11 | Base de datos persistente

Hasta ahora, todo el trabajo que has hecho se mantiene en la memoria de la computadora mientras tu programa esté corriendo. Cada vez que abres tu lista de contactos esta vacía y debes agregarlos a mano, pero cuando termina su ejecución desaparecen los contactos que tanto trabajo te costó agregar. Lo que harás en esta práctica es almacenar los registros de tu `BaseDeDatosAgenda` para que la próxima vez que la ejecutes, tu información siga ahí.

Meta

Que el alumno aprenda a escribir y leer información del disco duro.

Entrada y salida en Java

Definición 11.1: Flujo

Un **flujo** es una secuencia de datos. Un programa utiliza un **flujo de entrada** para leer datos desde una fuente, uno a la vez, y un **flujo de salida** para enviar datos a un destino, también uno a la vez. *I/O Streams* 2016

`Java` distingue entre dos formas de manejar esa información: binaria y como caracteres. Un grupo de clases lee y escribe bytes, mientras que otro está preparado para trabajar con diferentes codificaciones de texto. Las clases más comunmente utilizadas de ambos paquetes están ilustradas en Figura 11.1.

Para esta práctica necesitarás estar consultando constantemente la documentación de la API de `Java`, así que ten la dirección a la mano: `http://docs.oracle.com/javase/8/docs/api/`.

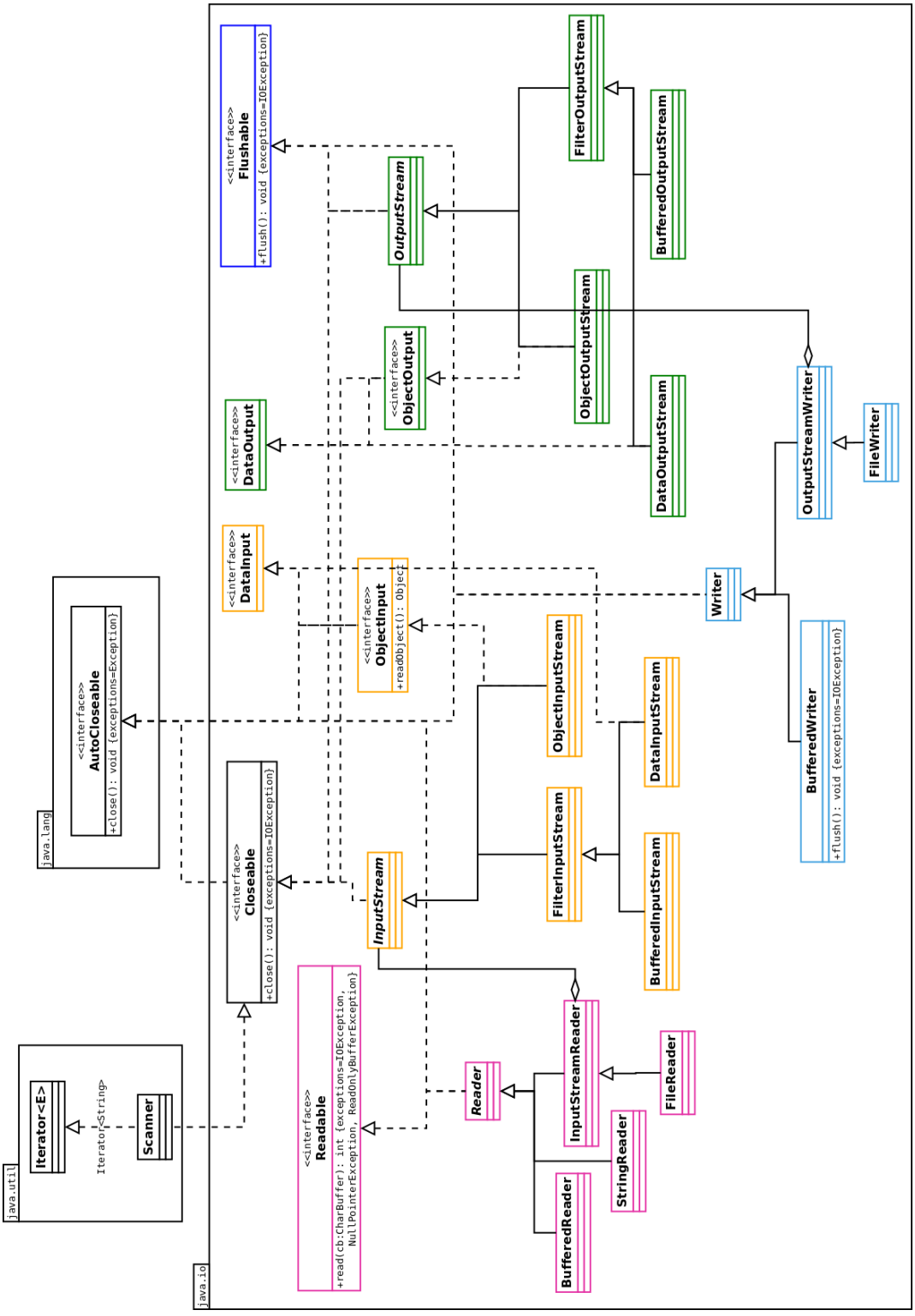

Figura 11.1 Diagrama de clases UML de algunas clases para entrada/salida en `Java`.

Recibiendo entrada desde la terminal

Si quieres hacer un programa con interfaz de usuario basada en texto, como el `bash` que se ejecuta en la terminal, necesitas acceder al flujo de entrada de la terminal. `Java` hace esto a través del objeto estático `System.in`. Éste es un `InputStream` y lee bytes. No queremos tener que decodificar cada caracter que teclee nuestro usuario byte por byte. El tipo de objeto que puede ayudarnos con ese trabajo es un `Scanner`. Si al `Scanner` le damos un flujo de entrada cuando lo construímos, él se encargará de hacer la decodificación por nosotros y devolvernos cadenas de caracteres (`String`). A continuación se incluye un pequeño ejemplo de cómo utilizarlo para este fin.

Actividad 11.1

Revisa la documentación oficial de `java.util.Scanner`. ¿Qué es un objeto tipo `File`?

```java
import java.util.Scanner;

/**
 * Demo básico sobre el uso de la clase Scanner.
 * @author blackzafiro
 */
public class DemoScanner {

  /**
    * Recibe una línea de texto desde el teclado y
    * reimprime el mensaje que recibió, hasta que el usuario
    * se despida.
    * @param args
    */
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    while(s.hasNext()) {
      String linea = s.nextLine();
      System.out.println("Eco: " + linea);
      if(linea.equals("Adios")) break;
    }
    s.close();
  }

}
```

Actividad 11.2

Copia y ejecuta el código anterior. Para hacerlo más rápido que con `ant`, cópialo en un archivo `DemoScanner.java` e invoca al compilador escribiendo `javac DemoScanner.java`. El archivo `.class` se generará ahí mismo. Ejecútalo con

`java DemoScanner`.

Cambia la línea 18 para usar el método `next()` en lugar de `nextLine()`. ¿Qué hace ahora el programa?

Leyendo archivos de texto

Un `BufferedReader` se parece a un `Scanner`, pero se especializa en archivos. Hay cosas que puedes hacer en un archivo, que no podrías hacer en terminal, como regresar a la línea anterior o leer todo el archivo en una sola pasada. El demo siguiente lee un archivo de texto llamado `texto.txt` y lo imprime línea por línea. Asegúrate de crear uno al lado de este código, para que puedas probarlo. Observa que si olvidas hacerlo, este método lanza una excepción, para avisar que el archivo no está y el método `main` la cacha, para convertirla en un mensaje más amigable con el usuario.

```
1  import java.io.BufferedReader;
2  import java.io.FileNotFoundException;
3  import java.io.FileReader;
4  import java.io.IOException;
5
6  /**
7   * Demo básico sobre el uso de FileReader y BufferedReader.
8   * @author blackzafiro
9   */
10 public class DemoBufferedReader {
11   /**
12     * Lee un archivo de texto e imprime su contenido línea por lí
          ↪ nea.
13     * @param args
14     */
15   public static void main(String[] args) {
16     try {
17       BufferedReader in = new BufferedReader(new FileReader("
            ↪ texto.txt"));
18       String line;
19       while((line = in.readLine()) != null) {
20         System.out.println(line);
21       }
22       in.close();
23     } catch (FileNotFoundException e) {
24       System.err.println("No se encontró el archivo texto.txt ¿
            ↪ Olvidaste crearlo?");
25     } catch (IOException ioe) {
26       System.err.println("Error al leer el contenido de texto.txt
            ↪ ");
27     }
28   }
29
30 }
```

Escribiendo archivos de texto

Para escribir archivos de texto utilizaremos un `PrintStream`. Esta clase desciende de `OutputStream` y de hecho la has usado frecuentemente ¿Recuerdas a `System.out`? Pues bien, éste es un objeto de tipo `PrintStream`. Así que siéntete como en casa. El ejemplo de abajo indica cómo crear uno que dirija la salida a un archivo, en lugar de a la terminal. Observa que la sintaxis del `try` es diferente: la creación del objeto que puede lanzar las excepciones se encuentra entre paréntesis. Esto ayudará también a que el objeto se cierre solo al terminar de ejecutar el bloque de código dentro del `try`. También puedes utilizar esta notación con los otros flujos.

```
1  import java.io.FileNotFoundException;
2  import java.io.PrintStream;
3
4  /**
5   * Demo básico sobre el uso de PrintStream.
6   * @author blackzafiro
7   */
8  public class DemoPrintStream {
9    /**
10    * Escribe el texto indicado en un archivo.
11    * @param args
12    */
13   public static void main(String[] args) {
14     String nombreArchivo = "Salida.txt";
15     try (PrintStream fout = new PrintStream(nombreArchivo)) {
16       fout.println("Inicio");
17       fout.format("Línea␣%d\n", 1);
18       fout.println("Fin");
19     } catch (FileNotFoundException fnfe) {
20       System.err.println("No␣se␣encotró␣el␣archivo␣" +
             ↪ nombreArchivo + "␣y␣no␣pudo␣ser␣creado");
21     } catch (SecurityException se) {
22       System.err.println("No␣se␣tiene␣permiso␣de␣escribir␣en␣el␣
             ↪ archivo");
23     }
24   }
25
26 }
```

Actividad 11.3

Copia y ejecuta el código anterior.

Revisa la documentación de `PrintStream`. Observa que si ejecutas el programa otra vez, se borra el contenido del archivo y se sobreescribe. Si quieres abrir un archivo para agregarle cosas, en lugar de borrarlo y escribir de nuevo, necesitarás usar un `FileWriter` o un `FileOutputStream` e indicar en el constructor que

quieres usar la opción `append`.

Ejercicios

Utiliza nuestro conocido `System.out` y la clase `Scanner` para crear una interfaz de texto con tu usuario.

1. Ahora la clase de uso servirá para crear y manejar la interfaz de texto. Crea un objeto `UsoBaseDeDatos` al inicio de tu método `main`.

2. En main, crea un ciclo donde el usuario pueda elegir entre:
   - Crear una base datos.
   - Cargar de disco.
   - Guardar la agenda.
   - Agregar un registro.
   - Buscar por nombre.
   - Buscar por teléfono.
   - Imprimirla toda.

   Puedes agregarle un método a tu clase `UsoBaseDeDatos`, que se encargue de imprimir este menú, y mandarlo llamar cuando sea necesario.

   Prueba que el menú funcione, aunque no haga nada. A continuación crearás métodos auxiliares para que tu objeto tipo `UsoBaseDeDatos` pueda atender las diferentes solicitudes.

3. La clase `UsoBaseDeDatos` deberá tener un método que pida al usuario el nombre, dirección y teléfono para un registro nuevo y devuelva un objeto de tipo `RegistroAgenda`, inicializado con los datos correspondientes. Así mismo, tu clase `BaseDeDatosAgenda` necesitará un método para agregar el registro.

   Cuando el usuario seleccione la opción "agregar un registro", usarás el método nuevo para pedir los datos al usuario y luego el método de la agenda para agregar el registro nuevo.

4. Para buscar un registro, pide el nombre o el teléfono usando `Scanner`. Luego usa los métodos de tu base de datos para encontrar la primer coincidencia. Imprime en pantalla el resultado de la búsqueda.

5. Para guardar la agenda.

   a) Crea en `RegistroAgenda` un método que reciba como parámetro un objeto de tipo `PrintStream` y que con él escriba su nombre, dirección y teléfono en líneas distintas. Imagina que estuvieras imprimiendo el contenido del registro en pantalla pero, en lugar de `System.out` usa el objeto que te pasan como parámetro (no hagas ninguna otra cosa con él).

b) Para guardar toda la agenda vamos a hacer ahora un poco de trampa.
Observa de nuevo el esquema del diseño de la base de datos Figura 10.4 y tu código para buscar por nombre y teléfono. Te darás cuenta de que el código de la función `esEste` se ejecuta sobre cualquier elemento de la lista, mientras que éste no devuelva `true` (dado que al devolver `true` se deja de recorrer el resto de la lista).
Extiende nuevamente a la interfaz `Buscador`, pero ahora tu clase se encargará de guardar los elementos de la lista en un archivo. Llámala `GuardaRegistro`.
   1) El constructor de tu clase debe recibir un `PrintStream` ya abierto como parámetro y lo guardará en un atributo.
   2) El método `esEste` utilzará al `printStream` para escribir con él el contenido del registro. Para ello manda llamar el método para guardar el registro que reciba como parámetro[1] y devuelve `false` siempre.

c) Agrega a la clase `BaseDeDatosAgenda` el método:

```
1 public void guardaAgenda(String nombreArchivo) {
2     ...
3 }
```

En este método crearás el `PrintStream` con la dirección indicada en `nombreArchivo`. Esta cadena deberá contener la dirección absoluta o relativa del archivo donde se guardará la agenda[2]. Al crear el `PrintStream` se abrirá el archivo correspondiente para escritura.
Luego usarás una instancia de la clase `GuardaRegistro` para guardar toda la base de datos, pasando como parámetro a su constructor el `PrintStream` que acabas de crear. Necesitarás del método `encuentra` de `ManejadorDeLista`. El procedimiento será casi idéntico a lo que hacías para buscar registros. Ojo: no debes modificar **nada** que se encuentre en el paquete `util`.

d) La interfaz de usuario requerirá un método correspondiente que solicite al usuario la dirección del archivo donde desea guarda la agenda y mande llamar a `guardaAgenda`.

e) Prueba tu método. Dado que utilizaste objetos que crean archivos de texto, debes poder abrir el archivo creado con cualquier editor de texto y leer los datos que fueron guardados.

6. Agrega un método `carga(String nombreArchivo)` que cree un flujo de entrada (puedes utilizar `BufferedReader` o `Scanner`), que lea los datos de tus archivos como líneas de texto y las utilice para insertar los registros en ese archivo a la agenda. Reutiliza los métodos que ya programaste.

7. Prueba todo tu programa.

[1] Recuerda hacer los castings necesarios.
[2] Recuerda incluir esta explicación en tu documentación.

PREGUNTAS

1. ¿Cómo preferirías que se llamara la interfaz `Buscador` para que su nombre fuera más acorde a todos los usos que le hemos dado? ¿Qué hay del método `encuentra`? Describe qué cambios sería necesario hacerle al código.