4. Persistencia de los datos

Hasta ahora hemos estudiado cómo manipular datos que están ubicados en memoria principal (variables y objetos). Este almacenamiento es temporal, es decir, los datos se pierden cuando una variable local queda fuera de alcance, o cuando el programa termina. Asimismo hemos interactuado con los usuarios a través de la entrada y salida estándar de Java, tanto para obtener datos de parte de los usuarios como para entregarle resultados.

En este capítulo trataremos la persistencia de los datos, es decir su almacenamiento a largo plazo. Para logar esto, los computadores usan archivos para almacenar los datos de forma persistente, es decir, para permitir que los datos puedan mantenerse en el tiempo incluso después de que terminan los programas que los crean y/o usan. Los archivos permiten almacenar los datos en memoria secundaria (discos, CD, pendrives, etc.).

4.1. Uso de Archivos en Java

En general, los archivos poseen como atributos un nombre y un contenido. El nombre se compone de dos partes: la primera corresponde a un nombre que identifica al archivo y la segunda que corresponde a la extensión del archivo que identifica el formato del archivo (por ejemplo: “datos.txt”, “planilla.xml”). El contenido de un archivo puede ser cualquier cosa: una carta, una lista de números, etc.

Las operaciones más comunes que se suelen realizar sobre un archivo son:

- Crear un archivo y asignarle un contenido (escribir a un archivo)
- Eliminar un archivo (incluyendo su contenido)
- Cambiar el nombre de un archivo (manteniendo su contenido)
- Sobreescribir un archivo (mantener el nombre, pero cambiar el contenido)
- Obtener el contenido de un archivo (leer un archivo)

Para la realización de dichas operaciones, Java ofrece un set de clases que son revisadas con más detalle a continuación.

4.1.1. Clase File

La clase **File**, es útil para recuperar información acerca de un archivo o directorio de un disco. Esta clase es parte del paquete java.io, por lo que para usarla se debe importar java.io.File. Los objetos de la clase File no abren archivos ni proporcionan herramientas para procesarlos. Sin embargo, se utilizan frecuentemente con objetos de otras clases de java.io para especificar los archivos y/o directorios que van a manipularse.

Entre los constructores de la clase File, existe uno que tiene como argumento un String que especifica el nombre de un archivo o directorio que se asociará con el objeto File. Su sintaxis es:

```
new File(<nombre archivo>)
```

Ejemplo:

```
File f1, f2;
f1 = new File("c:\MisDatos\cartaAMiriam");
f2 = new File("NumerosMagicos.txt");
```

Estas instrucciones crean dos objetos File que se relacionan con los archivos cartaAMiriam (contenido en el directorio c:\MisDatos\) y NumerosMagicos.txt, respectivamente. En el segundo caso, dado que no se especificó un directorio, se espera que el archivo se encuentre en la misma carpeta en la que se haya la clase desde la que se inició la ejecución (y que contiene al método main).

Cuando se crea un objeto File, no se crea el archivo con el que se relaciona. Es decir, si NumerosMagicos.txt no existe como un archivo en disco, la instrucción

```
f2 = new File("NumerosMagicos.txt");
```

no lo creará. Se debe tener en cuenta que un objeto File en un programa Java simplemente representa un archivo que posiblemente exista en el disco, y no garantiza que éste exista. Sin embargo, si el archivo correspondiente ya existe, el objeto File provee métodos que modelan algunas de las operaciones que se pueden realizar sobre un archivo, por ejemplo: borrar un archivo, cambiar nombre a un archivo, chequear si existe un archivo con el nombre especificado.

Para borrar un archivo, la clase File provee el método delete(), tal como se muestra en el siguiente ejemplo:

```
f1.delete();
```

La instrucción del ejemplo anterior borra el archivo cartaAMiriam, en el supuesto que exista en disco.

Para renombrar un archivo la clase File provee el método renameTo(), para chequear si un archivo existe provee el método exists(). Las siguientes instrucciones muestran un ejemplo de su uso:

```
File f3 = new File("Numeros.txt");
if (f3.exists()){
    f2.renameTo(f3);
}
```

Después de estas instrucciones el archivo NumerosMagicos.txt se llamará Numeros.txt en el disco.

4.1.2. Manipulación de los datos de un archivo

La capacidad de escribir o grabar datos en un archivo es abordado en Java por un conjunto de clases, dependiendo de la naturaleza del archivo y de la forma en que se necesita grabar la información.

A continuación se presentan las clases asociadas a la escritura de datos en archivos de texto, los cuales se organizan como un conjunto secuencial de líneas. Cada línea es una cadena de caracteres (string) que finaliza con un caracter especial de fin de línea.

Java proporciona una clase predefinida que modela un flujo de salida (stream) que va a un archivo; se llama **FileOutputStream** (forma parte del paquete java.io, para usarla se debe importar java.io.FileOutputStream). Es decir, por medio de esta clase es posible escribir datos en un archivo. El constructor de esta clase recibe un objeto File como argumento y su sintaxis es la siguiente:

```
new FileOutputStream(<objeto File>);
```

Ejemplo:

```
File f = new File("Numeros.txt");
FileOutputStream fos = new FileOutputStream(f);
```

Estas instrucciones abren el archivo Numeros.txt para que pueda recibir la salida. La referencia al nuevo objeto FileOutputStream se guarda en la variable fos. Justo cuando se crea el objeto FileOutputStream, se crea también el archivo si no existe, o se elimina el contenido si el archivo ya existía.

FileOutputStream modela el flujo de datos como una secuencia de pequeñas unidades de datos, bytes, pero no reconoce su organización en unidades más grandes como líneas o cadenas de caracteres. Así, no ofrece ningún método que pueda producir una salida visible (no existe println o print). La clase ofrece el camino de los datos hacia el archivo, pero no proporciona los métodos que se necesitan para trabajar con los datos.

Para abordar este problema, Java provee una clase llamada **PrintStream** (pertenece al paquete java.io, para usarla se debe importar java.io.PrintStream) que ofrece los métodos print y println, de hecho System.out es un objeto de esta clase. Sin embargo, System.out está asociado a la pantalla por lo que no es útil en este caso. Es necesario, entonces, crear un objeto PrintStream y relacionarlo con un flujo de salida al archivo sobre el que se desea grabar datos. Esto es:

```
PrintStream p = new PrintStream(<objeto FileOutputStream>);
```

Ahora, al invocar los métodos print y println de p, la salida irá al archivo cuyo flujo de salida se indica como argumento en el constructor.

Ejemplo:

```
File archivo = new File("datos.salida");
FileOutputStream flujoArchivo = new FileOutputStream(archivo);
PrintStream destino = new PrintStream(flujoArchivo);

destino.println("Hola, archivo");
```

Las instrucciones anteriores hacen que la cadena “Hola, archivo” se escriba en el archivo datos.salida. Si datos.salida no existe, se crea. Si ya existe, la cadena reemplaza el contenido que hubiere.

Una manera abreviada de crear un objeto de la clase PrintStream que esté asociado a un archivo de texto es usando otro de los constructores de esta clase. La sintaxis de dicho constructor es:

```
new PrintStream(<nombre archivo>)
```

El constructor anterior lanza excepciones del tipo FileNotFoundException.

Ejemplo:

```
PrintStream destino = new PrintStream("datos.salida");
destino.println("Hola, archivo");
```

Este método abreviado resulta útil cuando no se requiere de las funcionalidades provistas por las clases File y FileOutputStream.

Al igual que para escribir datos en un archivo, para leer datos desde un archivo que ya existe, Java proporciona una serie de clases, de entre las cuales se debe escoger las que resultan pertinentes dependiendo de la naturaleza del archivo y de la forma en que se necesita recuperar la información.

Para leer a datos desde archivos de texto, que se organizan como un conjunto secuencial de líneas, se necesita un objeto equivalente a System.in, pero que se pueda asociar a un archivo. El tipo de objeto que se precisa es **FileInputStream** (se encuentra en el paquete java.io, debiéndose importar java.io.FileInputStream). Uno de los constructores de esta clase recibe un objeto File como argumento, y su sintaxis es:

```
new FileInputStream(<objeto File>);
```

Una vez que se tiene una referencia a un objeto FileInputStream, se puede utilizar dicha referencia de la misma forma que antes se hizo con System.in para leer desde teclado. Esto es, se utilizará la referencia para construir un objeto **InputStreamReader**, el que a su vez se utilizará para construir un objeto **BufferedReader**, capaz este último de ofrecer las líneas de la entrada como objetos de tipo String.

Ejemplo:

El siguiente código muestra en pantalla dos líneas almacenadas en un archivo de texto.

```
File archivo = new File("lista.secreta");
FileInputStream fs = new FileInputStream(archivo);
InputStreamReader isr = new InputStreamReader(fs);
BufferedReader archivoDeTexto = new BufferedReader(isr);

String linea;
linea = archivoDeTexto.readLine();              // Lee la primera línea del archivo de texto
System.out.println(linea);                      // Despliega el contenido de la línea leída desde el archivo
System.out.println(archivoDeTexto.readLine());  // Lee y despliega el contenido de la segunda línea
```

fs es equivalente a System.in por lo que es posible usar este objeto en forma similar a aquel para acceder a los datos provenientes, en este caso, del archivo de texto llamado lista.secreta.

Por otro lado, la clase **FileReader** (que forma parte del paquete java.io, debiéndose importar java.io.FileReader) ofrece un modo breve de abrir y leer un archivo de texto. Esta clase cuenta con tres constructores, la sintaxis de dos de ellos es:

```
new FileReader(<objeto File>);
new FileReader(<nombre del archivo>);
```

El último constructor obvia la creación de un objeto File. Esto implica un ahorro de memoria, pero impide el uso de los métodos que provee la clase File. Cuál usar, dependerá de las funcionalidades con las que se necesita contar.

El objeto FileReader se puede utilizar para construir un objeto BufferedReader, el cual, como ya se ha señalado antes, cuenta con las operaciones que se precisan.

A continuación se presenta el mismo ejemplo anterior, pero usando el segundo constructor de la clase FileReader.

```
BufferedReader archivoDeTexto = new BufferedReader(new FileReader("lista.secreta"));

// Lee y despliega el contenido de la primera línea del archivo
System.out.println(archivoDeTexto.readLine());

// Lee y despliega el contenido de la segunda línea del archivo
System.out.println(archivoDeTexto.readLine());
```

La clase **Scanner** (que forma parte del paquete java.util, debiéndose importar java.util.Scanner) vista anteriormente para leer texto desde la entrada estándar, también sirve para realizar lecturas de texto desde archivos. Tiene la capacidad de leer un archivo y convertir el texto en tipos de datos primitivos (byte, int, float, etc.) y String utilizando métodos de acuerdo al tipo de dato.

Esta clase cuenta con dos constructores, estos son:

```
new Scanner(<objeto FileInputStream>);
new Scanner(<objeto FileReader>);
```

Los objetos recibidos como parámetro en estos constructores (explicados previamente) permiten asociar el escaner al archivo a leer, luego la lectura desde el archivo se realizará de forma similar a la utilizada en la entrada estándar.

A continuación se presenta el mismo ejemplo anterior, pero usando el segundo constructor de la clase Scanner.

```
Scanner archivoDeTexto = new Scanner(new FileReader("lista.secreta"));

// Lee y despliega el contenido de la primera línea del archivo
System.out.println(archivoDeTexto.nextLine());

// Lee y despliega el contenido de la segunda línea del archivo
System.out.println(archivoDeTexto.nextLine());
```

Ejemplo 1:

A continuación se presenta un ejemplo en el que se lee un archivo llamado "Numeros.txt" que contiene enteros positivos, donde la coma (",") se usa como delimitador. La cantidad de caracteres por línea es variable y el número de líneas es desconocido. Se leen los números almacenados en el archivo y se despliegan en pantalla, uno por línea. Al final se despliega la suma de los mismos.

El ejemplo utiliza el método abreviado para la creación y apertura del objeto asociado al archivo con los datos.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.StringTokenizer;
import java.lang.Integer;

public class LeeNumeros {
   public static void main(String[] args) {

      try {
         BufferedReader archivo = new BufferedReader(new FileReader("Numeros.dat"));

         String linea, numeroComoString;
         int suma = 0;

         // Lee la primera línea fuera del ciclo
         linea = archivo.readLine();

         // Lee y procesa las restantes líneas
         System.out.println("Los números en el archivo son: ");

         while (linea != null) { // Mientras hayan líneas con datos
            StringTokenizer stk = new StringTokenizer(linea, ", ");

            while (stk.hasMoreTokens()) { // Por cada token
               numeroComoString = stk.nextToken();
               System.out.println("--> " + numeroComoString);
               suma += Integer.parseInt(numeroComoString);
            }
            linea = archivo.readLine();
         }
         System.out.println("\nSuma de los números leídos: " + suma);
      } catch (FileNotFoundException e) {
         System.out.println("Error, archivo no encontrado");
      } catch (IOException e) {
         System.out.println("Error de entrada/salida al intentar leer/grabar en el archivo");
      }
   }
}
```

Usará "," y el espacio como caracteres delimitadores.

Ejemplo 2

A continuación se presenta el mismo ejemplo anterior pero utilizando sólo la clase FileReader y Scanner, ésta última tanto para leer desde el archivo como para leer los números individuales separados por el delimitador “,”.

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Scanner;

public class LeeNumerosScanner {
   public static void main(String[] args) {
      try {
         Scanner archivo = new Scanner(new FileReader("Numeros.dat"));
         int numero;
         int suma = 0;
         System.out.println("Los números en el archivo son: ");


         while (archivo.hasNextLine()) { // Mientras hayan líneas con datos
            Scanner linea = new Scanner(archivo.nextLine())
                  .useDelimiter("[,]"); // Creamos un nuevo escáner por línea

            while (linea.hasNextInt()) { // Por cada número de la línea
               numero = linea.nextInt();
               System.out.println("--> " + numero);
               suma += numero;
            }
         }
         System.out.println("\nSuma de los números leídos: " + suma);
      } catch (FileNotFoundException e) {
         System.out.println("Error, archivo no encontrado");
      }
   }
}
```

Ejemplo 3:

En seguida se presenta un ejemplo completo que incluye lectura desde un archivo de texto, creación de objetos a partir de esos datos, invocación de los métodos provistos por tales objetos para generar los datos solicitados en el problema y el almacenamiento de estos datos en un archivo de texto.

El problema concreto dice relación con el procesamiento de los datos incluidos en los Carnés de Votación de un conjunto de ciudadanos. Para ello se presenta la Figura 4.1 que muestra la gráfica UML de la clase CarnetVotación utilizada para resolver el problema.

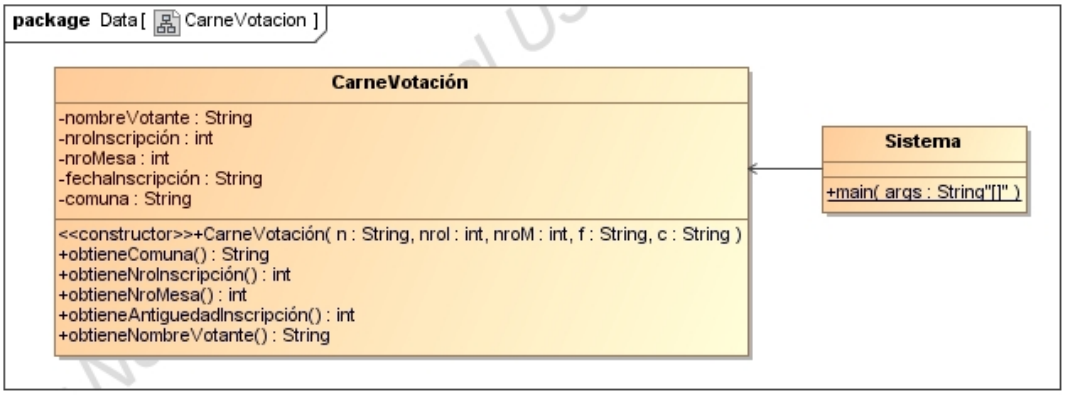

Figura 4.1. Representación UML de la clase CarneVotación.

En particular se necesita implementar una clase llamada Sistema que realice lo siguiente:

- Leer una secuencia de datos correspondiente a carnets de votación de distintas comunas. Los datos serán ingresados desde un archivo llamado entrada.txt, en cada línea se indican los datos asociados a un carné, separados por comas: Nombre votante, número de inscripción, número de mesa, fecha de inscripción y comuna.
- Crear un vector en el cual se almacenen objetos del tipo CarneVotación, a partir de los datos leídos.
- Generar y grabar en un archivo llamado resultado.txt la siguiente información sobre los votantes:
  a) Cantidad de votantes.
  b) Dada una comuna y un número de mesa (leídos desde teclado), cantidad de votantes de dicha mesa.
  c) Nombre de los votantes que tengan una antigüedad de inscripción superior a 10 años.
  d) Cantidad de votantes cuya comuna comienza con el caracter ‘C’ o ‘c’.

Solución:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.io.FileReader;
import java.io.IOException;
import java.util.StringTokenizer;
import java.util.Vector;

public class Sistema {
   public static void main (String[] args) {
      try {
         // Define variable para lectura desde teclado
         BufferedReader tcld = new BufferedReader(new InputStreamReader(System.in));
         // Define variable para grabar en el archivo "resultado.txt"
         PrintStream archSalida = new PrintStream("resultado.txt");
         // Define variables para lectura desde el archivo "entrada.txt"
         BufferedReader archEntrada = new BufferedReader(new FileReader("entrada.txt"));
         StringTokenizer stk;
         String nombre, comuna, fechaIns, linea ="";
         int nroMesa,nroIns;
         // Define vector que almacenará los objetos CarnéVotación
         Vector<CarnéVotación> votantes = new Vector<CarnéVotación>();

         // Lee datos de los carnés de votación
         linea = archEntrada.readLine();  // Lee primera línea fuera del ciclo

         while (linea != null){
            // Recupera los datos (tokens) desde línea
            stk = new StringTokenizer(linea,",");
            nombre = stk.nextToken();
            nroIns = Integer.parseInt(stk.nextToken());
            nroMesa = Integer.parseInt(stk.nextToken());
            fechaIns = stk.nextToken();
            comuna = stk.nextToken();
            // Crea el objeto correspondiente y lo almacena en el vector
            votantes.addElement(new CarnéVotación(nombre,nroIns,nroMesa,fechaIns,comuna));
            // Lee siguiente línea del archivo
            linea = archEntrada.readLine();
         }

         // Imprime cantidad total de votantes leídos desde el archivo
         archSalida.println("La cantidad de votantes leídos es: " + votantes.size());

         // Determina nro. de votantes de una cierta mesa en una determinada comuna
         int totalVotantes=0;
         System.out.print("\nIngrese comuna del votante: ");
         comuna = tcld.readLine();

         do {
            System.out.print("\nIngrese número de mesa: ");
            nroMesa = Integer.parseInt(tcld.readLine());
```

```
            if (nroMesa <= 0) {
                System.out.println("\nError! número de mesa debe ser mayor que 0");
            }
        } while (nroMesa <= 0);

        CarnéVotación votante;
        for (int i=0; i<votantes.size(); i++){
            votante = votantes.get(i);
            if (votante.obtieneComuna().equalsIgnoreCase(comuna) &&
                        votante.obtieneNroMesa() == nroMesa) {
                totalVotantes++;
            }
        }

        archSalida.println("\nEl total de votantes de la mesa: " + nroMesa + " en la comuna: " +
                    comuna + " es: " + totalVotantes);

        // Votantes con antigüedad > 10 años
        archSalida.println("\nVotantes con antigüedad mayor a 10 años: ");
        for (int i=0; i<votantes.size(); i++){
            votante = votantes.get(i);
            if (votante.obtieneAntiguedadInscripción() > 10) {
                archSalida.println("- " + votante.obtieneNombreVotante());
            }
        }

        // Determina nro. de votantes cuyas comunas comienzan con "C" o "c"
        totalVotantes = 0;
        for (int i=0; i<votantes.size(); i++){
            votante = votantes.get(i);
            if ((votante.obtieneComuna()).charAt(0)=='C' ||
                        (votante.obtieneComuna()).charAt(0)=='c') {
                totalVotantes++;
            }
        }
        archSalida.println("\nEl total de votantes cuya comuna empieza con C es: " + totalVotantes);
    }

    catch(IOException e) {
        System.out.println("Error de entrada/salida al intentar leer/grabar en el archivo");
    }

    catch(NumberFormatException e) {
        System.out.println("Error de formato en un dato numérico");
    }
  }
}
```

Si el archivo entrada.txt contiene los siguientes datos:

```
Juan Rozas Leal,10,03,10-10-2000,Chillán
Loreto Rivas León,5,9,12-12-1999,San Carlos
Nancy Segers Lira,212,3,01-01-1991,Contulmo
Adrián Arenas Lazo,177,5,03-03-1993,Los Ángeles
```

El archivo resultado.txt contendrá lo siguiente:

```
La cantidad de votantes leídos es: 4

El total de votantes de la mesa: 5 en la comuna: Los Ángeles es: 1

Votantes con antigüedad mayor a 10 años:
- Nancy Segers Lira
- Adrián Arenas Lazo

El total de votantes cuya comuna empieza con C es: 2
```

Ejercicio:

Modifique el ejemplo 3 remplazando la clase BufferedReader y StringTokenizer por la clase Scanner.