

```
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya películas en el archivo
      while(true) {
        // Lee una película
        peliculaLeida = leePelicula();

        // Si es la película buscada
        if(pelicula.equals(peliculaLeida)) {
          // Se posiciona al principio del registro
          archivo.seek(archivo.getFilePointer() - tamRegistro);

          // Escribe la película modificada
          escribePelicula(pelicula);

          // Termina la búsqueda
          break;
        }
      }
    }
    // Si se llegó al final del archivo
    catch (EOFException eofe) {
      throw new DAOException("La película no existe");
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Eliminaa la película del archivo que coincida con la película del
   * parametro.
   * Las claves de las películas del archivo y del parametro deben coincidir
   * @param pelicula Pelicula a borrar
   * @throws DAOException Si hay un error de entrada / salida,
   * el archivo no existe o no se puede eliminar la película.
   */
  public void elimina(Pelicula pelicula) throws DAOException {
    Pelicula peliculaLeida;

    // Abre el archivo de escritura/lectura
    try {
```

```
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya películas en el archivo
      while(true) {
        // Lee una película
        peliculaLeida = leePelicula();

        // Si es la película buscada
        if(pelicula.equals(peliculaLeida)) {
          // Se posiciona al principio del registro
          archivo.seek(archivo.getFilePointer() - tamRegistro);

          // Escribe un registro en blanco y empaca
          borraRegistro();
          empaca();

          // Termina la búsqueda
          break;
        }
      }
    }
    // Si se llegó al final del archivo
    catch (EOFException eofe) {
      throw new DAOException("La película no existe");
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite consultar las películas del archivo películas.
   * @return Una lista de las películas del archivo películas
   * @throws DAOException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public List<Pelicula> lista() throws DAOException {
    List<Pelicula> lista = new ArrayList<Pelicula>();
    Pelicula pelicula;

    // Abre el archivo de sólo lectura
```

```
    try {
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya películas en el archivo
      while(true) {
        // Lee una película
        pelicula = leePelicula();

        // Agrega la película a la lista de películas
        lista.add(pelicula);
      }
    }
    // Si se llegó al final del archivo
    catch (EOFException eofe) {
      // Regresa la lista de películas
      return lista;
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite consultar las películas del archivo películas
   * y que tienen el mismo título.
   * @param titulo Título de la película a buscar
   * @return Una lista de las películas del archivo películas del mismo
   * título
   * @throws DAOException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public List<Pelicula> listaTitulo(String titulo)
    throws DAOException {
    List<Pelicula> lista = new ArrayList<Pelicula>();
     Pelicula pelicula;

     // Abre el archivo de sólo lectura
     try {
       archivo = new RandomAccessFile(nomArchivo, "r");
     }
     catch(FileNotFoundException fnfe) {
```

```
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya películas en el archivo
      while(true) {
        // Lee una película
        pelicula = leePelicula();

        // Si es la película buscada
        if(titulo.equals(pelicula.getTitulo()))
          // Agrega la película a la lista de películas
          lista.add(pelicula);
      }
    }
    // Si se llegó al final del archivo
    catch (EOFException eofe) {
      // Regresa la lista de películas
      return lista;
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite consultar las películas del archivo películas
   * y que tienen el mismo actor
   * @param actor Actor de la película a buscar
   * @return Una lista de las películas del archivo películas del mismo actor
   * @throws DAOException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public List<Pelicula> listaActor(String actor)
    throws DAOException {
    List<Pelicula> lista = new ArrayList<Pelicula>();
     Pelicula pelicula;

    // Abre el archivo de sólo lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }
```

```
      try {
        // Mientras haya películas en el archivo
        while(true) {
          // Lee una película
          pelicula = leePelicula();

          // Si es la película buscada
          if(actor.equals(pelicula.getActor1()) ||
             actor.equals(pelicula.getActor2()))
            // Agrega la película a la lista de películas
            lista.add(pelicula);
        }
      }
      // Si se llegó al final del archivo
      catch (EOFException eofe) {
        // Regresa la lista de películas
        return lista;
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al acceder al archivo");
      }
      finally {
        try {
          // Cierra el archivo
          archivo.close();
        }
        // Si ocurrió un error de entrada salida
        catch (IOException eofe) {
          throw new DAOException("Error al cerrar el archivo");
        }
      }
  }

...

}
```

El siguiente es el listado de la clase `Generos` que implementa los métodos que nos permiten obtener, consultar, insertar, actualizar y eliminar registros en el archivo **generos.dat**. Cada registro proviene de o se convierte en un objeto del tipo `Genero`. Los métodos de esta clase lanzan excepciones del tipo `DAOException`, la cual fue vista en el Tema 6: Colecciones.

Generos

```
/*
 * Generos.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */

package dao;

import java.io.EOFException;
import java.io.FileNotFoundException;
```

```
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.List;
import java.util.ArrayList;

import objetosNegocio.Genero;
import excepciones.DAOException;


/**
 * Esta clase permite agregar, actualizar, eliminar y consultar géneros de
 * generos o películas del programa AmanteMusica en su versión que usa
 * archivos.
 *
 * @author mdomitsu
 */
public class Generos extends AccesoAleatorio {

  // Tamaño de un registro (datos de una género)
  //   cveGenero    7 caracteres 14 bytes
  //   nombre      20 caracteres 40 bytes
  //   tipoMedio      Char        2 bytes
  //  Total                     56 bytes

  public Generos(String nomArchivo) {
    super(nomArchivo, 56);
  }

  /**
   * Lee un género de un archivo
   * @return El género leído
   * @throws IOException Si hay un error de entrada / salida.
   */
  private Genero leeGenero() throws IOException {
    Genero genero = new Genero();

    // Lee del archivo cada uno de los atributos del género
    genero.setCveGenero(leeString(7));
    genero.setNombre(leeString(20));
    genero.setTipoMedio(archivo.readChar());

    return genero;
  }

  /**
   * Escribe un género a un archivo
   * @param genero Género a escribir
   * @throws IOException Si hay un error de entrada / salida.
   */
  private void escribeGenero(Genero genero) throws IOException {
   escribeString(genero.getCveGenero(), 7);
   escribeString(genero.getNombre(), 20);
   archivo.writeChar(genero.getTipoMedio());
  }

  /**
   * Regresa el genero del archivo que coincida con el genero del parametro.
```

```
   * Las claves de los generos del archivo y del parametro deben coincidir
   * @param genero Objeto de tipo Genero con la clave del género a buscar
   * @return El Genero si lo encuentra. null en caso contrario.
   * @throws DAOException Si hay un error de entrada / salida
   * o el archivo no existe.
   */
  public Genero obten(Genero genero) throws DAOException {
    Genero generoLeido;

    // Abre el archivo de sólo lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya generos en el archivo
      while(true) {
        // Lee un género
          generoLeido = leeGenero();
        // Si es el género buscado, regrésalo
        if(genero.equals(generoLeido)) {
          return generoLeido;
        }
      }
    }
    // Si se llegó al final del archivo sin encontrar el género
    catch (EOFException eofe) {
      return null;
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite agregar un género a un archivo.
   * @param genero Género a agregar en el archivo géneros
   * @throws DAOException Si hay un error de entrada / salida,
   * el archivo no existe.
   */
  public void agrega(Genero genero) throws DAOException {
    // Abre el archivo de escritura/lectura
    try {
```

```
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Se posiciona al final del archivo
      archivo.seek(archivo.length());

      // Escribe el género
      escribeGenero(genero);
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Actualiza el genero del archivo que coincida con el genero del
   * parametro.
   * Las claves de los generos del archivo y del parametro deben coincidir
   * @param genero El género a modificar
   * @throws DAOException Si hay un error de entrada / salida,
   * el archivo no existe o no se puede actualizar el género.
   */
  public void actualiza(Genero genero) throws DAOException {
    Genero generoLeido;

    // Abre el archivo de escritura/lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya géneros en el archivo
      while(true) {
        // Lee una género
        generoLeido = leeGenero();

        // Si es el género buscado
        if(genero.getCveGenero().equals(generoLeido.getCveGenero())) {
          // Se posiciona al principio del registro
```

```
          archivo.seek(archivo.getFilePointer() - tamRegistro);

          // Escribe el género modificado
          escribeGenero(genero);

          // Termina la búsqueda
          break;
        }
      }
    }
    // Si se llegó al final del archivo
    catch (EOFException eofe) {
      throw new DAOException("El género no existe");
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Elimina el genero del archivo que coincida con el genero del parametro.
   * Las claves de los generos archivo y del parametro deben coincidir
   * @param genero Genero a borrar
   * @throws DAOException Si hay un error de entrada / salida,
   * el archivo no existe o no se puede eliminar el género.
   */
  public void elimina(Genero genero) throws DAOException {
    Genero generoLeido;

    // Abre el archivo de escritura/lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya generos en el archivo
      while(true) {
        // Lee un género
        generoLeido = leeGenero();

        // Si es el género buscado
        if(genero.getCveGenero().equals(generoLeido.getCveGenero())) {
          // Se posiciona al principio del registro
```

```
          archivo.seek(archivo.getFilePointer() - tamRegistro);

          // Escribe un registro en blanco y empaca
          borraRegistro();
          empaca();

          // Termina la búsqueda
          break;
        }
      }
    }
    // Si se llegó al final del archivo
    catch (EOFException eofe) {
      throw new DAOException("La género no existe");
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite consultar los géneros del archivo generos.
   * @return Una lista de los objetos del tipo Genero del archivo generos
   * @throws DAOException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public List<Genero> lista() throws DAOException {
    List<Genero> lista = new ArrayList<Genero>();
    Genero genero;

    // Abre el archivo de sólo lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
      throw new DAOException("Archivo inexistente");
    }

    try {
      // Mientras haya generos en el archivo
      while (true) {
        // Lee un género
        genero = leeGenero();

        // Agrega el género a la lista de géneros
        lista.add(genero);
```

```
      }
    }
    // Si se llegó al final del archivo
    catch (EOFException eofe) {
      // Regresa la lista de generos
      return lista;
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new DAOException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite consultar los géneros del archivo generos
   * que tienen el mismo tipo de medio.
   * @param nombre Tipo de medio a buscar
   * @return Una lista de géneros del archivo generos que
   * tienen el mismo tipo de medio.
   * @throws DAOException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public List<Genero> listaMedio(char tipoMedio)
    throws DAOException {
    List<Genero> lista = new ArrayList<Genero>();
     Genero genero;

     // Abre el archivo de sólo lectura
     try {
       archivo = new RandomAccessFile(nomArchivo, "r");
     }
     catch(FileNotFoundException fnfe) {
       throw new DAOException("Archivo inexistente");
     }

     try {
       // Mientras haya generos en el archivo
       while(true) {
         // Lee una género
         genero = leeGenero();

         // Si es el género buscado
         if(tipoMedio == genero.getTipoMedio())
           // Agrega el género a la lista de generos
           lista.add(genero);
       }
     }
```

```
      // Si se llegó al final del archivo
      catch (EOFException eofe) {
        // Regresa la lista de generos
        return lista;
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new DAOException("Error al acceder al archivo");
      }
      finally {
        try {
          // Cierra el archivo
          archivo.close();
        }
        // Si ocurrió un error de entrada salida
        catch (IOException eofe) {
          throw new DAOException("Error al cerrar el archivo");
        }
      }
  }
}
```

Como ya se mencionó en el Tema 6: Colecciones, por encima de las clases `Canciones, Peliculas` y `Generos` que nos permiten el acceso a los archivos **canciones.dat, peliculas.dat** y **generos.dat** , se tendrá una clase que actue de fachada de la capa de persistencia, `PersistenciaArchivos`. Para asegurarnos que podamos sustituir un mecanismo de persistencia por otro, la clase `PersistenciaArchivos` implementa la interfaz `IPersistencia`, como se muestra en la figura 10.7:

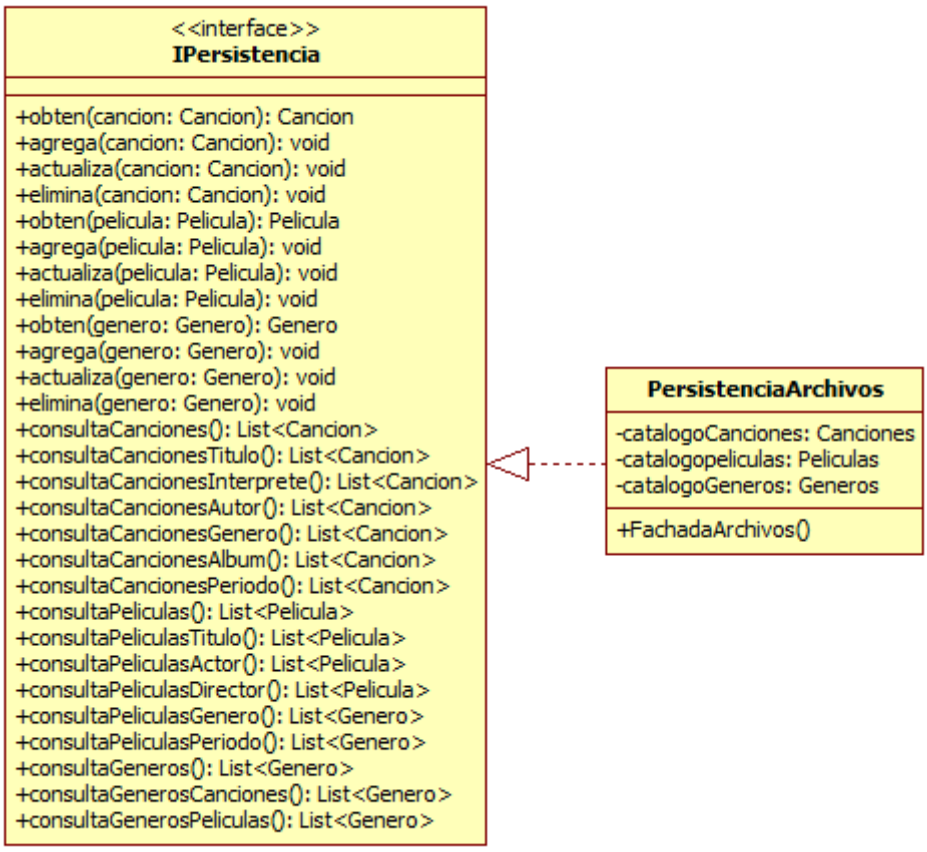


Figura 10.7. Fachada de la Capa de Persistencia del Programa AmanteMusica, Versión Archivos

Los métodos de la clase `PersistenciaArchivos` lanzan excepciones de la clase `PersistenciaException` vista en Tema 6: Colecciones: Excepciones. El código parcial de la clase `PersistenciaArchivos` es:

PersistenciaArchivos.java

```
/*
 * PersistenciaArchivos.java
 *
 * Created on 15 de septiembre de 2007, 12:21 PM
 *
 * @author mdomitsu
 */

package persistencia;

import java.util.List;

import objetosServicio.*;
import objetosNegocio.*;
import dao.*;
import interfaces.IPersistencia;
```

```
import excepciones.*;

/**
 * Esta clase implementa la interfaz IPersistencia del mecanismo de
 * persistencia de archivos del programa AmanteMusica
 */
public class PersistenciaArchivos implements IPersistencia {
  private Canciones catalogoCanciones;
  private Peliculas catalogoPeliculas;
  private Generos catalogoGeneros;

  /**
   * Constructor predeterminado
   */
  public PersistenciaArchivos() {
    // Crea un objeto del tipo catalogoCanciones para accesar al archivo
    // canciones
    catalogoCanciones = new Canciones("canciones.dat");

    // Crea un objeto del tipo catalogoPeliculas para accesar al archivo
    // peliculas
    catalogoPeliculas = new Peliculas("peliculas.dat");

    // Crea un objeto del tipo catalogoGeneros para accesar al archivo
    // géneros
    catalogoGeneros = new Generos("generos.dat");
  }

  /**
   * Obtiene una canción del catálogo de canciones
   * @param cancion Cancion a obtener
   * @return La canción si existe, null en caso contrario
   * @throws PersistenciaException Si no se puede obtener la canción.
   */
  public Cancion obten(Cancion cancion) throws PersistenciaException {
    Cancion cancionBuscada;

    // Obten la canción
    try {
      // Obten la canción
      cancionBuscada = catalogoCanciones.obten(cancion);
      if(cancionBuscada != null) {
          // Obten el género
          Genero genero = catalogoGeneros.obten(cancionBuscada.getGenero());
          // Agrégaselo a la canción
          cancionBuscada.setGenero(genero);
      }

      return cancionBuscada;
    }
    catch (DAOException pe) {
      throw new PersistenciaException("No se puede obtener la canción", pe);
    }
  }

  /**
   * Agrega una canción al catálogo de canciones. No se permiten canciones
```

```
   * con claves repetidas
   * @param cancion Cancion a agregar
   * @throws PersistenciaException Si la canción está repetida o no se puede
   * agregar la canción al catálogo de canciones.
   */
  public void agrega(Cancion cancion) throws PersistenciaException {
    Cancion cancionBuscada;

    // Busca la canción en el arreglo con la misma clave.
    try {
      cancionBuscada = catalogoCanciones.obten(cancion);

      // Si lo hay, no se agrega al arreglo
      if(cancionBuscada != null)
        throw new PersistenciaException("Canción repetida");
    }
    catch (DAOException pe) {
      // Si el archivo no existe no se hace nada
    }

    // Agrega la nueva canción al catálogo
    try {
      catalogoCanciones.agrega(cancion);
    }
    catch (DAOException pe) {
      throw new PersistenciaException("No se puede agregar la canción", pe);
    }
  }

  /**
   * Actualiza una canción del catálogo de canciones
   * @param cancion Cancion a actualizar
   * @throws PersistenciaException Si no se puede actualizar la canción del
   * catálogo de canciones.
   */
  public void actualiza(Cancion cancion) throws PersistenciaException {
    // Actualiza la  canción del catálogo
    try {
      catalogoCanciones.actualiza(cancion);
    }
    catch (DAOException pe) {
      throw new PersistenciaException("No se puede actualizar la canción",
          pe);
    }
  }

  /**
   * Elimina una canción del catálogo de canciones
   * @param cancion Cancion a eliminar
   * @throws PersistenciaException Si no se puede eliminar la canción del
   * catálogo de canciones.
   */
  public void elimina(Cancion cancion) throws PersistenciaException {
    // Elimina la  canción del catálogo
    try {
      catalogoCanciones.elimina(cancion);
    }
```

```
    catch (DAOException pe) {
      throw new PersistenciaException("No se puede eliminar la canción", pe);
    }
  }

  /**
   * Obtiene una película del catálogo de peliculas
   * @param pelicula Pelicula a obtener
   * @return La película si existe, null en caso contrario
   * @throws PersistenciaException Si no se puede obtener la película.
   */
  public Pelicula obten(Pelicula pelicula) throws PersistenciaException {
    Pelicula peliculaBuscada;

    try {
      // Obten la película
      peliculaBuscada = catalogoPeliculas.obten(pelicula);
      if(peliculaBuscada != null) {
          // Obten el género
          Genero genero = catalogoGeneros.obten(peliculaBuscada.getGenero());
          // Agrégaselo a la canción
          peliculaBuscada.setGenero(genero);
      }

      return peliculaBuscada;
    }
    catch (DAOException pe) {
      throw new PersistenciaException("No se puede obtener la película", pe);
    }
  }

  /**
   * Agrega una película al catálogo de peliculas. No se permiten peliculas
   * con claves repetidas
   * @param pelicula Película a agregar
   * @throws PersistenciaException Si la película está repetida o no se puede
   * agregar la película al catálogo de películas.
   */
  public void agrega(Pelicula pelicula) throws PersistenciaException {
    Pelicula peliculaBuscada;

    // Busca la canción en el arreglo con la misma clave.
    try {
      peliculaBuscada = catalogoPeliculas.obten(pelicula);

      // Si lo hay, no se agrega al arreglo
      if(peliculaBuscada != null)
        throw new PersistenciaException("Película repetida");
    }
    catch (DAOException pe) {
      // Si el archivo no existe no se hace nada
    }

    // Agrega la nueva película al catálogo
    try {
      catalogoPeliculas.agrega(pelicula);
    }
```

```
    catch (DAOException pe) {
      throw new PersistenciaException("No se puede agregar la película", pe);
    }
  }

  /**
   * Actualiza una película del catálogo de peliculas
   * @param pelicula Pelicula a actualizar
   * @throws PersistenciaException Si no se puede actualizar la película del
   * catálogo de películas.
   */
  public void actualiza(Pelicula pelicula) throws PersistenciaException {
    // Actualiza la  película del catálogo
    try {
      catalogoPeliculas.actualiza(pelicula);
    }
    catch (DAOException pe) {
      throw new PersistenciaException("No se puede actualizar la película",
        pe);
    }
  }

  /**
   * Elimina una película del catálogo de peliculas
   * @param pelicula Pelicula a eliminar
   * @throws PersistenciaException Si no se puede eliminar la película del
   * catálogo de películas.
   */
  public void elimina(Pelicula pelicula) throws PersistenciaException {
    // Elimina la  película del catálogo
    try {
      catalogoPeliculas.elimina(pelicula);
    }
    catch (DAOException pe) {
      throw new PersistenciaException("No se puede eliminar la película",
         pe);
    }
  }

  /**
   * Obtiene un género del catálogo de géneros
   * @param genero Género a obtener
   * @return El género si existe, null en caso contrario
   * @throws PersistenciaException Si no se puede obtener el género.
   */
  public Genero obten(Genero genero) throws PersistenciaException {
    // Obten el género
    try {
      return catalogoGeneros.obten(genero);
    }
    catch (DAOException pe) {
      throw new PersistenciaException("No se puede obtener el género", pe);
    }
  }

  /**
   * Agrega un género al catálogo de géneros. No se permiten géneros
```

```
   * con claves repetidas
   * @param genero Género a agregar
   * @throws PersistenciaException Si el género está repetido o no se puede
   * agregar el género al catálogo de géneros.
   */
  public void agrega(Genero genero) throws PersistenciaException {
    Genero generoBuscado;

    // Busca el género en el archivo con la misma clave.
    try {
      generoBuscado = catalogoGeneros.obten(genero);

      // Si lo hay, no se agrega al archivo
      if(generoBuscado != null)
        throw new PersistenciaException("Género repetido");
    }
    catch (DAOException pe) {
      // Si el archivo no existe no se hace nada
    }

    // Agrega el nuevo género al catálogo
    try {
      catalogoGeneros.agrega(genero);
    }
    catch (DAOException pe) {
      throw new PersistenciaException("No se puede agregar el género", pe);
    }
  }

  /**
   * Actualiza un género del catálogo de géneros
   * @param genero Género a actualizar
   * @throws PersistenciaException Si no se puede actualizar el género del
   * catálogo de géneros.
   */
  public void actualiza(Genero genero) throws PersistenciaException {
    // Actualiza el género del catálogo
    try {
      catalogoGeneros.actualiza(genero);
    }
    catch (DAOException pe) {
      throw new PersistenciaException("No se puede actualizar el género",
          pe);
    }
  }

  /**
   * Elimina un género del catálogo de géneros
   * @param genero Género a eliminar
   * @throws PersistenciaException Si no se puede eliminar el género del
   * catálogo de géneros.
   */
  public void elimina(Genero genero) throws PersistenciaException {
    // Elimina el género del catálogo
    try {
      catalogoGeneros.elimina(genero);
    }
```

```
    catch (DAOException pe) {
      throw new PersistenciaException("No se puede eliminar el género", pe);
    }
  }

   /**
   * Le agrega los atributos del género a cada canción de la lista
   * @param listaCanciones Lista de las canciones a las que se les
   * agregará los atributos del género
   * @return Lista de canciones
   * @throws PersistenciaException Si hay un problema al conectarse a la
   * base de datos
   */
  private List<Cancion> agregaGeneroCanciones(List<Cancion> listaCanciones)
    throws PersistenciaException  {
      Genero genero;
      Cancion cancion;

      try {
        // Para cada canción de la lista
        for (int i = 0; i < listaCanciones.size(); i++) {
          // Obtén la canción de la lista
          cancion = (Cancion) listaCanciones.get(i);

          // Obten el género de la canción del catálogo de géneros
          genero = catalogoGeneros.obten(cancion.getGenero());

          // Agrega el género a la canción
          cancion.setGenero(genero);
          listaCanciones.set(i, cancion);
        }

        // Regresa la lista canciones
        return listaCanciones;
      }
      catch (DAOException pe) {
        throw new PersistenciaException(
           "No se puede obtener la lista de canciones", pe);
      }
  }

 /**
   * Obtiene una lista todas las canciones
   * @return Lista de todas las canciones
   * @throws PersistenciaException Si no se puede obtener la lista de
   * canciones
   */
  public List<Cancion> consultaCanciones() throws PersistenciaException {
    // Regresa la lista de canciones
    try {
      return agregaGeneroCanciones(catalogoCanciones.lista());
    }
    catch (DAOException pe) {
      throw new PersistenciaException(
        "No se puede obtener la lista de canciones", pe);
    }
  }
```

```
  /**
   * Obtiene una lista de todas las canciones con el mismo título.
   * @param titulo Titulo de las canciones de la lista
   * @return Lista de todas las canciones con el mismo título.
   * @throws PersistenciaException Si no se puede obtener la lista de
   * canciones
   */
  public List<Cancion> consultaCancionesTitulo(String titulo)
    throws PersistenciaException {
    // Regresa la lista de canciones
    try {
      return agregaGeneroCanciones(catalogoCanciones.listaTitulo(titulo));
    }
    catch (DAOException pe) {
      throw new PersistenciaException(
         "No se puede obtener la lista de canciones", pe);
    }
  }

...

  /**
   * Obtiene una lista de los géneros de canciones
   * @return Lista de los géneros de canciones
   */
  public List<Genero> consultaGenerosCanciones()
     throws PersistenciaException {
    // Regresa la lista de géneros de canciones
    try {
      return catalogoGeneros.listaMedio('C');
    }
    catch (DAOException pe) {
      throw new PersistenciaException(
         "No se puede obtener la lista de géneros", pe);
    }
  }

  /**
   * Obtiene una lista de los géneros de películas
   * @return Lista de los géneros de películas
   */
  public List<Genero> consultaGenerosPeliculas()
     throws PersistenciaException {
    // Regresa la lista de géneros de películas
    try {
      return catalogoGeneros.listaMedio('P');
    }
    catch (DAOException pe) {
      throw new PersistenciaException(
         "No se puede obtener la lista de géneros", pe);
    }
  }
}
```

Para probar la clase `PersistenciaArchivos` y las clases `Canciones`, `Peliculas` y `Generos` en su versión que utilizan archivos como mecanismo de persistencia podemos usar la clase de prueba `Prueba3` vista en el Tema 6: Colecciones modificando sólo dos líneas:

- Cambiar la línea:

```
import persistencia.PersistenciaListas;
```

  por ésta:

```
import persistencia.PersistenciaArchivos;
```

- Cambiar la línea:

```
IPersistencia persistencia = new PersistenciaListas();
```

  por ésta:

```
IPersistencia persistencia = new PersistenciaArchivos();
```

Las modificaciones se muestran en el siguiente código:

Prueba4

```
/*
 * Prueba4.java
 *
 * Created on 15 de septiembre de 2007, 12:21 PM
 *
 * @author mdomitsu
 */

package pruebas;

import java.util.List;

import objetosServicio.*;
import objetosNegocio.*;
import persistencia.PersistenciaArchivos;
import interfaces.IPersistencia;
import excepciones.PersistenciaException;

/**
 * Esta clase se utiliza para probar la clase PersistenciaArchivos
 */
public class Prueba4 {
  /**
   * Método main() en el que se invocan a los métodos de la clase
   * PersistenciaArchivos para probarlos
   * @param args the command line arguments
   */
  public static void main(String[] args) {
```

```
    Prueba4 prueba4 = new Prueba4();

    // Crean la fachada de los objetos que permiten almacenar las
    // canciones y peliculas en arreglos
    IPersistencia persistencia = new PersistenciaArchivos();
    List<Genero> listaGeneros = null;
    List<Cancion> listaCanciones = null;
    List<Pelicula> listaPeliculas = null;
    Genero genero;
    Cancion cancion;

    // Se crean tres géneros de canciones
    Genero genero1 = new Genero("GC001", "Balada", 'C');
    Genero genero2 = new Genero("GC002", "Bossanova", 'C');
    Genero genero3 = new Genero("GC003", "Rock", 'C');

    // Se agrega el género 1 al catálogo de géneros
    try {
      persistencia.agrega(genero1);
      System.out.println("Se agrego el género 1 al catálogo de géneros");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 1");
    }

    // Se agrega el género 2 al catálogo de géneros
    try {
      persistencia.agrega(genero2);
      System.out.println("Se agrego el género 2 al catálogo de géneros");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 2");
    }

    // Se agrega el género 3 al catálogo de géneros
    try {
      persistencia.agrega(genero3);
      System.out.println("Se agrego el género 3 al catálogo de géneros");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 3");
    }

    // Se agrega de nuevo el género 1 al catálogo de géneros
    try {
      persistencia.agrega(genero1);
      System.out.println("Se agrego el género 1 al catálogo de géneros");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 1");
    }

    // Se crean tres géneros de películas
    Genero genero4 = new Genero("GP001", "Drama", 'P');
    Genero genero5 = new Genero("GP002", "Ciencia Ficción", 'P');
    Genero genero6 = new Genero("GP003", "Comedia", 'P');
```

```
    // Se agrega el género 4 al catálogo de géneros
    try {
      persistencia.agrega(genero4);
      System.out.println("Se agrego el género 4 al catálogo de géneros");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 4");
    }

    // Se agrega el género 5 al catálogo de géneros
    try {
      persistencia.agrega(genero5);
      System.out.println("Se agrego el género 5 al catálogo de géneros");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 5");
    }

    // Se agrega el género 6 al catálogo de géneros
    try {
      persistencia.agrega(genero6);
      System.out.println("Se agrego el género 6 al catálogo de géneros");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 6");
    }

    // Despliega el contenido del catalogo de géneros
    System.out.println("Lista de géneros");
    try {
      listaGeneros = persistencia.consultaGeneros();
      System.out.println(listaGeneros);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se modifica el genero de clave "GC002", a "Samba"
    try {
      genero = persistencia.obten(new Genero("GC002"));
      genero.setNombre("Samba");
      persistencia.actualiza(genero);
      System.out.println("Se actualizo el genero de clave GC002");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + "GC002");
    }

    // Se elimina el género "GP003" del catalogo de generos
    try {
      persistencia.elimina(new Genero("GP003"));
      System.out.println("Se elimino el genero de clave GP003");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + "GP003");
    }
```

```
    // Se elimina el género "GP004" (inexistente) del catalogo de generos
    try {
      persistencia.elimina(new Genero("GP004"));
      System.out.println("Se elimino el genero de clave GP004");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " GC004");
    }

    // Despliega el contenido del catalogo de géneros
    System.out.println("Lista de géneros");
    try {
      listaGeneros = persistencia.consultaGeneros();
      System.out.println(listaGeneros);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Despliega el contenido del catalogo de géneros de canciones
    System.out.println("Lista de géneros de canciones");
    try {
      listaGeneros = persistencia.consultaGenerosCanciones();
      System.out.println(listaGeneros);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Despliega el contenido del catalogo de género de películas
    System.out.println("Lista de géneros de películas");
    try {
      listaGeneros = persistencia.consultaGenerosPeliculas();
      System.out.println(listaGeneros);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se crean tres canciones
    Cancion cancion1 = new Cancion("C0001", "The long and winding road",
                                   genero1, "The Beatles", "John Lennon",
                                   "Let it be", 3, new Fecha(24, 3, 1970));
    Cancion cancion2 = new Cancion("C0002", "Garota de Ipanema", genero2,
                                   "Los Indios Tabajaras",
                                   "Antonio Carlos Jobim",
                                   "Bossanova Jazz Vol. 1", 3,
                                   new Fecha(1, 12, 1970));
    Cancion cancion3 = new Cancion("C0003", "Desafinado", genero2,
                                  "Joao Gilberto", "Joao Gilberto",
                                  "Bossanova Jazz Vol. 1", 3,
                                  new Fecha(3, 12, 1980));

    // Se agrega la canción 1 al catálogo de canciones
    try {
      persistencia.agrega(cancion1);
      System.out.println("Se agrego la cancion 1");
```

```
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 1");
    }

    // Se intenta agregar de nuevo la canción 1 al catálogo de canciones
    try {
      persistencia.agrega(cancion1);
      System.out.println("Se agrego la cancion 1");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 1");
    }

    // Se agrega la canción 2 al catálogo de canciones
    try {
      persistencia.agrega(cancion2);
      System.out.println("Se agrega la cancion 2");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 2");
    }

    // Se agrega la canción 3 al catálogo de canciones
    try {
      persistencia.agrega(cancion3);
      System.out.println("Se agrega la cancion 3");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 3");
    }

    // Se lista el catálogo de canciones
    System.out.println("Lista de canciones:");
    try {
      listaCanciones = persistencia.consultaCanciones();
      System.out.println(listaCanciones);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se actualiza la canción de clave "C0001" al genero "GC003"
    try {
      // Obtiene el género 3 del cátalogo de géneros de canciones
      genero = persistencia.obten(new Genero("GC003"));
      if(genero != null) {
        // Obtiene la canción 1 del cátalogo de canciones
        cancion = persistencia.obten(new Cancion("C0001"));
        if(cancion != null) {
          // Se actualiza la canción 1
          cancion.setGenero(genero);
          persistencia.actualiza(cancion);
          System.out.println("Se actualizo la canción de clave C0001 al
genero GC003");
        } else System.out.println("No existe la canción C0001");
      } else System.out.println("No existe el género GC003");
```

```
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se elimina la canción de clave "C0003"
    try {
        persistencia.elimina(new Cancion("C0003"));
        System.out.println("Se elimina la cancion de clave C0003");
    } catch(PersistenciaException fe) {
        // Muestra el mensaje de error amistoso
        System.out.println("Error: " + fe.getMessage() + " C0003");
    }

    // Se lista el catálogo de canciones
    System.out.println("Lista de canciones:");
    try {
      listaCanciones = persistencia.consultaCanciones();
      System.out.println(listaCanciones);
    } catch(PersistenciaException fe) {
        // Muestra el mensaje de error amistoso
        System.out.println("Error: " + fe.getMessage());
    }

    // Se lista las canciones con el interprete "The Beatles"
    System.out.println("Lista de canciones de The Beatles:");
    try {
      listaCanciones = persistencia.consultaCancionesInterprete("The
Beatles");
      System.out.println(listaCanciones);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se lista las canciones de samba, "GC002"
    System.out.println("Lista de canciones de Samba:");
    try {
      listaCanciones = persistencia.consultaCancionesGenero("GC002");
      System.out.println(listaCanciones);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se crean dos peliculas
    Pelicula pelicula1 = new Pelicula("P000001", "Casa Blanca", genero4,
                                      "Humphrey Bogart", "Ingrid Bergman",
                                      "Michael Curtiz", 102,
                                      new Fecha(1, 1, 1942));
    Pelicula pelicula2 = new Pelicula("P000002", "2001 Space Odyssey",
                                      genero5, "Keir Dullea",
                                      "Gary Lockwood", "Stanley Kubrick",
                                      141, new Fecha(1, 1, 1968));

    // Se agrega la película 1 al catálogo de películas
    try {
```

```
      persistencia.agrega(pelicula1);
      System.out.println("Se agrego la película 1");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 1");
    }

    // Se agrega la película 2 al catálogo de películas
    try {
      persistencia.agrega(pelicula2);
      System.out.println("Se agrego la película 2");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 2");
    }

    // Se lista el catálogo de películas
    System.out.println("Lista de peliculas:");
    try {
      listaPeliculas = persistencia.consultaPeliculas();
      System.out.println(listaPeliculas);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se lista las películas de Ingrid Bergman
    System.out.println("Lista de peliculas de Ingrid Bergman:");
    try {
      listaPeliculas = persistencia.consultaPeliculasActor("Ingrid Bergman");
      System.out.println(listaPeliculas);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Lista de peliculas en el periodo: 1/03/1970 a 1/05/1970
    Periodo periodo = new Periodo(new Fecha(1, 1, 1960),
                      new Fecha(1, 1, 1970));
    System.out.println("Lista de peliculas en el periodo: " + periodo);
    try {
      listaPeliculas = persistencia.consultaPeliculasPeriodo(periodo);
      System.out.println(listaPeliculas);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }
  }
}
```

Estos dos cambios son también los únicos que hay que hacerle a la clase de control Control vista en el Tema 8: Desarrollo de Aplicaciones para cambiar el mecanismo de persistencia, obviamente también es necesario sustituir la clase de persistencia y las clases `Canciones`, `Peliculas` y `Generos` a sus versiones de archivos.