TEMA 6

Tratamiento de Excepciones

- Y estructuras de datos externas

TRATAMIENTO DE EXCEPCIONES.

Cuando un programa Java viola las restricciones semánticas del lenguaje (**se produce un error**), la máquina virtual Java (MVJ) comunica este hecho al programa mediante una **excepción**.

Una excepción, puede ser provocada por diferentes tipos de errores, desde un desbordamiento de memoria o un disco duro estropeado, un intento de dividir por cero o "simplemente" intentar acceder a un vector fuera de sus límites.

Tratamiento de excepciones

- Cuando esto ocurre, la máquina virtual Java **crea un objeto de la clase** Exception o Error y se notifica el hecho al sistema de ejecución.
- Se dice que se ha lanzado una excepción ("*Throwing Exception*").
- Un **método se dice que es capaz de tratar una excepción** ("*Catch Exception*") si ha previsto el error que se ha producido y prevee también las operaciones a realizar para "recuperar" el programa de ese error.

Tratamiento de excepciones

- En el momento en que es lanzada una excepción, la **MVJ** recorre la pila de llamadas de métodos en busca de alguno que sea capaz de tratar la clase de excepción lanzada.

1.- Examina el método donde se ha producido la excepción; si este método no es capaz de tratarla, examina el método desde el que se realizó la llamada al método donde se produjo la excepción y así sucesivamente hasta llegar al último de ellos.

Tratamiento de excepciones

2.- En caso de que ninguno de los métodos de la pila sea capaz de tratar la excepción, la (MVJ) muestra un mensaje de error y el programa termina.

Los programas escritos en Java también pueden **lanzar excepciones** explícitamente mediante la **instrucción throw**, lo que facilita la devolución de un “código de error” al método que invocó el método que causó el error.

COMO PRIMER ENCUENTRO CON LAS EXCEPCIONES, EJECUTA EL SIGUIENTE PROGRAMA:

```
class Excepcion {
    public static void main(String arg[]) {
        int i = 5, j = 0;
        int k = i / j;   // División por cero
    }
}
```

Produce la siguiente salida al ser ejecutado:

java.lang.ArithmeticException: / by zero at Excepcion.main (Excepcion.java: 6)

Tratamiento de excepciones

- Lo que ha ocurrido es que la **MVJ** ha detectado una condición de error y ha creado un objeto de la clase java.lang. ArithmeticException.
- Como el método donde se ha producido la excepción **no es capaz de tratarla**, lo trata, la MVJ que **muestra el mensaje de error anterior y finaliza la ejecución** del programa.

Lanzamiento de excepciones (throw).

```
class LanzaExcepcion {
    public static void main(String args[]) throws ArithmeticException {
        int i = 1, j = 2;
        if (i / j < 1) {
            throw new ArithmeticException();
        } else {
            System.out.println(i / j);
        }
    }
}
```

Genera el siguiente mensaje:

java.lang.ArithmeticException at LanzaExcepcion.main (LanzaExcepcion.java:7)

Como el método donde se ha producido la excepción no es capaz de tratarla, se trata por la MVJ, que muestra el mensaje de error anterior y finaliza la ejecución del programa.

Tratamiento de excepciones.

En Java, se pueden tratar las excepciones previstas por el programador utilizando los manejadores de excepciones, que se estructuran en tres bloques:

El bloque **try.**

El bloque **catch.**

El bloque **finally**.

Un manejador de excepciones es una porción de código que se va a encargar de tratar las posibles excepciones que se puedan generar.

Tratamiento de excepciones

El bloque try.

Lo primero que hay que hacer para que un método sea capaz de tratar una excepción generada por la **MVJ** o por el propio programa mediante una instrucción throw es **encerrar las instrucciones susceptibles de generar el error en un bloque try**.

```
try {
        Bloque De Intrucciones
    }
```

Tratamiento de excepciones

- Cualquier excepción que se produzca dentro del bloque try será analizada por el bloque o bloques **catch** .
- En el momento en que se produzca la excepción, se abandona el bloque try y, por lo tanto, las instrucciones que sigan al punto donde se produjo la excepción no se ejecutaran**.**
- **Cada bloque try debe tener asociado por lo menos un bloque catch.**

Tratamiento de excepciones

- ```
  try {
        BloqueDeIntrucciones
        }
  catch (TipoExcepción nombreVariable) {
              BloqueCatch
         }
  catch (TipoExcepción nombreVariable) {
              BloqueCatch
         }
  ```

Para declarar el tipo de excepción se declara un objeto cuya clase es la clase de la excepción que se desea tratar o una de sus superclases.
  ```

Tratamiento de excepciones

Ejemplo: Try y catch

```
class ExcepcionTratada {
 public static void main(String arg[] ) {
int i=5, j=0;
  try {
        int k=i/j;
        System.out.println("Esto no se va a ejecutar.");
       }
catch (ArithmeticException ex) {
           System.out.println("Has intentado dividir por cero");
           }
System.out.println("Fin del programa"); }// Fin main
}// Fin clase
```

Salida:

Has intentado dividir por cero

Fin del programa

Tratamiento de excepciones

- Como hemos visto, un método también es capaz de lanzar excepciones.

Por ejemplo: El programa genera una condición de error si el dividendo es menor que el divisor:

En primer lugar, es necesario declarar todas las posibles excepciones que se pueden generar en el método, utilizando la cláusula **throws** de la declaración de métodos.

- Para lanzar la excepción es necesario **crear** un objeto de tipo Exception o alguna de sus subclases. (Por ejemplo: ArithmeticException)

Resumen excepciones

- Es una condición anormal durante la ejecución de un segmento de código.
- En java, una excepción es un objeto que almacena información y la transmite fuera de la secuencia normal del retorno de datos.
- Dicho objeto se envía al causante de la excepción, que puede tratar la aparición de la misma o no tratarla dejando que sea el sistema (dando un error) el encargado de ello.
- Pueden ser generadas por el intérprete de java o por el propio código.
- Para lanzar una excepción en java se utiliza la instrucción *throw* y previamente es necesario instanciar un objeto de la subclase de *Exception* apropiada.

Ventajas del tratamiento de excepciones

- Separación del código “útil” del tratamiento de errores.
- Propagación de errores a través de la pila de métodos.
- Agrupación y diferenciación de errores.
- Claridad del código y obligación del tratamiento de errores.

Excepcion 1 muestra como capturar una excepcion generica.

```
import java.util.Scanner;

public class Excepcion1 {

    public static void main(String arg[]) {
        Scanner sc = new Scanner(System.in);
        int i, j;
        boolean error = false;
        do {
            try {
                error = false;
                System.out.println("Teclea un numero");
                i = sc.nextInt();
                System.out.println("Teclea otro numero");
                j = sc.nextInt();
                System.out.println("La division de " + i + " por " + j + " es " + (i / j));
            } catch (Exception e) {//Imprime un texto del tipo de error
                System.out.println("Error: " + e.getMessage()
                );
                error = true;
            }
        } while (error);
    }
}
```

Excepcion 2 muestra como capturar una excepcion de forma selectiva.

```
import java.util.Scanner;
public class Excepcion2 {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int i, j;
        boolean error = false;
        do {
            try {
                error = false;
                System.out.println("Teclea un numero");
                i = sc.nextInt();
                System.out.println("Teclea otro numero");
                j = sc.nextInt();
                System.out.print("La division de " + i + " por " + j + " es " + (i / j));
            } catch (NumberFormatException e) {
                System.out.println("Error en la conversión a valor numérico");
                error = true;
            } catch (ArithmeticException e) {
                System.out.println("División por cero");
                error = true;
            } catch (Exception e) {
                System.out.println("Excepción desconocida");
                error = true;
            }
        } while (error);
    }
}
```

FICHEROS : CONCEPTOS BÁSICOS

Archivo. Un archivo es un **conjunto de datos estructurados** en una colección de entidades elementales básicas denominadas **registros** que son de igual tipo y constan a su vez de diferentes entidades de nivel más bajo denominadas **campos**

Clave (indicativo). Una clave o indicativo es un campo de datos que identifica unívocamente al registro y lo diferencia de otros registros. Esta clave debe ser diferente para cada registro. Claves típicas son números de identificación.

FICHEROS: CONCEPTOS BÁSICOS

- **Registro físico o bloque**.Un registro físico o bloque es la **cantidad de información que pueden transferirse en una operación de entrada/salida** entre la memoria central y los dispositivos periféricos o viceversa. Un bloque puede contener uno o más registros lógicos.

- **Factor de bloqueo.** El número de registros lógicos que puede contener un registro físico se denomina factor de bloqueo. Un registro lógico puede ocupar menos de un registro físico, exactamente un registro físico o más de un registro físico.

ORGANIZACIÓN DE ARCHIVOS

- **Organización secuencial.** Es una secuencia de registros almacenados consecutivamente sobre el soporte externo, de tal modo que para acceder a un registro determinado es obligatorio pasar por todos los registros que le preceden.
- **Todos los tipos de dispositivos de memoria auxiliar soportan la organización secuencial.**
- Los archivos organizados secuencialmente contienen un registro particular, el último, que contiene una marca de fin de archivo, detectable mediante la función de **EOF** FDA/ FDF.

FICHEROS : CONCEPTOS BÁSICOS.

- **Organización directa.** Un archivo está organizado en modo directo cuando el orden físico no se corresponde con el orden lógico. Los datos se sitúan en el archivo y se accede a ellos directamente mediante su posición, es decir, el lugar relativo que ocupan.
- Esta organización tiene la ventaja de que se pueden leer y escribir registros en cualquier orden y posición, y el acceso a la información que contienen es más rápido.

ORGANIZACIÓN DE ARCHIVOS

- La organización directa tiene el inconveniente de que necesita programar la relación existente entre el contenido de un registro y la posición que ocupa.
- El acceso a los registros en modo directo **implica la posible existencia de huecos libres** dentro del soporte y, por consecuencia, pueden existir huecos libres entre registros

TIPOS DE ACCESO.

Según las características del soporte empleado y el modo en que se han organizado los registros, se consideran dos tipos de acceso a los registros de un archivo:

El acceso secuencial implica el acceso a un archivo según el orden de almacenamiento de sus registros.

El acceso directo implica el acceso a un registro determinado, sin que ello implique la consulta de los registros anteriores. Este tipo de acceso sólo es posible en soportes direccionables.

En general, se consideran tres organizaciones fundamentales: **Secuencial , Directa o aleatoria y Secuencial indexada.**

ARCHIVOS EN JAVA

- Las **clases** que utiliza Java para el manejo de archivos, están todas basadas en el concepto de **stream o flujo de datos,** entre el programa y el destino u origen de datos.

LA CLASE FILE

- **La clase File** y otras del paquete **java.io** gestionan el **acceso a archivos.**
- **La clase File** permite el acceso, no al contenido, sino a sus características generales como permisos, tamaño, localización (getAbsolutePath), canWrite etc.
- Constructores y métodos básicos.

STREAMS

- Existen varios tipos de Streams (flujo) , y para casi todos los tipos de Streams de lectura existe el correspondiente de escritura.

Tipos de streams.

- Según su contenido
- Según su propósito

SEGÚN SU CONTENIDO:

- **Character Streams** utilizados para **archivos de texto** Se les llama readers o writers.
- **Byte Streams** utilizados para **archivos binarios** (datos, sonido, imágenes etc.) Se les llama input Streams o output Streams.

SEGÚN SU PROPÓSITO:

- **Data Sink Streams**, Son fuentes u orígenes de datos; son los que deben estar asociados a los archivos de disco.

- **Processing Streams,** Realizan algún tipo de proceso (almacenar en el buffer, descodificar caracteres, etc) de los datos leídos o escritos de una fuente.

JERARQUÍA DE CHARACTER STREAMS:

- **Reader**
  - BufferedReader
  - InputStreamReader
  - FilterReader
- **Writer**
  - BufferedWriter
  - OutputStreamWriter
  - FilterWriter
  - PrintWriter

JERARQUÍA DE LOS BYTE STREAMS

- **InputStream**
  - FileInputStream
  - FilterInputStream
  - ObjectInputStream
- **OutputStream**
  - FileOutputStream
  - FilterOutputStream
  - ObjectOutputStream.

FILEREADER / FILEWRITER

- Son Character Stream similares a los Byte Stream InputStream y OutputStream.
- Si al crear un flujo de salida el archivo de destino existe se destruye, si no existe se crea.
- Si se producen problemas al crear el archivo se genera una excepción del tipo IOException.

BUFFEREDREADER / BUFFEREDWRITER

- Se utilizan para evitar que cada lectura o escritura acceda directamente al archivo, ya que utilizan un buffer intermedio entre memoria y el Stream.
- Son Stream de procesamiento.

INPUTSTREAMREADER / OUTPUTSTREAMWRITER

- Estas clases sirven de unión entre Character Stream y Byte Streams.
- Los caracteres escritos en un OutputStreamWriter son transformados en bytes, y los bytes leídos de un InputStreamReader son transformados en caracteres.

RANDOMACCESSFILE

- Esta clase se utiliza para leer y escribir en archivos con posicionamiento **aleatorio.**
- Para el posicionamiento utiliza un puntero que debe colocarse en la posición a partir de la cual se quiere leer o escribir, después de la operación de lectura o escritura el puntero se habrá desplazado tantos bytes como se hayan escrito o leído, igual que ocurre con el tratamiento secuencial.
- Si durante la lectura se llega al final de archivo , se lanza una excepción EOFException (subtipo de la IOException).

OBJECT STREAMS

- Si deseamos escribir objetos completos en un Stream, debemos utilizar los ObjectStream.
- Las dos clases básicas son:
  - **ObjectOutputStream**
  - **ObjectInputStream**

OBJECT STREAMS

- Los datos que se escriben con esta clase, además de los valores de los atributos del objeto, guardan información de control para identificar el objeto.
- Para que un objeto pueda ser escrito utilizando Object Stream debe implementar el interfaz ***Serializable.***
- Se pueden escribir distintos tipos de objetos en un mismo archivo.