Tema 8

Entrada y Salida

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 8.1 se muestra un diagrama parcial de las interfaces y clases que podemos utilizar en este tipo de problemas.

- 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 8.1.

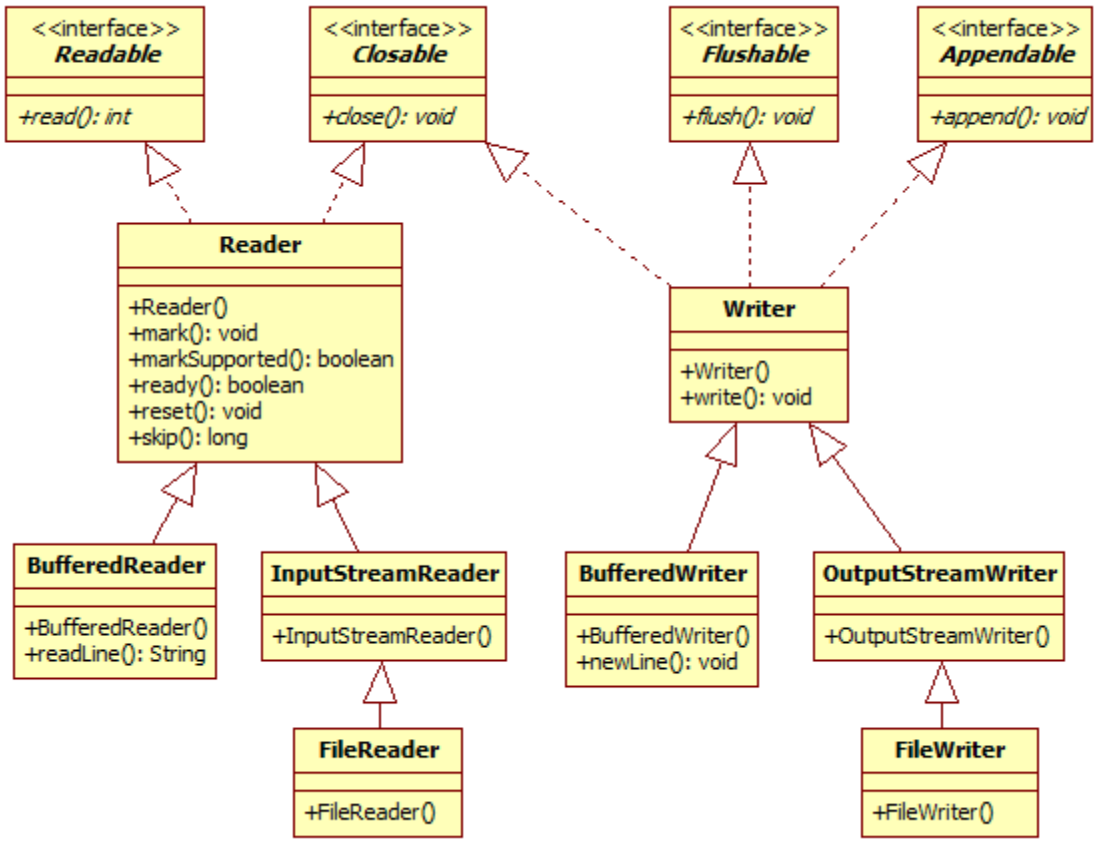


Figura 8.1 Interfaces y Clases para Archivos de Caracteres de Acceso Secuencial

Tabla 8.1 Método Declarado en la Interfaz `Readable`.

`int read(CharBuffer` *`cb`***`) throws IOException`** Intenta leer caracteres y los almacena en un buffer especificado **Parámetro**: *cb*. El bufeer en que se escribirán los caracteres. **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 8.2.

Tabla 8.2 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 8.3.

Tabla 8.3 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 8.4.

Tabla 8.4 Métodos Declarados en la Interfaz `Appendable`.

`Appendable append(char` *`c`*`) throws IOException` Agrega un carácter al objeto. **Parámetro**: *`c`*. El carácter a ser agregado al objeto. **Regresa**: Una referencia al objeto agregable. **Lanza**: `IOException` – Si ocurre un error de entrada/salida
`Appendable append(charSecuence` *`csq`*`) throws IOException` Agrega una secuencia de caracteres al objeto. **Parámetro:** *`csq`*. Secuencia de caracteres a ser agregados al objeto. **Regresa:** Una referencia al objeto agregable. **Lanza**: `IOException` **–** Si ocurre un error de entrada/salida

Tabla 8.4 Métodos Declarados en la Interfaz `Appendable`. Cont.

`Appendable append(charSecuence` *`csq`***`, int`** *`start`***`, int`** *`end`***`) throws IOException`** Agrega una subsecuencia de caracteres al objeto. **Parámetros:** *`csq`*. Secuencia de la que se obtendrá la secuencia de caracteres a ser agregados al objeto. *`start`*. Índice del primer carácter de la subsecuencia. *`end`*. Índice del último carácter de la subsecuencia. **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 8.5.

Tabla 8.5 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. **Parámetros**: *`readAheadLimit`*. 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()`.

Tabla 8.5 Métodos de la clase `Reader`. Cont.

`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
`int read(char[] cbuf) throws IOException` Lee caracteres en un arreglo. **Parámetro**: *`cbuf`*: Arreglo en el que se almacenarán los caracteres. **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 caracteres en una porción de un arreglo. **Parámetro**: *`cbuf`*: Arreglo en el que se almacenarán los caracteres. *`off`*: Índice en el arreglo del lugar en la que se empezarán a guardar los caracteres. *`len`*: Máximo número de caracteres a leer. **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. **Parámetro:** *`target`*: Arreglo en el que se almacenarán los caracteres. **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.

Tabla 8.5 Métodos de la clase `Reader`. Cont.

`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 **Parámetro:** *n*: Número de caracteres a leer y descartar. **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 8.6.

Tabla 8.6 Métodos de la clase `BufferedReader`.

`BufferedReader(Reader` *`in`*`)` Crea un flujo de caracteres de entrada con buffer que usa el buffer de entrada de tamaño predeterminado. **Parámetro:** *`in`*: Una instancia de `Reader`.
`BufferedReader(Reader` *`in`*`, int` *`sz`*`)` Crea un flujo de caracteres de entrada con buffer que usa el buffer de entrada de tamaño `sz`. **Parámetros:** *`in`*: Una instancia de `Reader`. *`sz`*: Tamaño del buffer de entrada. **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 8.7.

Tabla 8.7 Constructor de la clase `InputStreamReader`.

`InputStreamReader(InputStream` *`in`*`)` Crea un flujo del tipo `InputStreamReader`. **Parámetro:** *`in`*: Un flujo de entrada de tipo `InputStream`.

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

El constructor de esta clase se muestra en la tabla 8.8.

Tabla 8.8 Constructor de la clase `FileReader`.

`FileReader(String` *`fileName`*`) throws FileNotFoundException` Crea un flujo del tipo `FileReader`. **Parámetro**: *`fileName`*: Nombre del archivo a abrir para lectura**.** **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 8.9.

Tabla 8.9 Métodos de la clase `Writer`.

`Writer()` Crea un flujo de caracteres de salida.
`void write(int` ***`c`***`)` Escribe un solo caracter. **Parámetro**: **c** - Caracter a escribir. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void Write(char[]` *`cbuf`*`) throws IOException` Escribe un arreglo de caracteres. **Parámetro**: *`cbuf`* - Arreglo de caracteres a escribir. **Lanza**: `IOException` – Si ocurre un error de entrada/salida.
`abstract void Write(char[]` *`cbuf`***`, int`** *`off`***`, int`** *`len`*`) throws IOException` Escribe una porción de un arreglo de caracteres. **Parámetros**: *`cbuf`* - Arreglo de caracteres con la porción de caracteres a escribir. *`off`* - Índice de la posición en el arreglo a partir de la cual se escribirá. *`len`* - Número de caracteres a escribir. **Lanza**: `IOException` – Si ocurre un error de entrada/salida.

Tabla 8.9 Métodos de la clase `Writer`. Cont.

`void Write(String` *`str`*`) throws IOException` Escribe una cadena. **Parámetro**: *`str`* - Cadena a escribir. **Lanza**: `IOException` – Si ocurre un error de entrada/salida.
`void Write(String` *`str`*`, int`** ***`off`*`, int`** *`len`*`) throws IOException` Escribe una porción de una cadena. **Parámetro**: *`str`* – Cadena con la porción de caracteres a escribir.. *`off`* - Índice de la posición en la cadena a partir de la cual se escribirá. *`len`* - Número de caracteres a escribir. **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 8.10.

Tabla 8.10 Métodos de la clase `BufferedWriter`.

`BufferedWriter(Writer` *`out`***`)`** Crea un flujo de caracteres de salida con buffer que usa el buffer de entrada de tamaño predeterminado. Parámetro: *`out`*: Una instancia de `Writer`.

Tabla 8.10 Métodos de la clase `BufferedWriter`. Cont.

`BufferedWriter(Writer` *`out`***`, int`** *`sz`***`)`** Crea un flujo de caracteres de salida con buffer que usa el buffer de entrada de tamaño `sz`. **Parámetros**: *`out`*: Una instancia de `Writer`. *`sz`*: Tamaño del buffer de entrada. **Lanza**: `IllegalArgumentException` – Si `sz` <= 0
`void newLine()throws IOException` Escribe un separadoe 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 8.11.

Tabla 8.11 Constructor de la clase `OutputStreamWriter`.

`OutputStreamWriter(OutputStream` *`out`***`)`** Crea un flujo del tipo `OutputStreamWriter`. **Parámetro**: *`out`*: Un flujo de entrada de tipo **`OutputStream.`**

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

  Los constructores de esta clase se muestran en la tabla 8.12.

Tabla 8.12 Constructores de la clase `FileWriter`.

`FileWriter(String` *`fileName`*`)` `throws IOException` Crea un flujo del tipo `FileWriter`. **Parámetro**: *`fileName`*: Nombre del archivo a abrir para escritura**.** **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 del tipo `FileWriter` con `append` indicando si los datos a escribir se agregan al final del archive o no.. **Parámetro**: *`fileName`*: Nombre del archivo a abrir para escritura**.** **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);
  }
}
```

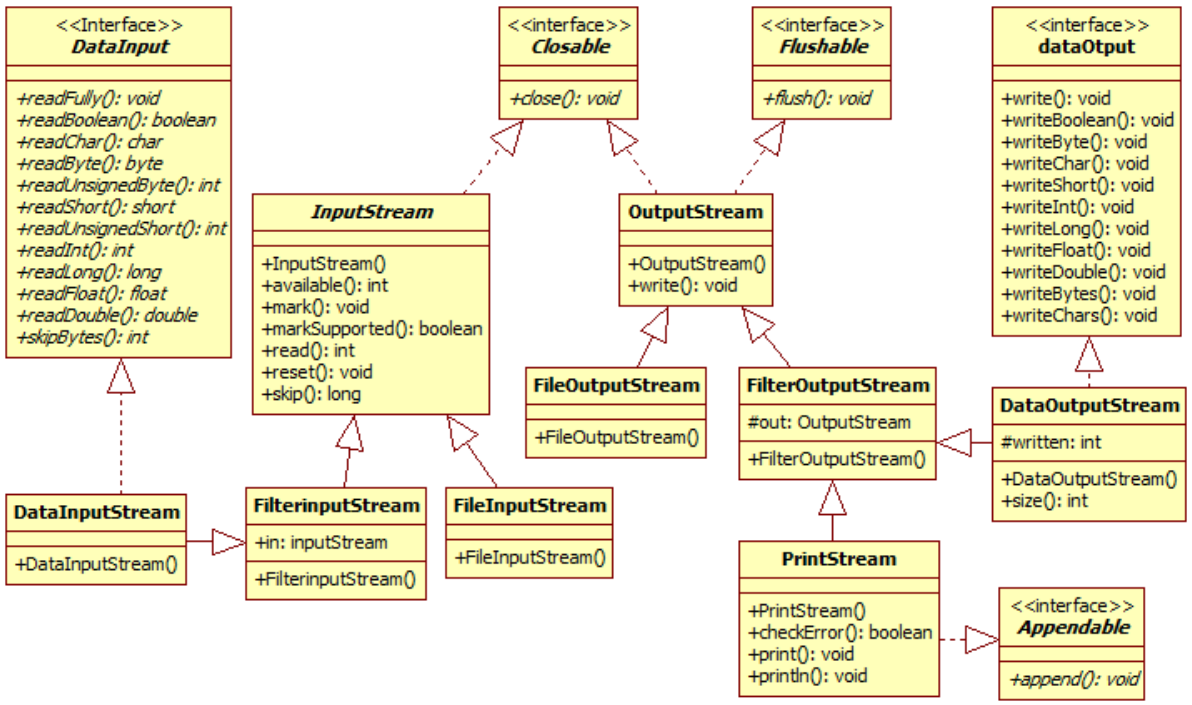
Al ejecutar el programa sobre la clase Estadisticas, tendremos el siguiente resultado:

```
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 8.2 se muestra un diagrama parcial de las interfaces y clases que podemos utilizar en este tipo de problemas.



8.2 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 8.13:

Tabla 8.13 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.

Parámetro:

b - El arreglo en el que se almacenan los bytes leídos.

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`.

Parámetros:

b - El arreglo en el que se almacenan los bytes leídos.

off – Un entero indicando el desplazamiento a partir del inicio del arreglo en que se almacenarán los datos.

len - Un entero indicando el número de bytes a leer.

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`.

Regresa:

El valor booleano leído.

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`.

Regresa:

El `char` Unicode leído.

Lanza:

`EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes.

`IOException` – Si ocurre un error de entrada/salida.

Tabla 8.13 Métodos de la Interfaz `DataInput`. Cont.

`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`. **Regresa:** El valor de 8 bits leído. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.
`int readUnsignedByte()IOException` Lee 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. **Regresa:** El valor de 8 bits leído como un número no signado. **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`. **Regresa:** El valor de 16 bits leído. **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. **Regresa:** El entero leído como un número no signado. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.

Tabla 8.13 Métodos de la Interfaz `DataInput`. Cont

`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`. **Regresa:** El entero leído. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.
`long readLon``g()`**`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`. **Regresa:** El entero largo leído. **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`. **Regresa:** El flotante leído. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.
`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`. **Regresa:** El doble leído. **Lanza:** `EOFException` – Si se llega al final del archivo sin haber leído todos sus bytes. `IOException` – Si ocurre un error de entrada/salida.

Tabla 8.13 Métodos de la Interfaz `DataInput`. Cont

`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. **Parámetros:** *n* - El número de bytes a leer y descartar. **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 8.14:

Tabla 8.14 Métodos de la Interfaz `DataOutput`.

`void write(int` *`b`*`)` Escribe al flujo de bytes el byte dado por el parámetro. **Parámetro:** *b* – Contiene el carácter a escribir en el flujo de bytes. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void write(byte[]` *`b`*`)` Escribe al flujo de bytes, todos los bytes del arreglo b. **Parámetro:** *b* - El arreglo de bytes a escribir en el flujo de bytes. **Lanza:** `IOException` – Si ocurre un error de entrada/salida. `NullPointerException` – Si *b* es `null`.

Tabla 8.14 Métodos de la Interfaz `DataOutput`. Cont.

`void write(byte[]` *`b`*`, int` *`off`*`, int` *`len`*`)` Escribe `len` bytes al flujo de bytes, del arreglo `b` a partir de la posición `off`. **Parámetros:** *`b`* - El arreglo del que se escribirán los bytes. *`off`* – Un entero indicando el desplazamiento a partir del inicio del arreglo del que se escribirán los datos. *`len`* - Un entero indicando el número de bytes a escribir. **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`.
`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. **Parámetro:** *`v`* - El valor booleano a escribirse. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void writeByte(int` *`v`*`)` Escribe al flujo de bytes un byte. **Parámetro:** *`v`* - El byte a escribirse. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void` **`writeChar`**`(int v)` Escribe al flujo de bytes un carácter. **Parámetro:** *`v`* - El char a escribirse. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void writeShort``(int v)` Escribe al flujo de bytes un entero corto. **Parámetro:** *`v`* - El entero corto a escribirse. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.

Tabla 8.14 Métodos de la Interfaz `DataOutput`. Cont.

`void` **`writeInt`**`(int v)` Escribe al flujo de bytes un entero. **Parámetro:** *v* - El entero a escribirse. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void` **`writeLong`**`(long v)` Escribe al flujo de bytes un entero largo. **Parámetro:** *v* - El entero largo a escribirse. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void` **`writeFloat`**`(float v)` Escribe al flujo de bytes un flotante. **Parámetro:** *v* - El flotante a escribirse. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void` **`writeDouble`**`(double v)` Escribe al flujo de bytes un doble. **Parámetro:** *v* - El doble a escribirse. **Lanza:** `IOException` – Si ocurre un error de entrada/salida.
`void writeBytes``(String s)` Escribe al flujo de bytes una cadena. Por cada carácter de la cadena se escribe un byte. **Parámetro:** *s* – La cadena a escribirse. **Lanza:** `IOException` – Si ocurre un error de entrada/salida. `NullPointerException` – Si *s* es `null`.

Tabla 8.14 Métodos de la Interfaz `DataOutput`. Cont.

`void writeChars``(String s)` Escribe al flujo de bytes una cadena. Por cada carácter de la cadena se escribe un caracter. **Parámetro:** *s* – La cadena a escribirse. **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 8.15.

Tabla 8.15 Métodos de la clase `InputStream`.

`InputStream()` Crea un flujo de bytes de entrada.
`int available()throws IOException` Regresa el número que pueden leerse (o leers ey descartarse) del flujo de entrada sin que el flujo se bloqueé en la siguiente invocación de un método de este flujo. **Regresa**: Número que pueden leerse (o leers ey descartarse) del flujo de entrada sin que el flujo se bloqueé. **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. **Parámetros**: *`readLimit`*. 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()`.

Tabla 8.15 Métodos de la clase `InputStream`. Cont.

`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
`int read(byte[]` *`b`*`) throws IOException` Lee algunos bytes y los almacena en el arreglo *b*. **Parámetro**: *b*: Arreglo en el que se almacenarán los bytes. **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 en un arreglo. **Parámetro**: *b*: Arreglo en el que se almacenarán los bytes. *`off`*: Índice en el arreglo del lugar en la que se empezarán a guardar los bytes. *`len`*: Máximo número de caracteres a leer. **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.

Tabla 8.15 Métodos de la clase `InputStream`. Cont.

`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 **Parámetro:** *n*: Número de caracteres a leer y descartar. **Regresa:** El 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 8.16.

Tabla 8.16 Constructor de la clase `FileInputStream`.

`FileInputStream(String` *`fileName`***`) throws FileNotFoundException`** Crea un flujo del tipo `FileInputStream`. **Parámetro**: *`fileName`*: Nombre del archivo a abrir para lectura**`.`** **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 8.17.

Tabla 8.17 Constructor de la clase `FilterInputStream`.

`FilterInputStream(InputStream` *`in`***`)`** Crea un flujo del tipo **`Filter`**`InputStreamReader`. **Parámetro:** *`in`*: Un flujo de entrada de tipo `InputStream`**`.`**

- 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 8.18.

Tabla 8.18 Constructor de la clase `DataInputStream`.

`DataInputStream(InputStream` *`in`*`) throws FileNotFoundException` Crea un flujo del tipo `DataInputStreamReader`. **Parámetro:** *`in`*: Un flujo de entrada de tipo `InputStream.`

- 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 8.19.

Tabla 8.19 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. **Parámetro**: **`c`** - Caracter a escribir. **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. **Parámetro**: *`b`* - Arreglo de bytes a escribir. **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. **Parámetros**: *`cbuf`* - Arreglo de bytes con la porción de bytes a escribir. *`off`* - Índice de la posición en el arreglo a partir de la cual se escribirá. *`len`* - Número de caracteres a escribir. **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 8.20.

Tabla 8.20 Constructores de la clase `FileOutputStream`.

`FileOutputStream(String` *`fileName`*`) throws IOException` Crea un flujo del tipo `FileOutputStream`. **Parámetro**: *`fileName`*: Nombre del archivo a abrir para escritura**.** **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` con `append` indicando si los datos a escribir se agregan al final del archive o no.. **Parámetro**: *`filename`*: Nombre del archivo a abrir para escritura**.** **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 8.21.

Tabla 8.21 Constructor de la clase `FilterOutputStream`.

`FilterOutputStream(OutputStream` *`out`***`)`** Crea un flujo del tipo `FilterOutputStreamWriter`. **Parámetro**: *`out`*: Un flujo de entrada de tipo **`OutputStream.`**

- 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 8.22.

Tabla 8.22 Métodos de la clase `DataOutputStream`.

`DataOutputStream`**`(OutputStream`** *`out`***`)`** Crea un flujo de salida de datos para escribir datos a un flujo de salida. **Parámetro**: *`out`*: Flujo de salida al que envuelve**.**

Tabla 8.22 Métodos de la clase `DataOutputStream`. Cont.

`final int size()` Regresa el número de bytes escritos al flujo de salida. **Regresa**: 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 8.23.

Tabla 8.23 Métodos de la clase `PrintStream`.

`PrintStream(OutputStream out)` Crea un flujo de salida nuevo. **Parámetro**: *`out`*: Flujo de salida al que envuelve**.**
`PrintStream(String fileName) throws FileNotFoundException` Crea un flujo de salida nuevo con el nombre de archivo dado. Si el archivo existe su longitud se trunca a cero. **Parámetro**: *`filename`*: nombre de archivo. **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()`.

Tabla 8.23 Métodos de la clase `PrintStream`.

`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");
  }
}
```

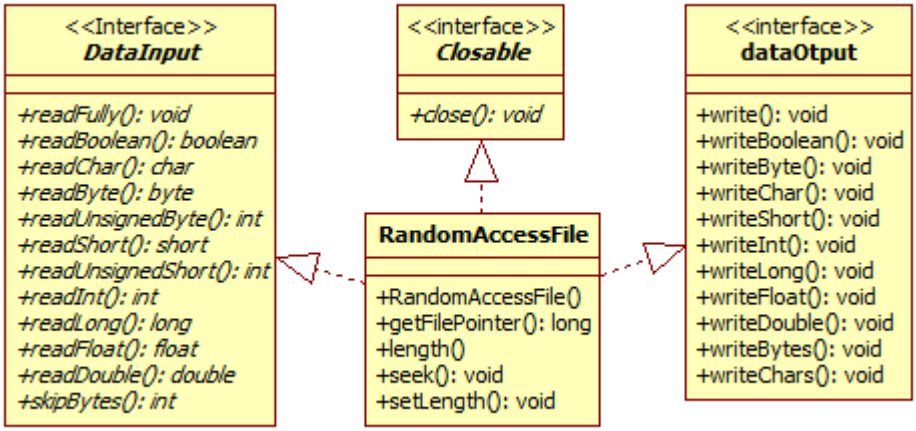
La corrida del programa anterior se muestra a continuación:

```
                      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 8.3 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()`.



8.3 Interfaces y Clases para Archivos de Acceso Aleatorio

Los métodos de esta clase se muestran en la tabla 8.24.

Tabla 8.24 Métodos de la clase `RandomAccessFile`.

`RandomAccessFile(String` *`fileName`*`, String` *`mode`*`) throws FileNotFoundException` Crea un flujo para un archivo de acceso aleatorio con el nombre de archivo y modo de acceso dados. 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. **Parámetros**: *`fileName`* – Nombre del archivo a abrirse o crearse. *`mode`* – Modo de apertura. **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.
`long getFilePointer()throws IOException` Regresa el valor actual del apuntador de archivo. Regresa: 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.

Tabla 8.24 Métodos de la clase `RandomAccessFile`.

`long length()throws IOException` Regresa el tamaño de este archivo. Regresa: Tamaño de este archivo en bytes. Lanza: `IOException` - Si ocurre un error de entrada/salida.
`void seek(long pos) IOException` Establece el desplazamiento 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. Parámetro: *pos* - El desplazamiento con respecto al inicio del archivo en bytes de la posición en la que establecerá el apuntador de archivo. 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. 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. Parámetro: Valor deseado del tamaño de archivo. Lanza: `IOException` - Si ocurre un error de entrada/salida.

Ejemplo de Archivos de Acceso Aleatorio

En el Tema 4: Arreglos y Cadenas 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 8.4.

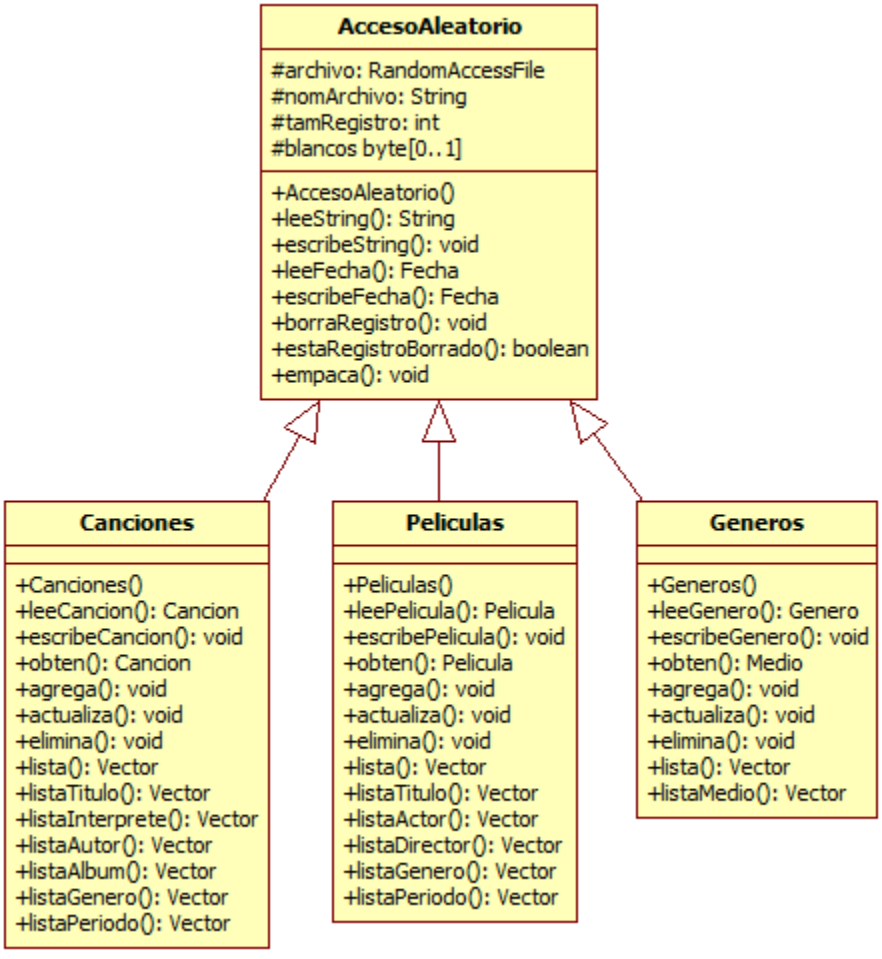


Figura 8.4 Diagrama de Clases del Mecanismo de Persistencia Basado en Archivos del Programa Amante Música.

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

AccesoAleatorio.java

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

package persistencia;

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 `PersistenciaException`, la cual fue vista en el tema 5: Arreglos y Cadenas.

Canciones.java

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

package persistencia;
```

```
import java.io.*;
import java.util.Vector;

import objetosServicio.*;
import objetosNegocio.*;
import excepciones.PersistenciaException;

/**
 * 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
   * Este método obtiene una canción de un archivo cuya clave es igual a la
   * clave de la canción dada por el parámetro.
   * @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 PersistenciaException Si hay un error de entrada / salida
   * o el archivo no existe.
   */
  public Cancion obten(Cancion cancion) throws PersistenciaException {
    Cancion cancionLeida;

    // Abre el archivo de sólo lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
```

```
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite agregar una canción a un archivo.
   * @param cancion Cancion a agregar en la tabla canciones
   * @throws PersistenciaException Si hay un error de entrada / salida,
   * el archivo no existe.
   */
  public void agrega(Cancion cancion) throws PersistenciaException {
    // Abre el archivo de escritura/lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("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 PersistenciaException Si hay un error de entrada / salida,
   * el archivo no existe o no se puede actualizar la canción.
   */
  public void actualiza(Cancion cancion) throws PersistenciaException {
    Cancion cancionLeida;
```

```
    // Abre el archivo de escritura/lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 PersistenciaException("La canción no existe");
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("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 PersistenciaException Si hay un error de entrada / salida,
   * el archivo no existe o no se puede eliminar la canción.
   */
```

```
  public void elimina(Cancion cancion) throws PersistenciaException {
    Cancion cancionLeida;

    // Abre el archivo de escritura/lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 PersistenciaException("La canción no existe");
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite consultar las canciones del archivo canciones.
   * @return Un vector con la lista de las canciones del archivo canciones
   * @throws PersistenciaException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
```

```
  public Vector lista() throws PersistenciaException {
    Vector lista = new Vector();
    Cancion cancion;

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

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

        // Agrega la canción al vector 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 PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("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 Un vector con la lista de las canciones del archivo canciones
   * @throws PersistenciaException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public Vector listaTitulo(String titulo) throws PersistenciaException {
    Vector lista = new Vector();
     Cancion cancion;

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

```
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 al vector 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 PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("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 Un vector con la lista de las canciones del archivo canciones
   * @throws PersistenciaException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public Vector listaGenero(String cveGenero) throws PersistenciaException {
    Vector lista = new Vector();
    Cancion cancion;

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

```
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 al vector 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 PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("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 Un vector con la lista de las canciones del archivo canciones
   * @throws PersistenciaException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public Vector listaPeriodo(Periodo periodo) throws PersistenciaException {
    Vector lista = new Vector();
    Cancion cancion;

    // Abre el archivo de sólo lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 al vector 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 PersistenciaException("Error al acceder al archivo");
      }
      finally {
        try {
          // Cierra el archivo
          archivo.close();
        }
        // Si ocurrió un error de entrada salida
        catch (IOException eofe) {
          throw new PersistenciaException("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 `PersistenciaException`, la cual fue vista en el tema 5: Arreglos y Cadenas.

Peliculas

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

package persistencia;

import java.io.*;
import java.util.Vector;

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

```
/**
 * 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 PersistenciaException Si hay un error de entrada / salida
   * o el archivo no existe.
   */
  public Pelicula obten(Pelicula pelicula) throws PersistenciaException {
    Pelicula peliculaLeida;

    // Abre el archivo de sólo lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("Error al cerrar el archivo");
      }
```

```
    }
  }

  /**
   * Este método permite agregar una película al archivo peliculas.
   * @param pelicula Pelicula a agregar en la tabla películas.
   * @throws PersistenciaException Si hay un error de entrada / salida,
   * el archivo no existe.
   */
  public void agrega(Pelicula pelicula) throws PersistenciaException {
    // Abre el archivo de escritura/lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("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 PersistenciaException Si hay un error de entrada / salida,
   * el archivo no existe o no se puede actualizar la película.
   */
  public void actualiza(Pelicula pelicula) throws PersistenciaException {
    Pelicula peliculaLeida;

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

```
      throw new PersistenciaException("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 PersistenciaException("La película no existe");
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("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 PersistenciaException Si hay un error de entrada / salida,
   * el archivo no existe o no se puede eliminar la película.
   */
  public void elimina(Pelicula pelicula) throws PersistenciaException {
    Pelicula peliculaLeida;

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

```
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 PersistenciaException("La película no existe");
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite consultar las películas del archivo películas.
   * @return Un vector con la lista de las películas del archivo películas
   * @throws PersistenciaException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public Vector lista() throws PersistenciaException {
    Vector lista = new Vector();
    Pelicula pelicula;

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

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

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

        // Agrega la película al vector 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 PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("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 Un vector con la lista de las películas del archivo películas
   * @throws PersistenciaException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public Vector listaTitulo(String titulo) throws PersistenciaException {
    Vector lista = new Vector();
     Pelicula pelicula;

     // Abre el archivo de sólo lectura
     try {
       archivo = new RandomAccessFile(nomArchivo, "r");
     }
     catch(FileNotFoundException fnfe) {
       throw new PersistenciaException("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 al vector 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 PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("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 Un vector con la lista de las películas del archivo películas
   * @throws PersistenciaException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public Vector listaActor(String actor) throws PersistenciaException {
    Vector lista = new Vector();
    Pelicula pelicula;

    // Abre el archivo de sólo lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 al vector 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 PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("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 `PersistenciaException`, la cual fue vista en el tema 5: Arreglos y Cadenas.

Generos

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

package persistencia;

import java.io.EOFException;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.Vector;

import objetosNegocio.Genero;
```

```
import excepciones.PersistenciaException;

/**
 * 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 PersistenciaException Si hay un error de entrada / salida
   * o el archivo no existe.
   */
  public Genero obten(Genero genero) throws PersistenciaException {
```

```
    Genero generoLeido;

    // Abre el archivo de sólo lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "r");
    }
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite agregar un género a un archivo.
   * @param genero Género a agregar en el archivo géneros
   * @throws PersistenciaException Si hay un error de entrada / salida,
   * el archivo no existe.
   */
  public void agrega(Genero genero) throws PersistenciaException {
    // Abre el archivo de escritura/lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("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 PersistenciaException Si hay un error de entrada / salida,
   * el archivo no existe o no se puede actualizar el género.
   */
  public void actualiza(Genero genero) throws PersistenciaException {
    Genero generoLeido;

    // Abre el archivo de escritura/lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 PersistenciaException("El género no existe");
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("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 PersistenciaException Si hay un error de entrada / salida,
   * el archivo no existe o no se puede eliminar el género.
   */
  public void elimina(Genero genero) throws PersistenciaException {
    Genero generoLeido;

    // Abre el archivo de escritura/lectura
    try {
      archivo = new RandomAccessFile(nomArchivo, "rw");
    }
    catch(FileNotFoundException fnfe) {
      throw new PersistenciaException("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 PersistenciaException("La género no existe");
    }
    // Si ocurrió un error de entrada salida
    catch (IOException eofe) {
      throw new PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("Error al cerrar el archivo");
      }
    }
  }

  /**
   * Este método permite consultar los géneros del archivo generos.
   * @return Un vector con la lista de los objetos del tipo Genero del
   * archivo generos
   * @throws PersistenciaException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public Vector lista() throws PersistenciaException {
    Vector lista = new Vector();
    Genero genero;

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

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

        // Agrega el género al vector 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 PersistenciaException("Error al acceder al archivo");
    }
    finally {
      try {
        // Cierra el archivo
        archivo.close();
      }
      // Si ocurrió un error de entrada salida
      catch (IOException eofe) {
        throw new PersistenciaException("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 Un vector con la lista de géneros del archivo generos que
   * tienen el mismo tipo de medio.
   * @throws PersistenciaException Si hay un error de entrada / salida o
   * el archivo no existe.
   */
  public Vector listaMedio(char tipoMedio) throws PersistenciaException {
    Vector lista = new Vector();
     Genero genero;

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

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

         // Si es la género buscada
         if(tipoMedio == genero.getTipoMedio())
           // Agrega la género al vector 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 PersistenciaException("Error al acceder al archivo");
      }
      finally {
        try {
          // Cierra el archivo
          archivo.close();
        }
        // Si ocurrió un error de entrada salida
        catch (IOException eofe) {
          throw new PersistenciaException("Error al cerrar el archivo");
        }
      }
  }
}
```

Como ya se mencionó en el Tema 5: Arreglos y Cadenas, 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 de fachada `FachadaArchivos`. Para asegurarnos que podamos sustituir un mecanismo de persistencia por otro, la clase `FachadaArchivos` implementa la interfaz `IFachada`, como se muestra en la figura 8.5:

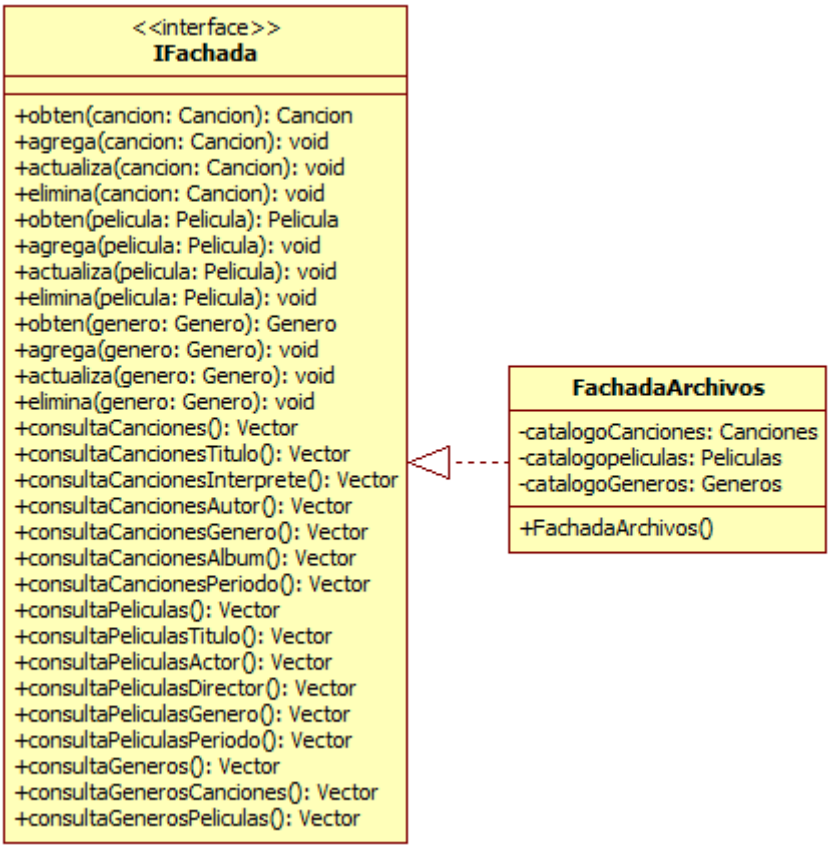


Figura 8.5. Diagrama de clases de FachadaArchivos

Los métodos de la clase `FachadaArchivos` lanzan excepciones de la clase `FachadaException` vista en Tema 5: Arreglos y Cadenas. El código parcial de la clase `FachadaArchivos` es:

FachadaArchivos.java

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

package fachadas;

import java.util.Vector;

import objetosServicio.*;
import objetosNegocio.*;
import persistencia.*;
import interfaces.IFachada;
import excepciones.*;

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

  /**
   * Constructor predeterminado
   */
  public FachadaArchivos() {
    // 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 FachadaException Si no se puede obtener la canción.
   */
```

```
  public Cancion obten(Cancion cancion) throws FachadaException {
    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 (PersistenciaException pe) {
      throw new FachadaException("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 FachadaException Si la canción está repetida o no se puede
   * agregar la canción al catálogo de canciones.
   */
  public void agrega(Cancion cancion) throws FachadaException {
    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 FachadaException("Canción repetida");
    }
    catch (PersistenciaException pe) {
      // Si el archivo no existe no se hace nada
    }

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

  /**
   * Actualiza una canción del catálogo de canciones
   * @param cancion Cancion a actualizar
   * @throws FachadaException Si no se puede actualizar la canción del
   * catálogo de canciones.
```

```
   */
  public void actualiza(Cancion cancion) throws FachadaException {
    // Actualiza la  canción del catálogo
    try {
      catalogoCanciones.actualiza(cancion);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede actualizar la canción", pe);
    }
  }

  /**
   * Elimina una canción del catálogo de canciones
   * @param cancion Cancion a eliminar
   * @throws FachadaException Si no se puede eliminar la canción del
   * catálogo de canciones.
   */
  public void elimina(Cancion cancion) throws FachadaException {
    // Elimina la  canción del catálogo
    try {
      catalogoCanciones.elimina(cancion);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("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 FachadaException Si no se puede obtener la película.
   */
  public Pelicula obten(Pelicula pelicula) throws FachadaException {
    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 (PersistenciaException pe) {
      throw new FachadaException("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 FachadaException Si la película está repetida o no se puede
   * agregar la película al catálogo de películas.
   */
  public void agrega(Pelicula pelicula) throws FachadaException {
    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 FachadaException("Película repetida");
    }
    catch (PersistenciaException pe) {
      // Si el archivo no existe no se hace nada
    }

    // Agrega la nueva película al catálogo
    try {
      catalogoPeliculas.agrega(pelicula);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede agregar la película", pe);
    }
  }

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

  /**
   * Elimina una película del catálogo de peliculas
   * @param pelicula Pelicula a eliminar
   * @throws FachadaException Si no se puede eliminar la película del
   * catálogo de películas.
   */
  public void elimina(Pelicula pelicula) throws FachadaException {
    // Elimina la  película del catálogo
    try {
      catalogoPeliculas.elimina(pelicula);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("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 FachadaException Si no se puede obtener el género.
   */
  public Genero obten(Genero genero) throws FachadaException {
    // Obten el género
    try {
      return catalogoGeneros.obten(genero);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("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 FachadaException Si el género está repetido o no se puede
   * agregar el género al catálogo de géneros.
   */
  public void agrega(Genero genero) throws FachadaException {
    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 FachadaException("Género repetido");
    }
    catch (PersistenciaException pe) {
      // Si el archivo no existe no se hace nada
    }

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

  /**
   * Actualiza un género del catálogo de géneros
   * @param genero Género a actualizar
   * @throws FachadaException Si no se puede actualizar el género del
   * catálogo de géneros.
   */
  public void actualiza(Genero genero) throws FachadaException {
    // Actualiza el género del catálogo
```

```
    try {
      catalogoGeneros.actualiza(genero);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede actualizar el género", pe);
    }
  }

  /**
   * Elimina un género del catálogo de géneros
   * @param genero Género a eliminar
   * @throws FachadaException Si no se puede eliminar el género del
   * catálogo de géneros.
   */
  public void elimina(Genero genero) throws FachadaException {
    // Elimina el género del catálogo
    try {
      catalogoGeneros.elimina(genero);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("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 Vector con la lista de canciones
   * @throws FachadaException Si hay un problema al conectarse a la
   * base de datos
   */
  private Vector agregaGeneroCanciones(Vector listaCanciones)
    throws FachadaException  {
    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.elementAt(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.setElementAt(cancion, i);
      }

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

```
    }
  }

 /**
   * Obtiene una lista todas las canciones
   * @return Vector con la lista de todas las canciones
   * @throws FachadaException Si no se puede obtener la lista de canciones
   */
  public Vector consultaCanciones() throws FachadaException {
    // Regresa el vector con la lista de canciones
    try {
      return agregaGeneroCanciones(catalogoCanciones.lista());
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("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 Vector con la lista de todas las canciones con el mismo
   * título.
   * @throws FachadaException Si no se puede obtener la lista de canciones
   */
  public Vector consultaCancionesTitulo(String titulo)
         throws FachadaException {
    // Regresa el vector con la lista de canciones
    try {
      return agregaGeneroCanciones(catalogoCanciones.listaTitulo(titulo));
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede obtener la lista de canciones",
                                 pe);
    }
  }

  ...

  /**
   * Obtiene una lista de los géneros de canciones
   * @return Vector con la lista de los géneros canciones
   */
  public Vector consultaGenerosCanciones() throws FachadaException {
    // Regresa el vector con la lista de géneros de canciones
    try {
      return catalogoGeneros.listaMedio('C');
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede obtener la lista de géneros",
                                 pe);
    }
  }

  /**
   * Obtiene una lista de los géneros de películas
```

```
   * @return Vector con la lista de los géneros películas
   */
  public Vector consultaGenerosPeliculas() throws FachadaException {
    // Regresa el vector con la lista de géneros de películas
    try {
      return catalogoGeneros.listaMedio('P');
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede obtener la lista de géneros",
                                 pe);
    }
  }

...

}
```

Para probar la clase `FachadaBD` y las clases `Canciones` y `Peliculas` en su versión que encapsulan las tablas de base de datos podemos usar la clase de prueba `Prueba3` vista en el Tema 5: Arreglos y Cadenas modificando sólo dos líneas:

- Cambiar la línea:

  ```
  import fachadas.FachadaArreglos;
  ```

  por ésta:

  ```
  import fachadas.FachadaArchivos;
  ```

- Cambiar la línea:

  ```
  IFachada fachada = new FachadaArreglos();
  ```

  por ésta:

  ```
  IFachada fachada = new FachadaArchivos();
  ```

Estos dos cambios son también los únicos que hay que hacerle a la clase de control Control vista en el Tema 7: Desarrollo de Aplicaciones para cambiar el mecanismo de persistencia, obviamente también es necesario sustituir la clase de fachada y las clases `Canciones`, `Peliculas` y `Generos` a sus versiones de archivos.