ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

ECYS | DIDT
DESARROLLO DE TRANSFERENCIA TECNOLÓGICA

Día, Fecha:	Jueves, 07/09/2023
Hora de inicio:	07:10

Introducción a la programación y computación 1 [F]

José Eduardo Morales García

Archivos

- Los archivos son esenciales para almacenar y manipular datos en programas.
- El manejo de archivos es una de las tareas más comunes en la programación. En Java, existen varias clases y métodos para leer, escribir y manipular archivos de diferentes tipos, como texto, imágenes y datos binarios. En esta presentación, exploraremos las diferentes técnicas y herramientas que podemos utilizar para manejar archivos en Java.

Tipos de archivos

Texto plano

- Los archivos de texto son una forma común de almacenar y compartir datos en formato legible por humanos.
- Por ejemplo, un archivo csv, txt o un archivo de configuración.

Imagen

- Las imágenes son fundamentales en muchas aplicaciones para representar información visual.
- Por ejemplo, un archivo png, jpg.

Binarios

- Los archivos binarios son ideales para almacenar datos complejos y eficientes en programas.
- Por ejemplo, un archivo ejecutable o un archivo de sonido.

Clases File

- **Características**
  - En Java es esencial para manipular archivos y directorios en el sistema de archivos.
  - Proporciona métodos para acceder a información y realizar operaciones en archivos y carpetas.
- **Verificación de Existencia y Propiedades**
  - exists(): Comprueba si el archivo o directorio existe.
  - isFile(): Verifica si es un archivo.
  - isDirectory(): Verifica si es un directorio.
  - length(): Obtiene el tamaño del archivo en bytes.

Clases File

- **Operaciones de Directorios**
  - mkdir(): Crea el directorio asociado a la ruta si no existe.
  - mkdirs(): Crea el directorio y sus directorios padres si no existen.
  - list(): Lista los archivos y subdirectorios en el directorio.
  - renameTo(File dest): Renombra el archivo o directorio a otro nombre o ubicación.
- **Operaciones de Archivos**
  - createNewFile(): Crea un archivo nuevo si no existe.
  - delete(): Elimina el archivo o directorio.
  - canRead(), canWrite(), canExecute(): Verifica permisos.

Ejemplo

```java
public static void main(String[] args) {

    File file = new File( pathname: "archivo.txt");
    if (file.exists()) {
        if (file.isFile()) {
            System.out.println( x: "Es un archivo.");
        } else if (file.isDirectory()) {
            System.out.println( x: "Es un directorio.");
        }
    } else {
        System.out.println( x: "El archivo no existe.");
    }

}
```

Ejemplo

```
public static void main(String[] args) {

    File file = new File( pathname: "nuevo_archivo.txt");
    try {
        if (file.createNewFile()) {
            System.out.println( x: "Archivo creado con éxito.");
        } else {
            System.out.println( x: "El archivo ya existe.");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Ejemplo

```java
public static void main(String[] args) {

    File directory = new File( pathname: "nuevo_directorio");
    if (directory.mkdir()) {
        System.out.println( x: "Directorio creado con éxito.");
    } else {
        System.out.println( x: "No se pudo crear el directorio.");
    }
```

Clases para manipulación de archivos de texto plano

Clases FileReader

- **Características**
  - La lectura eficiente de archivos de texto es esencial para procesar datos legibles por humanos en aplicaciones.
  - Java proporciona la clase FileReader para una lectura eficiente de datos de texto.
  - Aspectos importantes: Lectura de caracteres, uso de búferes internos.

- **Ventajas del Uso de FileReader**
  - *Eficiencia*: Lee datos en bloques en lugar de carácter por carácter, mejorando la velocidad.
  - *Lectura de Caracteres*: Ideal para archivos de texto legibles por humanos.

Clases FileReader

- **Consideraciones**
  - FileReader es útil para lecturas de texto eficientes.
  - Considera usar BufferedReader para una mayor eficiencia y facilidad de lectura de líneas.

Clases BufferedReader

- **Características**
  - La lectura eficiente de archivos de texto es crucial para el rendimiento de las aplicaciones.
  - Java ofrece la clase BufferedReader para mejorar la eficiencia en la lectura de texto.

Clases BufferedReader

- **Ventajas del Uso**
  - *Eficiencia*: Reduce la frecuencia de lecturas desde el flujo subyacente.
  - *Lectura de Líneas*: Facilita la lectura de líneas completas en lugar de caracteres individuales.

- **Consideraciones**
  - BufferedReader mejora la eficiencia en lecturas repetitivas desde un flujo de caracteres.
  - Útil para leer grandes archivos de texto o realizar muchas lecturas en un bucle.
  - BufferedReader envuelve un FileReader para mejorar la eficiencia al leer en bloques.

Clases BufferedReader

- **Lectura Línea a Línea**
  - *readLine():* Lee una línea completa del flujo de entrada.
  - Útil para procesar archivos de texto con líneas separadas.

- **Lectura Carácter a Carácter**
  - Aunque BufferedReader es más eficiente, aún puedes leer caracteres individualmente con read().

Clases FileWriter

- **Definición**
  - La escritura eficiente en archivos de texto es crucial para almacenar y compartir información legible por humanos.
  - Java proporciona la clase FileWriter para una escritura eficiente en archivos de texto.
- **Clase Clave**
  - Escribe caracteres en un archivo de texto de manera eficiente.
  - Definición: java.io.FileWriter
  - Aspectos importantes: Escritura de caracteres, uso de búferes internos.

Clases FileWriter

- **Ventajas del Uso**
  - *Eficiencia*: Escribe datos en bloques en lugar de carácter por carácter, mejorando la velocidad.
  - *Escritura de Caracteres*: Ideal para crear o modificar archivos de texto legibles por humanos.

- **Uso de Búferes para Escritura Eficiente**
  - Al igual que con otras clases de escritura, FileWriter también se puede mejorar usando búferes.
  - BufferedWriter: Envuelve un FileWriter para mejorar la eficiencia al escribir en bloques.

Clases BufferedWriter

- **Características**
  - La escritura eficiente en archivos de texto es crucial para almacenar y compartir información legible por humanos.
  - Java proporciona la clase BufferedWriter para una escritura eficiente en archivos de texto.
  - Envuelve un flujo de escritura de caracteres, mejorando la eficiencia al escribir en bloques.

Clases BufferedWriter

- **Ventajas del Uso de BufferedWriter**
  - Eficiencia: Escribe datos en bloques en lugar de carácter por carácter, mejorando la velocidad.
  - Escritura de Líneas: Facilita la escritura de líneas completas en lugar de caracteres individuales.

Beneficios y Limitaciones

Ventajas

- Legibles por humanos, adecuados para configuraciones, versionamiento y más.

Limitaciones

- Ineficiente para grandes volúmenes de datos, no aptos para todos los tipos de datos.

Ejemplo

```java
public static void main(String[] args) {
    try {
        FileReader reader = new FileReader( fileName: "archivo.txt");
        int charRead;
        while ((charRead = reader.read()) != -1) {
            // Procesar carácter
        }
        reader.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Ejemplo

```java
public static void main(String[] args) {
    try {
        FileReader fileReader = new FileReader( fileName: "archivo.txt");
        BufferedReader bufferedReader = new BufferedReader( in: fileReader);
        String line;
        while ((line = bufferedReader.readLine()) != null) {
            // Procesar línea
        }
        bufferedReader.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Ejemplo

```java
public static void main(String[] args) {
    try {
        FileWriter writer = new FileWriter( fileName: "archivo.txt");
        writer.write( str: "Hola, mundo!");
        writer.close();
        System.out.println( x: "Datos escritos en el archivo.");
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Ejemplo

```java
public static void main(String[] args) {
    try {
        FileWriter writer = new FileWriter( fileName: "archivo.txt");
        char[] data = {'H', 'o', 'l', 'a'};
        writer.write( cbuf: data);
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Ejemplo

```java
public static void main(String[] args) {
    try {
        FileWriter fileWriter = new FileWriter( fileName: "archivo.txt");
        BufferedWriter bufferedWriter = new BufferedWriter( out: fileWriter);
        String line = "Hola, mundo!";
        bufferedWriter.write( str: line);
        bufferedWriter.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Ejemplo

```java
public static void main(String[] args) {
    try {
        FileReader reader = new FileReader( fileName: "archivo.txt");
        BufferedReader bufferedReader = new BufferedReader( in: reader);
        String line = bufferedReader.readLine();
        bufferedReader.close();

        line = line.replace( target: "Hola",  replacement: "¡Hola");

        FileWriter writer = new FileWriter( fileName: "archivo.txt");
        writer.write( str: line);
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Clases para la manipulación de archivos binarios

Clases FileInputStream

- **Características**
  - La lectura eficiente de archivos binarios es esencial para el rendimiento de aplicaciones que manejan datos complejos.
  - Java proporciona la clase FileInputStream para una lectura eficiente de datos binarios.
  - Lee secuencias de bytes desde un archivo de manera eficiente.
- **Ventajas del Uso**
  - Eficiencia: Lee datos en bloques en lugar de byte por byte, mejorando la velocidad.
  - Manejo de Bytes: Útil para leer datos binarios como imágenes, audio y video.
  - Automatización: Automáticamente maneja el proceso de lectura y controla la lectura de bloques.

Clases FileInputStream

- **Uso de Búferes para Lectura Eficiente**
  - Aunque FileInputStream mejora la eficiencia, aún se puede mejorar más usando búferes.
  - BufferedInputStream: Una clase que envuelve un FileInputStream y mejora la eficiencia al leer en bloques.

Clases FileOutputStream

- **Características**
  - La escritura eficiente en archivos binarios es fundamental para almacenar datos complejos en aplicaciones.
  - Java ofrece la clase FileOutputStream para una escritura eficiente en archivos binarios.
  - Escribe secuencias de bytes en un archivo de manera eficiente.
- **Ventajas del Uso de FileOutputStream**
  - Eficiencia: Escribe datos en bloques en lugar de byte por byte, mejorando la velocidad.
  - Manipulación de Bytes: Útil para escribir datos binarios como imágenes, audio y video.
  - Automatización: Automáticamente maneja el proceso de escritura y controla la escritura de bloques.

Clases FileOutputStream

- **Uso de Búferes para Escritura Eficiente**
  - Al igual que con FileInputStream, FileOutputStream también se puede mejorar usando búferes.
  - BufferedOutputStream: Envuelve un FileOutputStream para mejorar la eficiencia al escribir en bloques.

Beneficios y Limitaciones

Ventajas

- Eficientes para grandes volúmenes de datos, adecuados

Limitaciones

- No legibles por humanos, menos adecuados para configuraciones y versionamiento.

Ejemplo

```java
public static void main(String[] args) {
    try {
        FileInputStream fis = new FileInputStream( name: "archivo.bin");
        int byteRead;
        while ((byteRead = fis.read()) != -1) {
            // Procesar byte
        }
        fis.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Ejemplo

```java
public static void main(String[] args) {
    try {
        FileInputStream fis = new FileInputStream( name: "archivo.bin");
        BufferedInputStream bis = new BufferedInputStream( in: fis);
        int byteRead;
        while ((byteRead = bis.read()) != -1) {
            // Procesar byte
        }
        bis.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Ejemplo

```java
public static void main(String[] args) {
    try {
        FileOutputStream fos = new FileOutputStream( name: "archivo.bin");
        byte[] data = {65, 66, 67, 68}; // Datos a escribir
        fos.write( b: data);
        fos.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Ejemplo

```java
public static void main(String[] args) {
    try {
        FileOutputStream fos = new FileOutputStream( name: "archivo.bin");
        BufferedOutputStream bos = new BufferedOutputStream( out: fos);
        byte[] data = {65, 66, 67, 68}; // Datos a escribir
        bos.write( b: data);
        bos.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Clases para la manipulación de imágenes

Clases para imágenes

- **Características**
  - Las imágenes son fundamentales en muchas aplicaciones para representar información visual.
  - Java proporciona la clase ImageIO para cargar y guardar imágenes en diferentes formatos.

- **Clase Clave**
  - ImageIO: Proporciona métodos para leer y escribir imágenes en diferentes formatos.
  - Definición: javax.imageio.ImageIO
  - Aspectos importantes: Carga y guarda imágenes, manejo de formatos como JPEG, PNG, GIF, etc.

Beneficios y Limitaciones

Ventajas

- Carga y guardado de imágenes en varios formatos, manejo automático de formatos.

Limitaciones

- No es adecuado para manipulaciones avanzadas de imágenes.

Carga de Imágenes

READ(FILE INPUT):

- Carga una imagen desde un archivo.
- maneja automáticamente la detección del formato de imagen.
- puede devolver una instancia de bufferedimage.

```
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

try {
    File file = new File("imagen.jpg");
    BufferedImage image = ImageIO.read(file);
    System.out.println("Imagen cargada con éxito."
} catch (IOException e) {
    e.printStackTrace();
}
```

Guardado de Imágenes

- Read(file input): Write(renderedimage im, string formatname, file output): guarda una imagen en un archivo con un formato específico.
- El parámetro formatname especifica el formato de imagen a utilizar.

```java
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

try {
    BufferedImage image = new BufferedImage(800, 600, BufferedImage.TYPE_INT_RGB);
    File output = new File("nueva_imagen.png");
    ImageIO.write(image, "png", output);
    System.out.println("Imagen guardada con éxito.");
} catch (IOException e) {
    e.printStackTrace();
}
```

Dudas y Preguntas

Parte práctica