Arreglos y Archivos (Parte II)

Archivos: escribiendo y leyendo información.

Competencias

- Aplicar métodos de la clase File.

Introducción

Manejarás, conocerás e implementará la creación de archivos y ficheros. En este nuevo mundo es importante dar el enfoque requerido ya que la información que antes se manejaba como prueba ahora quedará persistida en archivos.

Conoceremos e implementaremos nuevas clases de Java, además aprenderemos a leer códigos paso a paso para comprender la lógica de Java. Como consejo, debes recordar que al persistir la información se puede manipular cuantas veces sea necesario.

Estructura de archivos y ficheros.

Para crear un directorio o fichero se debe especificar la ruta y nombre como parámetro de entrada. En el siguiente ejemplo se ilustra cómo se pasan estos parámetros:

Importación

```
import java.io.File;
```

Directorio

```
File directorio = new File("src/carpeta"); // ruta donde quedará mi carpeta
```

Fichero

```
File archivo = new File("src/carpeta/texto.txt"); // ruta donde quedará mi archivo
```

File y sus métodos

File es la principal clase para trabajar con archivos ya que nos permite crear archivos y directorios a través de sus métodos.

A partir del ejemplo anterior, donde se declaró la estructura de un archivo y fichero, veremos sus principales métodos:

Método `exists()`

Este método se debe mantener presente a la hora de crear archivos ya que validará si el directorio o fichero que se creará existe o no dentro del proyecto.

Este método retorna `false` cuando no existe y `true` cuando exista el archivo o fichero.
Siempre se utiliza en operadores condicionales.

```
directorio.exists()
```

Método `mkdirs()`

Con este método podemos crear los directorios dentro de nuestro proyecto, lo debemos considerar para utilizarlo en condiciones de validaciones.

```
directorio.mkdirs()
```

Es importante que antes de crear un directorio se valide si este existe previamente ya que, si no se valida, su existencia se sobreescribirá en el directorio anterior y esto puede hacer que se pierda toda la información contenida dentro de él.

Ejemplo de cómo se ve un directorio creado en Eclipse:

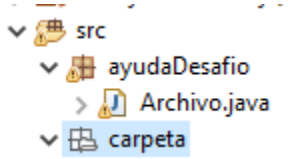

Imagen 1. Directorio en Eclipse.

Método `createNewFile()`

Este método nos permite crear un archivo físico en nuestro programa. Debes considerar siempre que, si el archivo ya está creado y no se realizan las validaciones de preexistencia, este sobrescribirá toda la información sobre el archivo.

```
archivo.createNewFile();
```

Ejemplo de cómo se ve un archivo creado en Eclipse.

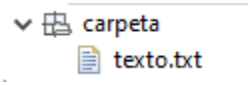

Imagen 2. Archivo creado.

Ejercicio guiado: Crear directorio

Crear un directorio usando métodos de la clase File.

Paso 1: Crear un un método llamado “crearDirectorio” que recibirá como parámetro el nombre del directorio.

```
import java.io.File;

public static void crearDirectorio(String nombre) {

}
```

Paso 2: Instanciar la clase File para crear el directorio.

```
public static void crearDirectorio(String nombre) {
	File directorio = new File("src/"+nombre);
}
```

Paso 3: Se requiere validar que el directorio no existe para poder crearlo. Si el directorio no existe, lo creamos con el método `mkdir()`.

```
public static void crearDirectorio(String nombre) {
	File directorio = new File("src/"+nombre);
	if (directorio.exists() == false) {
		directorio.mkdir();
	}
}
```

Paso 4: Llamar al método `crearDirectorio` dentro del método `main`, pasándole como parámetro el nombre que le daremos al directorio.

```
public static void main(String[] args) {
      crearDirectorio("directorio");
}
```

Solución Final

El código completo quedaría de la siguiente manera:

```
import java.io.File;

      public static void main(String[] args) {
            crearDirectorio("directorio");
      }

      public static void crearDirectorio(String nombre) {
            File directorio = new File("src/"+nombre);
            if (directorio.exists() == false) {
                  directorio.mkdir();
            }
      }
```

Ejercicio propuesto (1)

- Se solicita crear un método llamado “Crear fichero” que recibe como parámetro de entrada el nombre del fichero y el nombre del directorio.
- Se requiere validar que el directorio no exista para completar la creación del directorio.

Escritura de un archivo BufferedWriter - FileWriter

Competencias

- Comprender Clase FileWriter para el uso de los métodos en la construcción de los archivos.
- Comprender Clase BufferedWriter para el uso de los métodos en la escritura de archivos.

Introducción

Aprenderemos a escribir en archivos físicos utilizando las mejores prácticas dentro de la programación en Java, para esto utilizaremos las clases `FileWriter` y `BufferedWriter`, las cuales optimizan el uso de los archivos.

Todo el contenido estudiado en este capítulo nos servirá para manejar, conocer y comprender el uso práctico de los archivos que pueden ir desde un texto plano hasta un PDF.

Clase FileWriter

Esta clase es usada para escribir caracteres en archivos. Su método `write()` permite escribir caracteres o strings a un fichero. Esta clase normalmente está envuelta en objetos Writer de más alto nivel, como BufferedWriter.

Para crear un FileWriter, luego de realizar la importación `import java.io.FileWriter;`, se necesita un String como ruta de archivo o una clase File. Para este caso, y siguiendo la línea del curso, utilizaremos un File:

```
File archivo = new File("src/carpeta/texto.txt");
FileWriter fileW = new FileWriter(archivo);
```

Clase BufferedWriter

Esta clase es usada para hacer clases de bajo nivel como FileWriters de una manera más eficiente y más fácil de usar. Comparado con las clases FileWriter, los BufferedWriters escriben relativamente grandes cantidades de información en un archivo, lo que minimiza el número de veces que las operaciones de escritura de archivos, que son operaciones más lentas, se llevan a cabo. La clase BufferedWriter también provee un método llamado `newLine()` creando separadores de línea específicos de la plataforma de manera automática.

Para crear un BufferedWriter, luego de realizar la importación `import java.io.BufferedWriter;`, se necesita un objeto `FileWriter`:

```
File archivo = new File("src/carpeta/texto.txt");
FileWriter fileW = new FileWriter(archivo);
BufferedWriter bufferedWriter = new BufferedWriter(fileW);
```

Para escribir utilizaremos el método `write("texto")`, donde el texto es lo que se escribirá en el archivo:

```
bufferedWriter.write("texto");
```

Para que se guarde la información debemos cerrar el archivo, esto hace que guarde automáticamente los cambios. Utilizaremos el método `close()`:

```
bufferedWriter.close();
```

Lectura de un archivo BufferedReader - FileReader

Competencias

- Construir Clase FileReader a través de su constructor para lectura de archivos.
- Aplicar la Clase BufferedReader y sus métodos para lectura de archivos.

Introducción

Conoceremos las Clases BufferedReader y FileReader ya que nos permitirá leer archivos físicos de grandes volúmenes de información y manejar el trato de estos. Esto nos servirá para trabajar en aplicaciones de grandes empresas o en cualquier aplicativo donde se necesite almacenar archivos físicos.

Clase FileReader

Esta clase es usada para leer archivos de caracteres. Su método `read()` es usado a bajo nivel, permitiendo leer caracteres de manera singular.

Para crear un FileReader, luego de realizar la importación `import java.io.FileReader;`, se necesita un String como ruta de archivo o una clase File. Para este caso, y siguiendo la línea del curso, utilizaremos un File:

```
File archivo = new File("src/carpeta/fichero.txt");
FileReader fr = new FileReader(archivo);
```

Clase BufferedReader

Esta clase es usada para hacer clases Reader de bajo nivel como FileReader pero de una manera más eficiente y más fácil de usar. Comparado con los FileReader, los BufferedReader leen relativamente grandes cantidades de un archivo a la vez, y mantienen esta información en el buffer (memoria de la Java virtual machine). Cómo está precargada hace que la información se cargue en memoria y sea más fácil de leer y manejar.

```
import java.io.FileReader;
import java.io.BufferedReader;
```

```
File archivo = new File("src/carpeta/fichero.txt");
FileReader fr = new FileReader(archivo);
BufferedReader br = new BufferedReader (fr);
```

Ejercicio guiado: Crear y escribir en un archivo

Vamos a crear un método llamado `crearFile` el cual creará un directorio físico llamado "miDirectorio", dentro de este directorio crearemos un archivo llamado fichero.txt, en este archivo escribiremos texto.

Seguiremos todos los pasos aprendidos en la lección.

Paso 1: Crear el directorio.

```
import java.io.*;

public static void crearFile()} // comienzo del método
   File directorio = new File("src/carpeta");
   directorio.mkdirs();
```

Paso 2: Crear el fichero o archivo.

Creamos el objeto File llamado objeto con la ruta.

```
File archivo = new File("src/carpeta/texto.txt");
archivo.createNewFile();
```

Paso 3: Crear el FileWriter y BufferedWriter.

- Creamos el objeto FileWriter con un Archivo File.
- Creamos el objeto BufferedWriter con un Archivo FileWriter .

```
FileWriter fileW = new FileWriter(archivo);
BufferedWriter bufferedWriter = new BufferedWriter(fileW);
```

Paso 4: Escribir y cerrar el archivo.

- Utilizamos el método write para escribir en el archivo.
- Hacemos un salto de línea con el método newLine().
- Cerramos el archivo con el método close().

```
bufferedWriter.write("texto 1");
bufferedWriter.write("texto 2");
bufferedWriter.newLine();
bufferedWriter.close();

} // cierre del método
```

Ejercicio propuesto: (2)

Siguiendo con el ejercicio guiado anterior:

- Agregar un nuevo método llamado `lectura()`.
- Este método leerá toda la información guardada dentro del archivo creado en el ejercicio anterior.

Soluciones ejercicios propuestos

Solución ejercicio propuesto (1)

1. Importamos los métodos del paquete java.io.

```
import java.io.*;
```

2. Crear método llamado crearFichero (String nombre, String fichero).

```
public static void crearFichero(String nombre,String fichero) {
}
```

3. Inicializamos la variable local “directorio” con la ruta del directorio.

```
public static void crearFichero(String nombre,String fichero) {
      File directorio = new File("src/"+nombre);
}
```

4. Preguntamos con If si el directorio no existe.

```
public static void crearFichero(String nombre,String fichero) {
      File directorio = new File("src/"+nombre);
      if(directorio.exists() ==false) {
      }
}
```

5. Creamos el directorio.

```
public static void crearFichero(String nombre,String fichero) {
            File directorio = new File("src/"+nombre);
            if(directorio.exists() ==false) {
                  directorio.mkdir();
            }
      }
```

6. Creamos otra variable local llamado “archivo” con la ruta del directorio.

```
public static void crearFichero(String nombre,String fichero) {
            File directorio = new File("src/"+nombre);
            if(directorio.exists() ==false) {
                  directorio.mkdir();
                  File archivo = new File(directorio+fichero+".txt");
            }
      }
```

7. Utilizamos el método `createNewFile()` para crear el archivo.

```
public static void crearFichero(String nombre,String fichero) {
            File directorio = new File("src/"+nombre);
            if(directorio.exists() ==false) {
                  directorio.mkdir();
                  File archivo = new File(directorio+fichero+".txt");
                  archivo.createNewFile();
            }
      }
```

Solución ejercicio propuesto (2)

1. Importamos los métodos del paquete java.io.

```
import java.io.*;
```

2. Ruta del directorio a buscar.

```
public static void lectura() {
File archivo = new File("src/carpeta/fichero.txt");
```

3. Crear las clases FileReader y BufferedReader.

    a. Crear una clase FileReader con un archivo File.
    b. Crear una clase BufferedReader con un objeto FileReader.

```
FileReader fr = new FileReader(archivo);
BufferedReader br = new BufferedReader(fr);
```

4. Leemos toda la información del archivo y la dejamos en una variable.

```
String data = br.readLine();
```

5. Con el ciclo While recorremos todo el archivo hasta que la variable data no tenga datos.

```
while (data != null) {
      System.out.println(data);
      data = br.readLine();
}
```

6. Cerramos el archivo y el método.

```
br.close();
} // Cierre método lectura
```