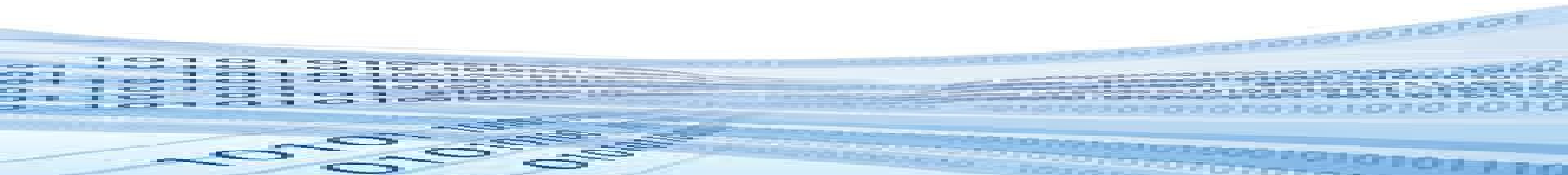
UD9.- Gestión de excepciones

Módulo: Programación
1.º DAM

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

CONTENIDOS

- Errores y excepciones
- Excepciones en Java
- Gestión de excepciones
  - Capturar excepciones
  - Propagar excepciones
  - Lanzar excepciones
  - Crear clases de excepciones



Errores y excepciones (I)

- **Errores**
  - Un error es un evento que se produce a lo largo de la ejecución de un programa y provoca una interrupción en el flujo de ejecución. Al producirse esta situación, el error genera un objeto **Exception**.

- **Excepciones**
  - Una excepción (Exception) es un objeto generado por un error, que contiene información sobre las características del error que se ha producido.

Errores y excepciones (II)

- Ejemplos de excepciones
  - Indexación de array fuera de rango
  - Referencia a ningún objeto (null)
  - Errores de formato
  - Errores en operaciones matemáticas (división por 0, ...)
  - El archivo que queremos abrir no existe
  - Falla la conexión en una red
  - La clase que se quiere utilizar no se encuentra en ninguno de los paquetes utilizados desde los import

Errores y excepciones (III)

- Hasta ahora nuestros programas no gestionan las excepciones, si se producen:
  - 1.- Se informa por consola del problema (se muestra la traza)
  - 2.- Se para la ejecución.

Errores y excepciones. Ejemplo

- Dado el siguiente código:

```
System.out.print("Valor: ");
int valor = lector.nextInt();
System.out.print("Hemos leido : "+valor);
```

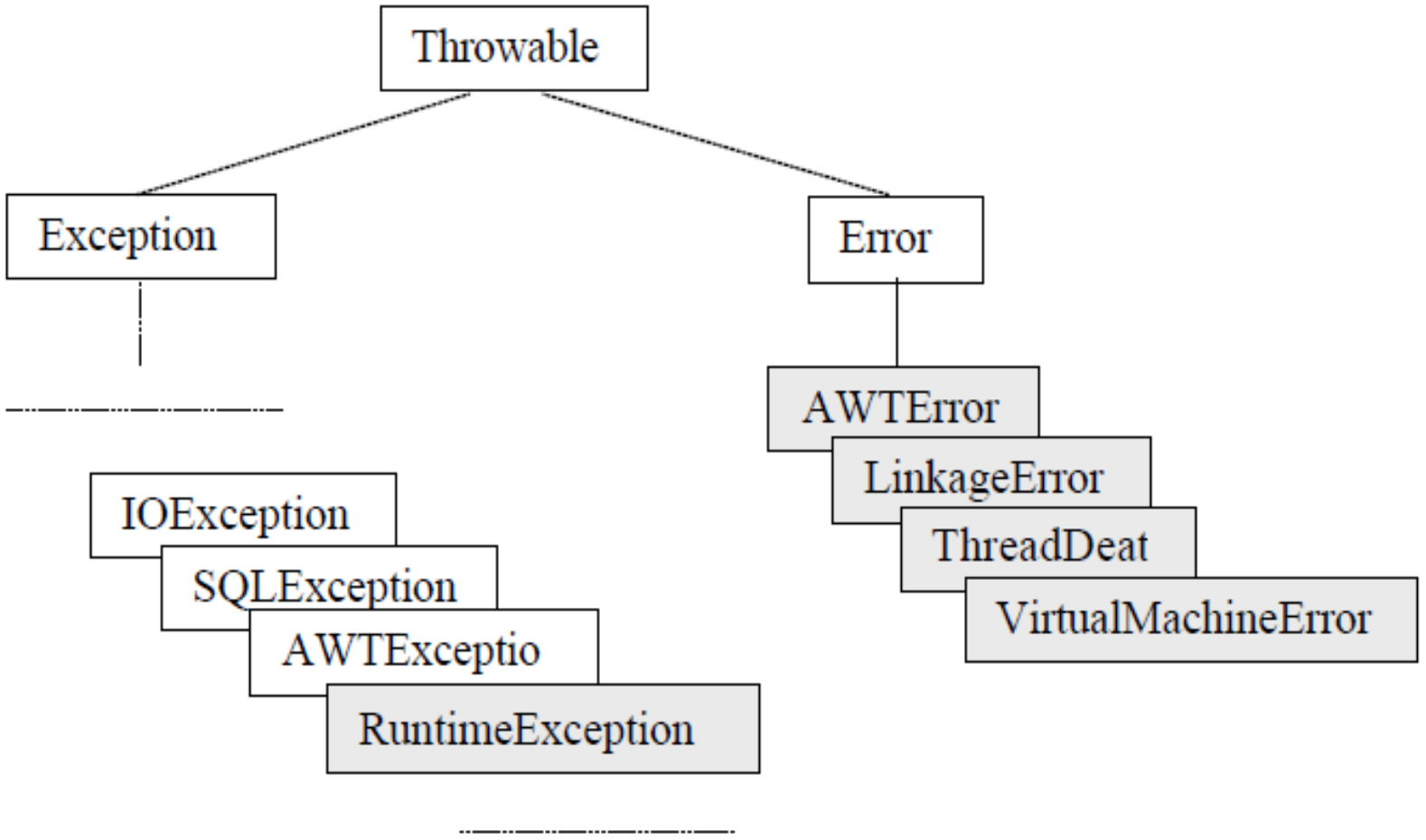
- Si lo ejecutamos e introducimos caracteres no numéricos se lanza excepción InputMismatchException

```
Valor: hola
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:909)
        at java.util.Scanner.next(Scanner.java:1530)
        at java.util.Scanner.nextInt(Scanner.java:2160)
        at java.util.Scanner.nextInt(Scanner.java:2119)
        at unidad07.SinExcepcion1.main(SinExcepcion1.java:16)
```

Excepciones en Java (I)

- Java lanza excepciones como respuesta a situaciones poco habituales.
- El programador también puede lanzar sus propias excepciones.
- Las excepciones en Java son objetos de clases derivadas de la clase base ***Exception***
- La clase ***Exception*** deriva de la clase base ***Throwable*** (java.lang.Throwable)

Excepciones en Java (II)



Excepciones en Java (III)

- Existe toda una jerarquía de clases derivada de la clase base *Exception*.
- Las excepciones se dividen en dos grupos principales:
  - Excepciones en tiempo de ejecución o RuntimeExceptions (excepciones implícitas)
  - Excepciones explícitas

Excepciones implícitas

- Java las comprueba a lo largo de la ejecución y las lanza de forma automática. No hay que realizar un tratamiento explícito, es decir, Java no obliga a tratarlas.
- Estas excepciones le indican al programador qué tipos de errores tiene el programa para que los solucione antes de continuar.
- Ejemplo:
  - Cuando se sobrepasa la dimensión de un array, se lanza una excepción *ArrayIndexOutOfBounds*.
  - Cuando se hace uso de una referencia a un objeto que todavía no ha sido creado, se lanza la excepción *NullPointerException*.

Excepciones explícitas

- Java **obliga a tratarlas** si se producen.
- Ejemplo:
  - *IllegalAccessException* indica que no se puede encontrar un método.
  - IOException cuando se produce error de entrada/salida (ejemplo de BufferedReader).

Algunas excepciones *RuntimeException*

Clase	Situación de excepción
NumberFormatException	Indica que una aplicación ha intentado convertir una cadena a un tipo numérico, pero la cadena no tiene el formato apropiado.
ArithmeticException	Cuándo ha tenido lugar una condición aritmética excepcional, como por ejemplo una división por cero.
ArrayStoreException	Para indicar que se ha intentado almacenar un tipo de objeto erróneo en un array de objetos.
IllegalArgumentException	Indica que a un método le han pasado un argumento ilegal.
IndexOutOfBoundsException	Indica que un índice de algún tipo (un array, cadena) está fuera de rango.
NegativeArraySizeException	Si una aplicación intenta crear un array con medida negativa.
NullPointerException	Cuando una aplicación intenta utilizar *null* donde se requiere un objeto.
InputMismatchException	Lanzada por Scanner para indicar que el valor recuperado no coincide con el patrón esperado.

Excepciones y packages.

- Las clases derivadas de Exception pueden pertenecer a diferentes *packages* (agrupaciones de clases) de Java, dependiendo del tipo de excepción que representan:
  - package java.lang
  - package java.io
  - package java.util
  - …

Capturar excepciones

- Detecta y responde a errores mientras se ejecuta una aplicación.
- Utiliza ***try***…***catch***…***finally*** para encapsular y proteger bloques de código que podrían provocar errores:
  - Cada bloque tiene uno o más controladores asociados.
  - Cada controlador especifica alguna forma de condición de filtro en el tipo de excepción que controla.
- Ventajas:
  - Permite la separación entre la lógica y el código de gestión de errores.
  - Facilita la lectura, depuración y mantenimiento del código.

Como utilizar la estructura *try...catch* (I)

- Colocar el código que podría lanzar excepciones en un bloque try.
- Gestionar las excepciones en uno o más bloques catch.

```
try {
   código que puede provocar errores

} catch (ExceptionA a) {
  ....
} catch (ExceptionB b) {
  ....
}
```

Como utilizar la estructura *try…catch* (II)

- Desde el bloque *catch* se maneja la excepción.
- Cada *catch* maneja un tipo de excepción.
- Cuando se produce una excepción, se busca el primer *catch* que utilice el mismo tipo de excepción que se ha producido:
  - El último *catch* tiene que ser el que capture excepciones genéricas y los primeros tienen que ser los más específicos.
  - Si vamos a tratar todas las excepciones (sean del tipo que sean), tenemos que pensar si con un catch que capture objetos *Exception* nos vale, pero generalmente no suele ser una buena práctica.

Como utilizar la estructura *try...catch* (III)

```
String[] texto = {"Uno", "Dos", "Tres", "Cuatro", "Cinco"};

for (int i = 0; i < 10; i++) {

   try {

      System.out.println("indice " + i + " = " + texto[i]);

   } catch (ArrayIndexOutOfBoundsException e) {

      System.out.println("Fallo en el índice " + i);
   }
}
```

Como utilizar la estructura *try...catch* (IV)

```
try {
   System.out.print("Valor: ");
   int valor = lector.nextInt();
   int auxiliar = 8 / valor;
   System.out.println(auxiliar);

} catch (ArithmeticException e) {
   if(valor == 0)
      System.out.println("Division por cero");
   else
      System.out.println("Excepción aritmética");
}
```

Como utilizar la estructura *try...catch* (V)

```
try {
    System.out.print("Valor: ");
    int valor = lector.nextInt();
    int auxiliar = 8 / valor;
    System.out.println(auxiliar);
} catch (ArithmeticException e1) {
    System.out.println("Division por cero");
} catch (InputMismatchException e2) {
    System.out.println("No se ha leído un entero...");
} catch (Exception e9) {
    System.out.println("Error general");
} finally {
    lector.nextLine();
}
```

Como utilizar la estructura *try...catch* (VI)

```
int valor = 0;
boolean leido = false;
Scanner lector = new Scanner(System.in);
do {
    try {
        System.out.print("Entra un número entero: ");
        valor = lector.nextInt();
        leido = true;
    } catch (InputMismatchException e) {
        System.out.println("Error en la introducción del número");
        lector.nextLine(); //vaciamos el buffer de entrada
    }
} while (!leido);
System.out.println("Hemos leido : " + valor);
```

Como utilizar el bloque *finally*

- Es opcional; si se incluye, se ejecuta siempre.
- Emplearlo, por ejemplo, para colocar código de “limpieza”, como el que se emplea para cerrar archivos.

```
try {
  //código que puede provocar errores

} catch (ExceptionA a) {
 ....

} catch (ExceptionB b) {
 ....

} finally {

  //código que se ejecuta siempre
}
```

Algunos métodos de *Exception*

- **String getMessage()**
  - Recupera el mensaje descriptivo de la excepción o una indicación específica del error producido.

```
try{
....
} catch (IOException ioe){
    System.out.println(ioe.getMessage());
}
```

- **String toString()**
  - Convierte a cadena la información del error. Normalmente contiene la clase de excepción y el texto de getMessage().

```
try{
....
} catch (IOException ioe){
    System.out.println(ioe.toString());
}
```

- **void printStackTrace()**
  - Escribe la traza de invocaciones que ha provocado la excepción y su mensaje asociado (es lo que se denomina información de pila).
  - Es el mismo mensaje que muestra el ejecutor (máquina virtual de Java) cuando no se controla la excepción.

Directrices para la captura de excepciones

- No emplear la gestión estructurada de excepciones para errores que se produzcan de forma rutinaria. En estos casos es mejor utilizar otros bloques de código para controlar estos errores.
  - if ...else if, etc.
- Organizar los bloques *catch* desde los más específicos hasta los más generales.

Propagar excepciones

- Cuando se produce un error:
  - Se puede capturar en el método en el cual se ha producido o
  - se puede propagar hacia el método invocador para que lo capture. Si este no lo captura, se propaga sucesivamente hasta el main() y el main(), si no lo captura, lo propaga hasta el sistema operativo.
- Solo se propagan los errores que derivan de la clase RuntimeException. El resto no se propagan y se deben capturar o lanzar.

Ejemplo (I)

```
importe java.util.Scanner;

public class Prueba1 {

  public static int llegirNumero(){

    Scanner reader=new Scanner(System.in);

    int num=0;

    try {

      System.out.print("Inserta un número: ");

      num=Integer.parseInt(reader.nextLine());

      System.out.println("Número correcto");

    } catch (NumberFormatException e){

      System.out.println("Solo se pueden
introducir números.");

    }

   return num;

  }

  public static void main(String[] args) {

    try {

      llegirNumero();

    } catch (NumberFormatException e){

      System.out.println("Captura de la excepción
desde el main.");

    }

  }

} // Fin de la clase Prueba1
```

¿Qué muestra este programa si introducimos una letra?

Ejemplo (II)

```
importe java.util.Scanner;

public class Prueba1 {

  public static int llegirNumero(){

    Scanner reader=new Scanner(System.in);

    int num=0;

    System.out.print("Inserta un número: ");

    num=Integer.parseInt(reader.nextLine());

    System.out.println("Número correcto");

    return num;

  }

  public static void main(String[] args){

    try{

      llegirNumero();

    } catch (NumberFormatException e){

      System.out.println("Captura de la excepción
desde el main.");

    }

  }

} // Fin de la clase Prueba1
```

¿Qué muestra ahora el programa si introducimos una letra?

Lanzar excepciones (I)

- Situación 1.- Dentro de un método donde se puede producir uno de los errores que Java obliga a controlar (por ejemplo, *IOException*). Dos opciones:
  - Capturar y tratar el error con *try-catch*
  - Lanzar el error. Se expresa de la siguiente forma en la cabecera del método:

```
tipoRetornado nombreMétodo (parámetros) throws Excepción
tipoRetornado nombreMétodo (parámetros) throws Excepción1,Excepción2...
```

- Los errores que derivan de la clase **RuntimeException** (por ejemplo, NumberFormatException) no es necesario lanzarlos, ya que **se propagan automáticamente** si no han sido capturados.

Lanzar excepciones (II)

- Situación 2.- Dentro de un método queremos controlar un error, pero también queremos controlarlo fuera de ese método. Para hacerlo:
  - Capturar el error dentro del método (con **el try-catch**).
  - Dentro del catch, lanzar el error para que lo pueda recibir el método invocador.
  - Un error se lanza de la siguiente forma:

```
throw (nombreObjectoTipoExcepción);
```

  - La instrucción throw provoca que se abandone la ejecución del método donde se encuentra y pase el control al método invocador, al catch que captura el error.

Crear clases de excepciones

- Debe ser una subclase de java.lang.Throwable.
  - Ejemplo: `class MyMistakes extends Exception {...}`

- Es importante incluir información de por qué se crea (circunstancia excepcional que provoca excepción)

```
class MyMistakes extends Exception {

      public MyMistakes(String msg){
         super(msg);
      }
}
```