La Sénia
Institut d'Educació Secundària
Paiporta

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Gestión de Ficheros

Curso 2024 – 2025

Gestión de Ficheros:

- Sistemas de ficheros
- Ficheros de datos. Registros
- Manipular ficheros
- Persistencia. Serialización

2. Sistema de Ficheros

clase **File**

- proporciona información sobre archivos y directorios
- Crear archivos:

```
try{
    File file = new File("fichero.txt");
    if (file.createNewFile()) {
        System.out.println("Fichero creado: " + file.getName());
    } else {
        System.out.println("El fichero ya existe");
    }
}catch (IOException e) {
        System.out.println("Error: No puedo crear el fichero");
}
```

clase **File**

- Crear directorio:

```
try{
    File directorio = new File("directorio");
    if (directorio.mkdir()) {
        System.out.println("Fichero creado: " + file.getName());
    } else {
        System.out.println("El fichero ya existe");
    }
}catch (IOException e) {
        System.out.println("Error: No puedo crear el fichero");
}
```

2. Sistema de Ficheros

Otros **métodos** de la clase **File**:

```
delete()        Borra el fichero indicado
renameTo()      Renombra un fichero con el nombre pasado como parámetro
exists()        Boolean que nos indicará si el fichero existe
isFile()        Boolean que indica si el fichero es un archivo
isDirectory()   Boolean que indica si el fichero es un directorio
listFiles()     Devuelve un array con los ficheros del directorio
getName()       Devuelve un String con el nombre del fichero
getPath()       Devuelve un String con la ruta relativa
getAbsolutePath()  Devuelve un String con la ruta absoluta
getParent()     Devuelve un String con el directorio padre
canWrite() Boolean que nos indicará si el fichero puede ser escrito
canRead()  Boolean que nos indicará si el fichero puede ser leído
```

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/File.html

2. Sistema de Ficheros. Ejemplo

Crear un método que permita **listar** el contenido de un directorio
Debe comprobar que la ruta es de un directorio y lista su contenido:

```
public static void listarDirectorio(){
    String ruta;
    Scanner sc = new Scanner(System.in);
    System.out.println("indica la ruta del directorio a listar");
    ruta=sc.nextLine();
    if(ruta.length > 0) {
        File f = new File(ruta); //Creamos o cargamos el directorio en File
        if(f.isDirectory()) {     //Comprobamos que es un directorio y no un archivo
          File[] ficheros = f.listFiles();
          System.out.println("Listado de los ficheros");
          for(File file : ficheros)   //listamos el directorio
            System.out.println("\t" + file.getName());
        }
    }
}
```

2. Sistema de Ficheros. Ejercicios

1. Crear un programa en java (llamado `rmj`) que permita borrar un fichero pasándole el nombre del fichero como argumento. El script tendrá la forma:

```
java rmj fichero.txt
```

El programa debe comprobar errores como que el fichero no exista o que sea un directorio.

2. Crear un programa que a partir de un nombre pasado como argumento diga si es un fichero o un directorio y muestre la ruta absoluta, el tamaño y sus permisos.

3. Ficheros de Datos

Los ficheros pueden ser almacenados como:

- Un flujo de bytes (ficheros binarios)
- Un flujo de caracteres (ficheros de texto)

La información es tratada como:

- Un **flujo de entrada** (lectura)
- Un **flujo de salida** (escritura)

Flujo de datos	**Flujo de entrada o lector**	**Flujo de salida o escritor**
Bytes	InputStream	OutputStream
Carácteres	Reader	Writer

3. Flujos E/S en Java (Recordatorio)

Gestionar flujos en java: **java.io**

- Flujo de bytes: clases **abstractas** **InputStream** y **OutputStream**.
- Flujo de caracteres: clases **abstractas** **Reader** y **Writer**.
- Flujos estándar:
  - Flujo de entrada (teclado): **System.in**
  - Flujo de Salida (pantalla): **System.out**

3. Ficheros de Datos

Clases para manipulación de ficheros:

Bytes: FileInputStream y **FileOutputStream**

Caracteres: FileReader y **FileWriter**

Entrada y salida estándar de Java (System) usa flujos de datos: InputStream (entrada) y OutputStream (salida).

4. Manipular Ficheros. Operaciones básicas

Operaciones básicas con ficheros:

- **Abrir** el fichero: creamos un objeto (`new`) que va a cargar los datos del fichero.
- **Leer** datos: se leen con `read()` o `readLine()`.
- Avance de línea: `newLine()`
- **Guardar** datos: se escriben con `write()` o `print()`
- **Cerrar** el fichero: se usa `close()` para cerrar el fichero

Importante: Siempre hay que **cerrar** el fichero. No se puede ver el contenido escrito en él si no se cierra

clase **FileInputStream**

- Permite leer información de un fichero binario (bytes):

```
try {
    File f = new File("lectura.bin");
    FileInputStream  fis = new FileInputStream(f);
                        //new FileInputStream("lectura.bin");
    int content = fis.read();
    while(content != -1) {
        System.out.println(content);
        content = fis.read();
    }
    fis.close();
} catch(IOException e) {
    e.printStackTrace();
}
```

clases **FileReader** y **BufferedReader**

- Permite leer información de un fichero:
  - FileReader: lee carácter a carácter
  - BufferedReader: guarda caracteres de un buffer

```
try {
    File f = new File("lectura.txt");
    FileReader fr = new FileReader(f); //new FileReader("lectura.txt")
    BufferedReader br = new BufferedReader(fr);
    String linea = br.readLine(); //fr.read(), para un caracter
    System.out.println();
    while(linea != null) {
        System.out.println(linea);
        linea = br.readLine();
    }
    br.close();
    fr.close();
} catch(IOException e) {
    e.printStackTrace();
}
```

clase **FileOutputStream**

- Permite leer información de un fichero binario:

```
try {
    File f = new File("escritura.bin");
    FileOuputStream  fos = new FileOuputStream(f);
    fos.write(0x41);
    fos.write(0x42);
    fos.write(0x43);
    fos.write('C');
    fos.close();
} catch(IOException e) {
    e.printStackTrace();
}
```

4. Manipular Ficheros

clases **FileWriter** y **PrintWriter**

- Permite escribir información a un fichero.
- PrintWriter permite formatear la escritura.

```
try {
    FileWriter fw = new FileWriter("file1.txt");
    fw.write("Hola, esto es un ejemplo de FileWriter\nOK");
    fw.close();  //Mete el contenido en el fichero
} catch(IOException e) {
    e.printStackTrace();
}
try {
    PrintWriter pw = new PrintWriter("file1.txt");
    pw.println("Hola, esto es un ejemplo de PrintWriter");
    pw.printf("Este es un número: %d", 42);
    pw.close();
} catch(IOException e) {
    e.printStackTrace();
}
```

4. Manipular Ficheros. Ejemplo

Copiamos el contenido de un fichero fuente.txt en otro destino.txt carácter a carácter:

```
public static void copiaFichero() {
    int caracter;
    try {
        FileReader in = new FileReader("fuente.txt");
        // new FileWriter("archivo",true) para abrir en modo append
        FileWriter out = new FileWriter("destino.txt");
        while( (caracter = in.read()) != -1) {
            out.write(caracter);
        }
        in.close();
        out.close();
    } catch(FileNotFoundException e1) {
        System.err.println("Error: No se encuentra el fichero");
    } catch(IOException e2) {
        System.err.println("Error leyendo/escribiendo fichero");
    }
}
```

4. Manipular Ficheros. Ejemplo

Copia el contenido de un fichero fuente.txt en otro destino.txt línea a línea: **BufferedReader**:

```
public static void copiaFicheroLinea() {
    String linea;
    try {
        FileReader in = new FileReader("fuente.txt");
        //Usamos BufferedReader para poder leer lineas y no caracteres
        BufferedReader entrada = new BufferedReader(in);
        FileWriter out = new FileWriter("destino.txt");
        linea=entrada.readLine();
        while (linea != null) {
            out.write(linea);
            linea=entrada.readLine();
            out.flush(); //Guarda los datos en el fichero
        }
        in.close();
        out.close();
    } catch(FileNotFoundException e1) {
        System.err.println("Error: No se encuentra el fichero");
    } catch(IOException e2) {
        System.err.println("Error leyendo/escribiendo fichero");
    }
}
```

4. Manipular Ficheros. Ejercicios

1. Crear un programa que permita introducir texto por teclado y lo irá guardando en un fichero.

2. Crear un programa que pida un nombre de usuario y contraseña y lo guarde en un fichero con el formato:

```
<nombre>:<Contraseña>
```

Utiliza ese fichero para validar contraseñas. Crea un sistema que pida el nombre de un usuario y su contraseña y valide si el nombre existe y, en caso afirmativo, si la contraseña es correcta.

5. Persistencia y Serialización

Serialización:

Transformar un objeto en una secuencia de bytes que represente el estado del objeto.

Una vez serializado un objeto, se puede enviar a un fichero.

Persistencia:

Un objeto queda serializado y almacenado en un fichero, y nos permite recomponer el objeto.

El estado de un objeto representa el estado de sus campos. Si un objeto es un campo de otro objeto, debe ser serializado para poder serializar el primer objeto.

5. Persistencia y Serialización

Para poder serializar un objeto de una clase es necesario que implemente la interfaz **java.io.Serializable**.

Esta interfaz no define ningún método, el objetivo es marcar las clases que convertiremos en secuencias de bytes.

Cuando marcamos una clase es **Serializable**, Java se encargará de realizar la serialización de forma automática.

Dos clases utilizadas para serializar objetos: **ObjectOutputStream** (flujo desde nuestro programa hacia fuera)

ObjectInputStream (flujo desde fuera hacia nuestro programa)

Los métodos definidos son **writeObject**() y **readObject**().

5. Persistencia y Serialización

Imaginemos una clase Amigo serializable que guarda el nombre y el teléfono:

```
public class Amigo implements Serializable {

    private String nombre;
    private String telefono;

    public Amigo(String nombre, String telefono) {
        this.nombre = nombre;
        this.telefono = telefono;
    }

    public String toString() {
        return nombre + " -> " + telefono);
    }
}
```

5. Persistencia y Serialización. Ejemplo

Creamos 2 objetos de la clase “Amigo”, los escribimos en un fichero (serializar)

```
public class Serializar{
    public static void main(String[] args) {
        File f = new File("amigos.dat");
        try { //Serializamos objetos
            FileOutputStream fs = new FileOutputStream(f);
            ObjectOutputStream oos = new ObjectOutputStream(fs);
            Amigo a = new Amigo("Pedro Perez", "666123456");
            oos.writeObject(a);
            Amigo a1 = new Amigo("Ana Arnau", "699654321");
            oos.writeObject(a1);
            oos.close();
            fs.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

5. Persistencia y Serialización. Ejemplo

Para leer los objetos del fichero (deserializar), usamos los siguientes objetos:

```
//DESERIALIZAMOS OBJETOS
try {
    FileInputStream fis = new FileInputStream("amigos.dat");
    ObjectInputStream ois = new ObjectInputStream(fis);

    System.out.println("Listado de amigos en el fichero:");
    Amigo a3 = (Amigo)ois.readObject();
    System.out.println(a3);
    Amigo a4 = (Amigo)ois.readObject();
    System.out.println(a4);
    ois.close();
    fis.close();
} catch (IOException e) {
    System.out.println("Excepción: "+e.getMessage());
} catch (ClassNotFoundException e) {
    System.out.println("Excepción: "+e.getMessage());
}
```