Unidad 15

Flujos (Streams) en Java

Contenidos

- Parte 1: Flujos (Streams) en Java 3
- Parte 2: Utilización de Flujos en Java 9

Parte 1: Flujos (Streams) en Java

Los **flujos (streams)** son abstracciones utilizadas para realizar operaciones de entrada y salida en Java. Proporcionan una manera uniforme y eficiente de leer y escribir datos de diversas fuentes como archivos, redes o incluso memoria. Este capítulo abarca los **tipos de flujos**, su división en **flujos de bytes y caracteres**, y las clases asociadas, incluyendo su jerarquía.

1. Tipos de Flujos

Los flujos en Java se clasifican en dos categorías principales según la dirección del flujo de datos:

1. Flujos de Entrada (`InputStream` / `Reader`):

- Permiten **leer datos** desde una fuente, como archivos o sockets.

2. Flujos de Salida (`OutputStream` / `Writer`):

- Permiten **escribir datos** a un destino, como archivos o la consola.

1.1. Ejemplo de Flujo de Entrada

```
import java.io.FileInputStream;
import java.io.IOException;
public class Main {
    public static void main(String[] args) {
        try (FileInputStream fis = new FileInputStream("archivo.txt")) {
            int dato;
            while ((dato = fis.read()) != -1) {
                System.out.print((char) dato);
            }
        } catch (IOException e) {
            System.out.println("Error al leer el archivo: " + e.getMes-
sage());
        }
    }
}
```

1.2. Ejemplo de Flujo de Salida

```
import java.io.FileOutputStream;
import java.io.IOException;
public class Main {
    public static void main(String[] args) {
        try (FileOutputStream fos = new FileOutputStream("salida.txt")) {
            String texto = "Este es un ejemplo de flujo de salida.";
            fos.write(texto.getBytes());
        } catch (IOException e) {
            System.out.println("Error al escribir en el archivo: " +
e.getMessage());
        }
    }
}
```

2. Flujos de Bytes y Flujos de Caracteres

Los flujos en Java se dividen en **flujos de bytes** y **flujos de caracteres** según el tipo de datos que procesan.

2.1. Flujos de Bytes

1. Definición:

- Manejan datos binarios o **bytes crudos**.
- Son útiles para trabajar con archivos de imágenes, audio o cualquier formato no textual.

2. Clases Comunes:

- Entrada: `InputStream`, `FileInputStream`.
- Salida: `OutputStream`, `FileOutputStream`.

3. Ejemplo de Lectura de Bytes

```
import java.io.FileInputStream;
import java.io.IOException;
public class Main {
    public static void main(String[] args) {
        try (FileInputStream fis = new FileInputStream("imagen.jpg")) {
            int byteLeido;
            while ((byteLeido = fis.read()) != -1) {
                System.out.println(byteLeido); // Procesa cada byte
            }
        } catch (IOException e) {
            System.out.println("Error al leer el archivo: " + e.getMes-
sage());
        }
    }
}
```

2.2. Flujos de Caracteres

1. Definición:

- Manejan datos de texto, procesando caracteres en lugar de bytes.
- Son útiles para trabajar con archivos de texto, lectura y escritura de strings.

2. Clases Comunes:

- Entrada: `Reader`, `FileReader`, `BufferedReader`.
- Salida: `Writer`, `FileWriter`, `BufferedWriter`.

3. Ejemplo de Lectura de Caracteres

```
import java.io.FileReader;
import java.io.IOException;
public class Main {
    public static void main(String[] args) {
        try (FileReader fr = new FileReader("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.getMes-
sage());
        }
    }
```

3. Clases Relativas a Flujos

Las clases relacionadas con flujos en Java están organizadas de manera jerárquica y se agrupan en dos paquetes principales:

1. `java.io`: **Proporciona las clases tradicionales para entrada y salida de datos.**
2. `java.nio`: **Introduce flujos más eficientes para operaciones con buffers.**

4. Jerarquía de Clases

La jerarquía de clases en Java comienza con las clases abstractas `InputStream`, `OutputStream`, `Reader`, y `Writer`.

```
InputStream (Bytes)
|--FileInputStream
|__ ByteArrayInputStream
|__ BufferedInputStream
OutputStream (Bytes)
|-- FileOutputStream
|__ ByteArrayOutputStream
|__ BufferedOutputStream
Reader (Caracteres)
|-- FileReader
|-- StringReader
|__ BufferedReader
Writer (Caracteres)
|-- FileWriter
|-- StringWriter
|__ BufferedWriter
```

5. Operaciones Comunes con Flujos

5.1. Lectura y Escritura Combinadas

```
import java.io.*;
public class Main {
    public static void main(String[] args) {
        try (
            BufferedReader br = new BufferedReader(new FileReader("entrada.
txt"));
            BufferedWriter bw = new BufferedWriter(new FileWriter("salida.
txt"))
        ) {
            String linea;
            while ((linea = br.readLine()) != null) {
                bw.write(linea.toUpperCase());
                bw.newLine();
            }
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

6. Ejercicio Guiado

Problema

1. Crea un programa que lea un archivo llamado `numeros.txt` y calcule la suma de todos los números en él.

2. Escribe el resultado en un archivo llamado `resultado.txt`.

3. Usa flujos de caracteres para leer y escribir los datos.

Código Propuesto

```
import java.io.*;
public class Main {
    public static void main(String[] args) {
        int suma = 0;
        try (
            BufferedReader br = new BufferedReader(new FileReader("numeros.txt"));
            BufferedWriter bw = new BufferedWriter(new FileWriter("resultado.txt"))
        ) {
            String linea;
            while ((linea = br.readLine()) != null) {
                suma += Integer.parseInt(linea); // Convierte cada línea en un número
            }
            bw.write("La suma total es: " + suma);
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

7. Preguntas de Evaluación

1. ¿Qué clase es la base para los flujos de bytes en Java?

a) `InputStream`.
b) `Reader`.
c) `BufferedStream`.
d) `OutputStream`.

2. ¿Qué flujo utilizarías para manejar datos de texto?

a) `FileInputStream`.
b) `BufferedWriter`.
c) `FileReader`.
d) `FileWriter`.

3. ¿Cuál es la diferencia principal entre flujos de bytes y flujos de caracteres?

a) Los flujos de bytes manejan datos binarios, los de caracteres manejan texto.
b) Los flujos de caracteres son más rápidos.
c) No hay diferencia.
d) Los flujos de bytes solo se usan con imágenes.

Parte 2: Utilización de Flujos en Java

Los flujos (streams) son esenciales para la **entrada y salida (I/O)** en Java. Permiten leer datos de diversas fuentes, escribir datos en destinos, procesar información de manera eficiente y gestionar operaciones con archivos, sockets o memoria. Este capítulo se centra en ejemplos prácticos y cómo utilizar flujos de entrada y salida.

1. Flujo de Entrada (`InputStream` y `Reader`)

1.1. Leer Datos desde un Archivo

Ejemplo con `FileInputStream` (Flujo de Bytes)

Lee un archivo byte por byte, ideal para datos binarios.

```
import java.io.FileInputStream;
import java.io.IOException;
public class Main {
    public static void main(String[] args) {
        try (FileInputStream fis = new FileInputStream("archivo.dat")) {
            int byteLeido;
            while ((byteLeido = fis.read()) != -1) {
                System.out.print((char) byteLeido); // Muestra el contenido como caracteres
            }
        } catch (IOException e) {
            System.out.println("Error al leer el archivo: " + e.getMessage());
        }
    }
}
```

Ejemplo con `BufferedReader` (Flujo de Caracteres)

Lee un archivo línea por línea, ideal para texto.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class Main {
    public static void main(String[] args) {
        try (BufferedReader br = new BufferedReader(new FileReader("archivo.
txt"))) {
            String linea;
            while ((linea = br.readLine()) != null) {
                System.out.println(linea);
            }
        } catch (IOException e) {
            System.out.println("Error al leer el archivo: " + e.getMes-
sage());
        }
    }
}
```

1.2. Leer Datos desde el Teclado

Ejemplo con `Scanner`

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Introduce tu nombre: ");
        String nombre = scanner.nextLine();
        System.out.println("Hola, " + nombre);
    }
}
```

2. Flujo de Salida (`OutputStream` y `Writer`)

2.1. Escribir Datos en un Archivo

Ejemplo con `FileOutputStream` (Flujo de Bytes)

Escribe datos byte por byte.

```
import java.io.FileOutputStream;
import java.io.IOException;
public class Main {
    public static void main(String[] args) {
        try (FileOutputStream fos = new FileOutputStream("salida.dat")) {
            String texto = "Ejemplo de escritura en bytes";
            fos.write(texto.getBytes());
        } catch (IOException e) {
            System.out.println("Error al escribir en el archivo: " +
e.getMessage());
        }
    }
}
```

Ejemplo con `BufferedWriter` (Flujo de Caracteres)

Escribe datos línea por línea, ideal para texto.

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
public class Main {
    public static void main(String[] args) {
        try (BufferedWriter bw = new BufferedWriter(new FileWriter("salida.
txt"))) {
            bw.write("Primera línea");
            bw.newLine();
            bw.write("Segunda línea");
        } catch (IOException e) {
            System.out.println("Error al escribir en el archivo: " +
e.getMessage());
        }
    }
}
```

3. Operaciones Combinadas

Leer de un Archivo y Escribir en Otro

Ejemplo Completo

```
  import java.io.*;
  public class Main {
      public static void main(String[] args) {
            try (
                  BufferedReader br = new BufferedReader(new FileReader("entrada.
txt"));
                  BufferedWriter bw = new BufferedWriter(new FileWriter("salida.
txt"))
            ) {
                  String linea;
                  while ((linea = br.readLine()) != null) {
                        bw.write(linea.toUpperCase()); // Escribe las líneas en
mayúsculas
                        bw.newLine();
                  }
            } catch (IOException e) {
                  System.out.println("Error: " + e.getMessage());
            }
      }
  }
```

4. Manejo de Excepciones en Flujos

Uso de `try-with-resources`

Java permite manejar recursos automáticamente con `try-with-resources`. Esto asegura que los recursos (como archivos) se cierren adecuadamente.

```
try (BufferedReader br = new BufferedReader(new FileReader("archivo.txt"))) {
    String linea;
    while ((linea = br.readLine()) != null) {
        System.out.println(linea);
    }
} catch (IOException e) {
    System.out.println("Error: " + e.getMessage());
}
```

5. Ejercicio Guiado

Problema

1. Crea un programa que lea un archivo llamado `numeros.txt` con números enteros (uno por línea).
2. Suma todos los números y escribe el resultado en un archivo llamado `resultado.txt`.

Código Propuesto

```
import java.io.*;
public class Main {
    public static void main(String[] args) {
        int suma = 0;
        try (
            BufferedReader br = new BufferedReader(new FileReader("numeros.
txt"));
            BufferedWriter bw = new BufferedWriter(new FileWriter("resulta-
do.txt"))
        ) {
            String linea;
            while ((linea = br.readLine()) != null) {
                suma += Integer.parseInt(linea); // Convierte cada línea
en un número
            }
            bw.write("La suma total es: " + suma);
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Entrada (`numeros.txt`):

```
10
20
30
40
```

Salida (`resultado.txt`):

```
La suma total es: 100
```

6. Preguntas de Evaluación

1. ¿Qué clase se utiliza para leer un archivo línea por línea?

a) `FileInputStream`.
b) `BufferedReader`.
c) `Scanner`.
d) `BufferedWriter`.

2. ¿Qué método se usa para escribir texto en un archivo con `BufferedWriter`?

a) `writeLine`.
b) `newLine`.
c) `write`.
d) `print`.

3. ¿Qué característica proporciona el bloque `try-with-resources`?

a) Lectura más rápida.
b) Manejo automático de excepciones.
c) Cierre automático de recursos.
d) Escritura más eficiente.