Unidad 9

Lectura y escritura de información

Contenidos

- Parte 1: Lectura y escritura de información 3

Parte 1: Lectura y escritura de información

El manejo de archivos y flujos de datos es una parte esencial de muchas aplicaciones en Java. Permite almacenar, leer, procesar y escribir información en diversos formatos. Java proporciona una rica biblioteca de clases en el paquete `java.io` para manejar operaciones de entrada y salida.

1. Fundamentos de Lectura y Escritura en Java

1. Entrada:

- Implica leer datos de diversas fuentes, como teclado, archivos o redes.
- Se realiza utilizando **InputStreams** o clases como **Scanner**.

2. Salida:

- Implica escribir datos en un destino, como la consola, archivos o redes.
- Se realiza utilizando **OutputStreams** o clases como **PrintWriter**.

2. Lectura de Información

2.1. Lectura desde la Consola

La clase **Scanner** es ideal para leer datos desde la entrada estándar (teclado).

Ejemplo

```
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);
    }
}
```

Salida:

```
Introduce tu nombre: Juan
Hola, Juan
```

2.2. Lectura desde un Archivo

Usando `FileReader` y `BufferedReader`

`BufferedReader` permite leer archivos línea por línea, lo que lo hace eficiente para procesar 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());
           }
      }
  }
```

2.3. Lectura desde un Archivo con Scanner

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(new File("archivo.txt"))) {
            while (scanner.hasNextLine()) {
                System.out.println(scanner.nextLine());
            }
        } catch (FileNotFoundException e) {
            System.out.println("Archivo no encontrado: " + e.getMes-
sage());
        }
    }
```

3. Escritura de Información

3.1. Escritura en la Consola

Usa **System.out.print** o **System.out.println** para mostrar información en la consola.

```
System.out.println("Este es un mensaje en la consola.");
```

3.2. Escritura en un Archivo

Usando `FileWriter` y `BufferedWriter`

`BufferedWriter` permite escribir grandes cantidades de texto de manera eficiente.

```
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("archivo.
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());
        }
    }
}
```

3.3. Escritura con `PrintWriter`

`PrintWriter` facilita la escritura de texto, especialmente para dar formato.

```
import java.io.PrintWriter;
public class Main {
    public static void main(String[] args) {
        try (PrintWriter pw = new PrintWriter("archivo.txt")) {
            pw.println("Línea 1");
            pw.println("Línea 2");
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

4. Combinando Lectura y Escritura

Ejemplo Completo

Lee un archivo línea por línea y escribe cada línea en un nuevo archivo, añadiendo un prefijo.

```
  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;
                  int contador = 1;
                  while ((linea = br.readLine()) != null) {
                        bw.write("Línea " + contador + ": " + linea);
                        bw.newLine();
                        contador++;
                  }
            } catch (IOException e) {
                  System.out.println("Error: " + e.getMessage());
            }
      }
  }
```

5. Manejo de Excepciones en I/O

El manejo de excepciones es esencial en operaciones de entrada y salida para garantizar que los recursos se liberen adecuadamente.

1. **Uso de** `try-catch`:

   Atrapa excepciones como **FileNotFoundException** o **IOException**.

2. **Bloque** `try-with-resources`:

   Se asegura de cerrar los recursos automáticamente.

6. Ejercicio Guiado

Problema

1. Crea un programa que:

- Lea un archivo llamado `datos.txt` línea por línea.
- Cuente cuántas líneas contiene el archivo.
- Escriba el resultado en un archivo llamado `resumen.txt`.

Código Propuesto

```
import java.io.*;
public class Main {
    public static void main(String[] args) {
        int contadorLineas = 0;
        try (
            BufferedReader br = new BufferedReader(new FileReader("datos.
txt"));
            BufferedWriter bw = new BufferedWriter(new FileWriter("resumen.
txt"))
        ) {
            while (br.readLine() != null) {
                contadorLineas++;
            }
            bw.write("El archivo contiene " + contadorLineas + "
líneas.");
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

7. Preguntas de Evaluación

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

a) `BufferedReader`.
b) `Scanner`.
c) Ambas.
d) Ninguna de las anteriores.

2. ¿Qué método de `BufferedReader` se usa para leer una línea completa?

a) `read`.
b) `readLine`.
c) `nextLine`.
d) `readFile`.

3. ¿Qué bloque asegura el cierre automático de recursos en Java?

a) `try-catch`.
b) `try-finally`.
c) `try-with-resources`.
d) Ninguna de las anteriores.