

Programación

UD 8: Excepciones y ficheros

Excepciones y ficheros

1.- Tratamiento de excepciones

1.1.- Captura de excepciones: try / catch / finally

1.2.- Propagación de excepciones: throw y throws

2.-Ficheros

2.1.- Creación y eliminación de ficheros

2.2.- Lectura y escritura de ficheros

2.3.- Serialización de objetos

2.4.- Utilización de los sistemas de ficheros

1.- Tratamiento de excepciones

Una excepción provoca la terminación abrupta del programa. Sin embargo, una excepción puede tratarse para evitar dicha terminación abrupta.

Existen dos maneras de tratar una excepción:

- ✓ Capturar la excepción (bloque try/catch/finally)
- ✓ Propagar la excepción (throw y throws)

1.1 Captura de excepciones: try / catch / finally

1.1 Captura de excepciones: try / catch / finally

```
try {  
    ... // instrucciones que pueden provocar alguna excepción  
  
}  
catch ( NombreDeExcepcion1 nomE1 ) {  
    ... // instrucciones a realizar si se produce  
        // la excepción NombreDeExcepcion1  
  
}  
... // puede haber varios catch por cada try  
catch ( NombreDeExcepcionX nomEX ) {  
    ... // instrucciones a realizar si se produce  
        // la excepción NombreDeExcepcionX  
  
}  
finally {  
    ... // bloque opcional, instrucciones a realizar tanto  
        // si se ha producido una excepción como si no  
}
```

1.1 Captura de excepciones: try / catch / finally

Básicamente, el significado es como sigue:

1. Se intenta (**try**) ejecutar un bloque de código en el que pueden ocurrir errores que se representan con alguna de las excepciones que se explicitan en los bloques catch.
2. Si se produce un error, el sistema lanza una excepción (**throws**) que puede ser capturada (**catch**) en base al tipo de excepción, ejecutándose las instrucciones correspondientes.
3. Finalmente, tanto si se ha producido o no una excepción y si ésta ha sido o no tratada, se ejecutan las instrucciones asociadas a la cláusula **finally**.
4. Al finalizar todo el bloque, la ejecución se reanuda del modo habitual.

Siempre que aparece una cláusula try, debe existir al menos una cláusula catch o finally.

Nótese que, **para una única cláusula try, pueden existir tantos catch como sean necesarios** para tratar las excepciones que se puedan producir en el bloque de código del try.

Cuando en el bloque try se lanza una excepción, **los bloques catch se examinan en orden**, y el primero que se ejecuta es aquel cuyo tipo sea compatible con el de la excepción lanzada.

1.1 Captura de excepciones: try / catch / finally

Así pues, el orden de los bloques catch es importante.

Por ejemplo, el orden en los bloques catch que siguen no sería adecuado:

```
catch (Exception e) {  
    ...  
}  
catch (ExcepcionNumeroNegativo e) {  
    ...  
}
```

Con este orden, el segundo bloque catch nunca se alcanzaría, puesto que todas las excepciones serían capturadas por el primero, ya que la excepción `ExcepcionNumeroNegativo` se deriva de la clase `Exception`.

Afortunadamente, el compilador advierte sobre esto. El orden correcto consiste en invertir los bloques catch para que la excepción más específica aparezca antes que cualquier excepción de una clase antecesora.

1.2 Propagación de excepciones: throw y throws

1.2 Propagación de excepciones

Si la excepción se produce dentro de un método, puede elegirse dónde se trata.

Si se trata dentro del método se utilizan las cláusulas ya vistas try/catch/finally pero si no, puede decidirse su propagación hacia el punto del programa desde donde se invocó el método.

Para hacer esto es necesario que el método tenga en su cabecera la cláusula **throws** y el nombre de la clase de excepción (o clases, separadas por comas) que se propagarán desde dicho método.

De este modo, si se produce una excepción dentro del cuerpo del método se abortará la ejecución del mismo y se lanzará la excepción hacia el punto desde donde se invocó al método.

throws NombreExcepcion: se sitúa en la cabecera del método e indica que, si se produce una excepción de ese tipo, el método la propagará al punto de invocación del mismo;

throw NombreExcepcion: indica que se lanza la excepción que se escribe a continuación.

1.2 Propagación de excepciones

```
import java.util.*;
public class Principal {
    public static void main(String[] Args) {
        int numero_positivo=0;
        try {
            numero_positivo = leer_numero_positivo();
            System.out.println("Hemos leído el numero: "+numero_positivo);
        } catch (NumberFormatException e){
            System.out.println(e.getMessage());
        }
    }

    public static int leer_numero_positivo() throws NumberFormatException {
        int dato_leido=0;
        Scanner s = new Scanner(System.in);
        System.out.println("Introduzca un numero positivo:");
        dato_leido = s.nextInt();
        if (dato_leido < 0) {
            throw new NumberFormatException(dato_leido + " es incorrecto!");
        }
        return dato_leido;
    }
}
```

2.- Ficheros

Distinguimos dos tipos de ficheros: los de **texto** y los **binarios**.

En los **ficheros de texto** la información se guarda como caracteres. Esos caracteres están codificados en **Unicode**, o en **ASCII** u otras codificaciones de texto.

Los **ficheros binarios** almacenan la información en bytes, codificada en binario, pudiendo ser de cualquier tipo: fotografías, números, letras, archivos ejecutables, etc.

Los archivos binarios guardan una representación de los datos en el fichero. O sea que, cuando se guarda texto no se guarda el texto en sí, sino que se guarda su representación en **código UTF-8**.



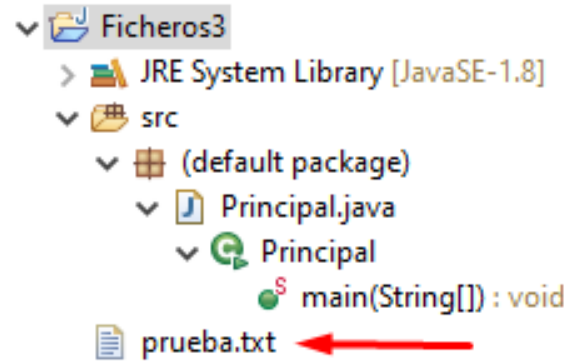
2.1- Creación y eliminación de ficheros

2.1- Creación y eliminación de ficheros

Creacion de ficheros

```
public static void main(String[] args){  
  
    //Creación del objeto File, pero aun no existe en el sistema de archivos.  
    File Archivo = new File("prueba.txt");  
    try {  
        //Se crea fisicamente en el sistema de archivos  
        Archivo.createNewFile();  
    }catch (IOException e) {  
        System.out.println(e.getMessage());  
    }  
  
}
```

Si no se especifica ninguna ruta, el archivo se creará en la carpeta del proyecto:



2.1- Creación y eliminación de ficheros

El método `createNewFile`

El método que utilizamos para la creación de ficheros devuelve `TRUE` si el fichero se ha creado y `FALSE` si no se creado **porque ya existía**, por lo que podríamos completar el código anterior del siguiente modo:

```
public static void main(String[] args){  
  
    //Creación del objeto File, pero aun no existe en el sistema de archivos.  
    File Archivo = new File("prueba.txt");  
    try {  
        if (Archivo.createNewFile()) {  
            System.out.println("El archivo ha sido creado");  
        }else {  
            System.out.println("El archivo ya existia");  
        }  
    } catch (IOException e) {  
        System.out.println(e.getMessage());  
    }  
}
```

1ª ejecución: `<terminated> Principal (25) [Java Application] C:\Program Files\JAVA\`
El archivo ha sido creado

2ª ejecución: `<terminated> Principal (25) [Java Application] C:\Program Files\JAVA\`
El archivo ya existia

2.1- Creación y eliminación de ficheros

El método delete

El método delete devuelve TRUE si el fichero se pudo borrar con éxito y FALSE si no se pudo borrar **porque alguien ya lo ha borrado o se ha cambiado su ubicación o permanece abierto**.

```
//Creación del objeto File, pero aun no existe en el sistema de archivos.
File Archivo = new File("prueba.txt");
try {
    if (Archivo.createNewFile()) {
        System.out.println("El archivo ha sido creado");
    }else {
        System.out.println("El archivo ya existia");
    }
} catch (IOException e) {
    System.out.println(e.getMessage());
}

boolean resultado_borrar = Archivo.delete();
if (resultado_borrar) {
    System.out.println("El archivo fue borrado");
}else {
    System.out.println("El archivo no pudo ser borrado");
}
```

2.2.- Lectura y escritura de ficheros.

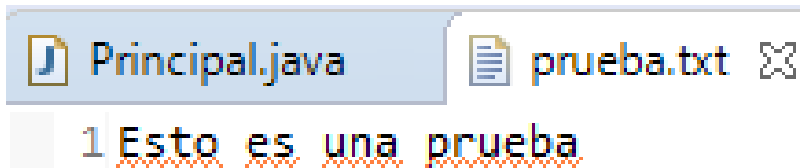
2.2- Lectura y escritura de ficheros

La clase `FileWriter` para escribir en ficheros

```
//Creación del objeto File, pero aun no existe en el sistema de archivos.
File Archivo = new File("prueba.txt");
try {
    Archivo.createNewFile();
} catch (IOException e) {
    System.out.println(e.getMessage());
}

//FileWriter - clase que me permite escribir en un archivo a través de su método write
try {
    FileWriter file = new FileWriter(Archivo);
    file.write("Esto es una prueba\n");
    file.close();
} catch (IOException e) {
    System.out.println(e.getMessage());
}
```

Es muy importante cerrar el fichero con el método **close**, ya que sino, NO veremos los datos escritos en el fichero.




2.2- Lectura y escritura de ficheros

La clase `FileWriter` para escribir varias líneas en un fichero

```
//FileWriter - clase que me permite escribir en un archivo a través de su método write
try {
    FileWriter file = new FileWriter(Archivo);
    for (int i=0; i < 10; i++) {
        file.write("Esto es una prueba" + i + "\n");
    }
    file.close();
} catch (IOException e) {
    System.out.println(e.getMessage());
}
```

En determinadas ocasiones, es interesante conocer el tamaño del archivo después de escribir, para lo que utilizaremos el método **`Archivo.length()`** que devolverá el tamaño del archivo en bytes .



```
1 Esto es una prueba0
2 Esto es una prueba1
3 Esto es una prueba2
4 Esto es una prueba3
5 Esto es una prueba4
6 Esto es una prueba5
7 Esto es una prueba6
8 Esto es una prueba7
9 Esto es una prueba8
10 Esto es una prueba9
```

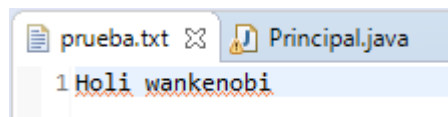
2.2- Lectura y escritura de ficheros

La clase `FileWriter` para añadir líneas a un fichero ya creado

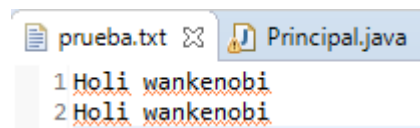
Si le pasamos un segundo parámetro a “true” en la creación de un objeto `FileWriter`, los datos siempre se añadirán al final del fichero, de modo que NO machacamos el contenido de anteriores ejecuciones.

```
try {  
    FileWriter file = new FileWriter(Archivo, true);  
    file.write("Holi wankenobi\n");  
    file.close();  
} catch (IOException e) {  
    System.out.println(e.getMessage());  
}
```

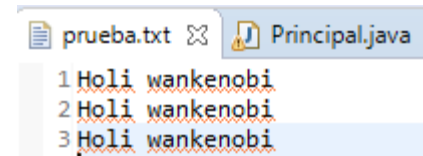
1ª Ejecución



2ª Ejecución



3ª Ejecución



2.2- Lectura y escritura de ficheros

La clase Scanner para leer de ficheros

Utilizaremos el método hasNext() para leer hasta final de fichero

```
try {  
    Scanner sc = new Scanner(Archivo);  
    while (sc.hasNext()) {  
        String linea = sc.nextLine();  
        System.out.println(linea);  
    }  
    sc.close();  
}  
catch (IOException e) {  
    System.out.println(e.getMessage());  
}
```

<terminated> Principal (25) [Java .

Esto es una prueba0
Esto es una prueba1
Esto es una prueba2
Esto es una prueba3
Esto es una prueba4
Esto es una prueba5
Esto es una prueba6
Esto es una prueba7
Esto es una prueba8
Esto es una prueba9

2.3.- Serialización de objetos

2.3- Serialización de objetos

La **serialización** consiste en transformar un objeto en una secuencia o serie de **bytes** de tal manera que represente el **estado** de dicho objeto.

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

La **persistencia** se consigue al tener el objeto seriado y almacenado en un fichero, porque sería posible recomponer el objeto.

El **estado de un objeto** es básicamente el estado de cada uno de los campos. Imaginemos que un campo es a su vez otro objeto, en ese caso debería de ser serializado para serializar el primer objeto.

2.3- Serialización de objetos

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

Dicha interfaz no define ningún método, el objetivo es marcar las clases que vamos a convertir en secuencias de bytes.

Ejemplo:

```
public class Amigo implements Serializable {  
    //atributos y métodos de la clase  
}
```

El objeto Amigo se ha marcado como serializable, ahora Java se encargará de realizar la serialización de forma automática.

2.3 - Serialización de objetos

Imaginemos la clase “Amigo” que guarda el nombre y el teléfono:

```
public class Amigo implements Serializable{

    private String nombre;
    private long telefono;

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

    public void datos_amigo() {
        System.out.println(nombre + " -> " + telefono);
    }

}
```

Desde la clase “Principal” crearemos 2 objetos de la clase “Amigo”, los escribiremos en un fichero (serializar) y luego leeremos del mismo fichero los objetos (deserializar).

2.3- Serialización de objetos

Para **serializar**, creamos un objeto de tipo File (como antes) y ahora, además:

- **FileOutputStream**: crea un flujo de datos para escribir en File.
- **ObjectOutputStream**: nos permite escribir objetos en ese flujo de datos.

```
public static void main(String[] args) {  
    File f = new File("C:\\Users\\franp\\Desktop\\amigos.txt");  
    try {  
        FileOutputStream fs = new FileOutputStream(f);  
        ObjectOutputStream oos = new ObjectOutputStream(fs);  
  
        Amigo a = new Amigo("Paco Perez", 655643140);  
        oos.writeObject(a);  
        Amigo a2 = new Amigo("Juan Perez", 666641123);  
        oos.writeObject(a2);  
        oos.close();  
        fs.close();  
    }  
    catch(IOException e) {  
        e.printStackTrace();  
    }  
}
```

amigos.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
-í [sr [Amigo][µ·N-^ J [telefonoL [nombret [Ljava/lang/String;xp 'R)t  
Paco Perezsq ~ '%ãt  
Juan Perez
```

2.3- Serialización de objetos

Para **deserializar**, creamos los siguientes objetos

- **FileInputStream**: crea un flujo de datos para leer de File.
- **ObjectInputStream**: nos permite leer objetos en ese flujo de datos.

```
try {  
    FileInputStream fis = new FileInputStream(f);  
    ObjectInputStream ois = new ObjectInputStream(fis);  
  
    Amigo a = (Amigo)ois.readObject();  
    a.datos_amigo();  
  
    Amigo a2 = (Amigo)ois.readObject();  
    a2.datos_amigo();  
  
    ois.close();  
    fis.close();  
}  
catch(Exception e) {  
    System.out.println("Excepción: " + e.getMessage());  
}
```

```
<terminated> Principal (24) [Java Application]
```

```
Paco Perez -> 655643140
```

```
Juan Perez -> 666641123
```

2.4.- Utilización de los sistemas de ficheros

2.4- Utilización de los sistemas de ficheros

Creación de un directorio con mkdir

```
public static void main(String[] args) {  
    File carpeta = new File("C:\\Users\\franp\\Desktop\\micarpeta");  
    boolean exito = carpeta.mkdir();  
    if (exito) {  
        System.out.println("Directorio " + carpeta + " creado!");  
    }else {  
        System.out.println("Directorio " + carpeta + " no creado!");  
    }  
}
```

Agregar archivos al directorio recién creado con createNewFile

```
File Archivo = new File("C:\\Users\\franp\\Desktop\\micarpeta\\fichero.txt");  
File Archivo2 = new File("C:\\Users\\franp\\Desktop\\micarpeta\\fichero2.txt");  
try {  
    Archivo.createNewFile();  
    Archivo2.createNewFile();  
} catch (IOException e) {  
    System.out.println(e.getMessage());  
}
```

2.4- Utilización de los sistemas de ficheros

Listar los ficheros de un directorio con list

```
String[] ficheros = carpeta.list();  
for (int i=0; i< ficheros.length; i++) {  
    System.out.println(ficheros[i]);  
}
```

```
Directorio C:\Users\franp\Desktop\micarpeta creado!  
fichero.txt  
fichero2.txt
```

Renombrar archivo o carpeta con RenameTo

```
File nueva_carpeta = new File("C:\\Users\\franp\\Desktop\\nueva_carpeta");  
boolean resultado = carpeta.renameTo(nueva_carpeta);  
if (resultado) {  
    System.out.println("Carpeta renombrada con éxito");  
}else {  
    System.out.println("No se pudo renombrar la carpeta");  
}
```

