Archivos

Contenido

Archivos

- ¿Qué es un sistema de archivos?
- Funciones del sistema de archivos.
- ¿Qué es un path?
- Rutas relativas vs. rutas absolutas.

Archivos

- Gestión de archivos (tradicional).
  - La clase File.
  - La clase FileWriter.
  - La clase FileReader.
  - La clase BufferedWriter.
  - La clase BufferedReader.
- Gestión de archivos (moderno)
  - La clase Paths.
  - La clase Files.

¿Qué es un sistema de archivos?

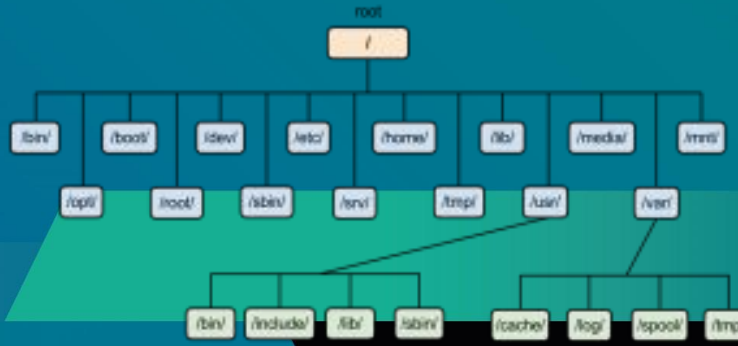
Un **sistema de archivos (*file system*)** es un elemento del sistema operativo que organiza y controla cómo se almacenan y recuperan los datos guardados en un medio de almacenamiento.

- Proveen métodos para **crear**, **mover**, **renombrar** y **eliminar** tanto **archivos** como **directorios**.
- También asignan propiedades como **sólo lectura** y **permisos de acceso**.
- La mayoría de los sistemas operativos manejan su propio sistema de archivos.

En Java, estas operaciones son facilitadas principalmente por las API de **java.io** y **java.nio.file**, que permiten interactuar con archivos de manera robusta y flexible.

Funciones del sistema de archivos

- Asignación de espacio a los archivos.
- Administración del espacio libre y del acceso a los datos resguardados.
- Estructurar la información guardada en un dispositivo de almacenamiento de datos.
- Permite representar la estructura de forma textual o gráfica (gestor de archivos).



¿Qué es un path?

Una **ruta (*path*)** es la forma de referenciar un **archivo** informático o **directorio** en un sistema de archivos de un sistema operativo determinado.

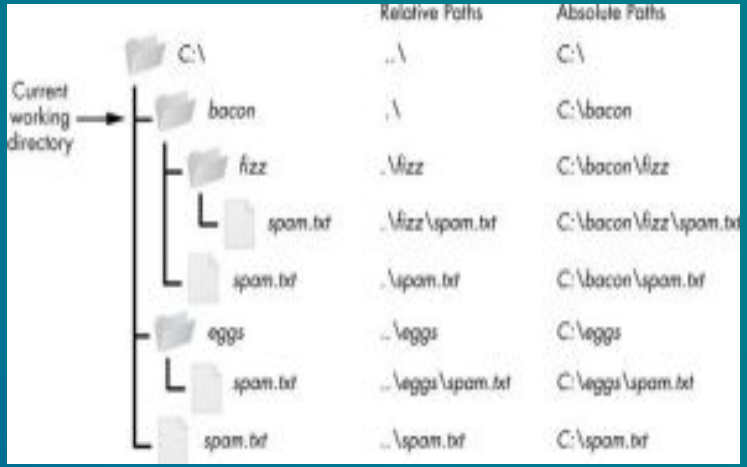
Señala la localización exacta de un archivo o directorio mediante una cadena de caracteres concreta.

Esta puede ser de diversas formas dependiendo del sistema operativo y del sistema de archivos en cuestión.

Rutas relativas vs. rutas absolutas

Rutas absolutas
*Señalan la ubicación de un **archivo** o **directorio** desde el **directorio raíz** del sistema de archivos.*

C:\bacon\fizz\spam.text

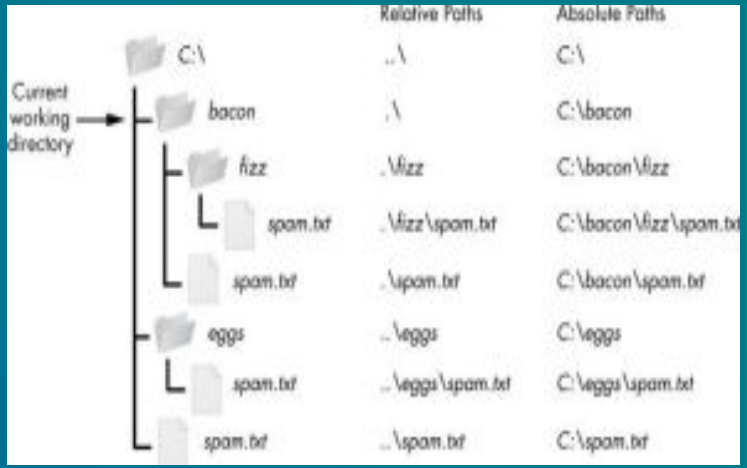


Rutas relativas vs. rutas absolutas

Rutas relativas
*Señalan la ubicación de un **archivo** o **directorio** en relación a la **posición actual**.*

. → representa la ***posición actual*** en la que estamos ubicados en el sistema de archivos.
.\spam.txt ***C:\bacon\spam.txt***

.. → nos mueve al directorio *padre* de la ubicación actual.
..\spam.txt ***C:\spam.txt***



Gestión de archivos (tradicional)

La forma clásica de trabajar con archivos y directorios en Java se encuentra en ***java.io***.

- **File**: Representa una referencia abstracta a archivos y directorios. Permite realizar operaciones como creación, eliminación, verificación de existencia, establecer permisos, etc.
- **FileReader / FileWriter**: Clases para leer y escribir caracteres en un archivo.
- **BufferedReader / BufferedWriter**: Facilitan la lectura y escritura eficiente de datos de caracteres.
- **FileInputStream / FileOutputStream**: Para leer y escribir bytes en archivos binarios.

La clase File

File	exists	Archivos y directorios
	getAbsolutePath	Archivos y directorios
	mkdir	Directorios
	delete	Archivos y directorios
	isDirectory	Directorios
	isFile	Archivos
	list	Archivos y directorios
	listFiles	Directorios

La clase File

```java
String rutaAbsoluta = "C:/Users/maxin/Documents/NetBeansProjects/2024/App/recursos";

File directorio = new File(rutaAbsoluta);

if( ! directorio.exists()){
    if (directorio.mkdir()) {
        System.out.println(directorio.getAbsolutePath());
        // C:/Users/maxin/Documents/NetBeansProjects/2024/App/recursos

        // Establecer permisos
        directorio.setReadable(true, false);  // Permitir lectura para todos
        directorio.setWritable(true, false);  // Permitir escritura para todos
        directorio.setExecutable(true, false);  // Permitir ejecución para todos

        System.out.println("Lectura: " + directorio.canRead());
        System.out.println("Escritura: " + directorio.canWrite());
        System.out.println("Ejecución: " + directorio.canExecute());
    } else {
        System.out.println("No se pudo crear el directorio.");
    }
}
```

```java
String rutaRelativa = "./recursos";

File directorio = new File(rutaRelativa);
```

La clase FileWriter

FileWriter	FileWriter(String)	Si el archivo no existe, lo crea. Si existe, lo sobreescribe.
	FileWriter(String, boolean)	Indica si se agrega texto.
	write(String)	Escribe la cadena de caracteres.

La clase FileWriter

```java
try (FileWriter fw = new FileWriter(directorio.getAbsolutePath()+ "/archivo.txt")) {
    fw.write("Esta es una línea de texto.\n");
} catch (IOException e) {
    System.out.println("Error al escribir en el archivo: " + e.getMessage());
}
```

```java
try (FileWriter fw = new FileWriter(directorio.getAbsolutePath()+ "/archivo.txt", true)) {
    fw.write("Esta es una línea de texto agregaga.\n");
} catch (IOException e) {
    System.out.println("Error al escribir en el archivo: " + e.getMessage());
}
```

La clase FileReader

FileReader	FileReader(String)	Si el archivo no existe, lanza una excepción.
	read()	Lee un caracter. Retorna el caracter leido o -1 si se llega al fin del archivo.

```
try (FileReader fr = new FileReader(directorio.getAbsolutePath() + "/archivo.txt")) {
    int caracter;
    while ((caracter = fr.read()) != -1) {
        System.out.print((char) caracter);
    }
} catch (IOException e) {
    System.out.println("Error al leer el archivo: " + e.getMessage());
}
```

La clase BufferedWriter

BufferedWriter	BufferedWriter(Writer)	Se comporta de acuerdo cómo se instancia *FileWriter*.
	newLine()	Agrega un salto de línea.
	write(String)	Escribe la cadena de caracteres.

La clase BufferedReader

BufferedReader	BufferedReader(Reader)	Mismo comportamiento que *FileReader*.
	read()	Mismo comportamiento que el método *read* de FileReader.
	readLine()	Lee una línea de caracteres hasta el salto de línea. Lo retorna cómo un string.

BufferedWriter y BufferedReader


```java
try (BufferedWriter escritor = new BufferedWriter(new FileWriter(rutaArchivo))) {
    escritor.write("Este es el primer mensaje.");
    escritor.newLine();
    escritor.write("Este es el segundo mensaje.");
} catch (IOException e) {
    System.out.println("Ocurrió un error al escribir en el archivo: " + e.getMessage());
}
```

```java
try (BufferedReader lector = new BufferedReader(new FileReader(rutaArchivo))) {
    String linea;
    while ((linea = lector.readLine()) != null) {
        System.out.println(linea);
    }
} catch (IOException e) {
    System.out.println("Ocurrió un error al leer el archivo: " + e.getMessage());
}
```

Ejercitación

Gestión de archivos (moderno)

Es la forma más potente y flexible de trabajar con archivos y directorios en Java. Se encuentra en ***java.nio.file***.

- **Paths**: Para obtener referencia a la ubicación de un archivo o directorio.
- **Files:** Ofrece una gran cantidad de métodos estáticos para realizar operaciones en archivos, como leer, escribir, copiar, mover y eliminar.

La clase Paths

Paths	Método Estático	get
	Método de instancia	getFileName

```
String rutaArchivo = "./recursos/archivo_ejemplo.txt";
Path archivo = Paths.get(rutaArchivo);

System.out.println(archivo.getFileName());
// archivo_ejemplo.txt
```

La clase Files

Files	copy	Archivos
	delete	Archivos
	exists	Archivos
	isDirectory / createDirectories	Directorios
	readAllLines	Archivos
	readString	Archivos
	write	Archivos
	writeString	Archivos

La clase Files

```java
try {
    List<String> lineas = Files.readAllLines(archivo);
    lineas.forEach(System.out::println);
} catch (IOException e) {
    System.out.println("Error al leer el archivo: " + e.getMessage());
}
```

```java
Path destino = Paths.get(System.getProperty("user.dir") + "/copia.txt");

try {
    Files.copy(archivo, destino);
} catch (IOException e) {
    System.out.println("Error al copiar archivo: " + e.getMessage());
}
```

La clase Files

```java
List<String> lineas = Arrays.asList("Línea 1", "Línea 2", "Línea 3");

try {
    Files.write(destino, lineas, StandardCharsets.UTF_8, StandardOpenOption.APPEND);
} catch (IOException e) {
    System.out.println("Error al escribir en el archivo: " + e.getMessage());
}
```

```java
if(Files.exists(destino)) {
    try {
        Files.delete(destino);
    } catch (IOException e) {
        System.out.println("Error al borrar archivo: " + e.getMessage());
    }
}
```

Ejercitación