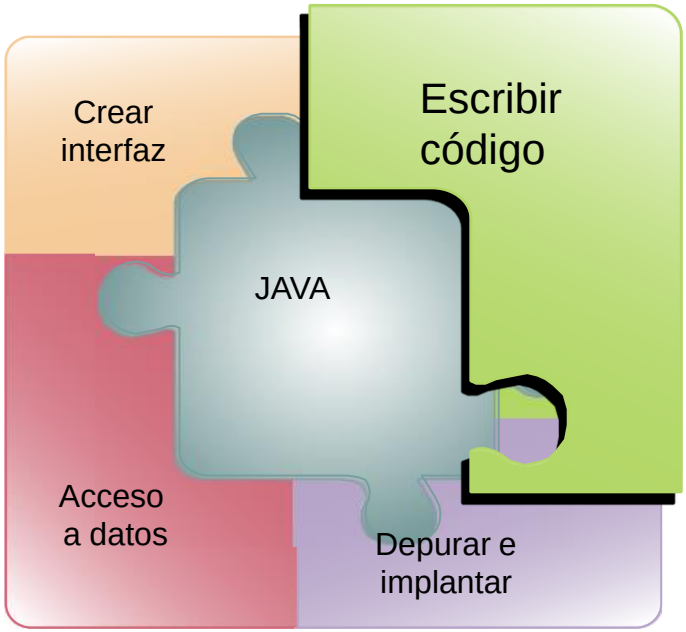
Gestión de Excepciones

- **Unidad 7**
- **Apuntes referenciados:**
  - ✓ A1.- Capítulo 6
  - ✓ A5.- Tema Excepciones (pág.. 71)
  - ✓ A3.- Las excepciones

Descripción



- Errores y Excepciones
- Excepciones en JAVA
- La gestión estructurada de excepciones
- Cómo utilizar la instrucción try…catch
- Cómo utilizar el bloque finally
- Algunos métodos de Exception
- Directrices para el uso de la gestión estructurada de excepciones
- Crear, Lanzar y Propagar una excepción

Errores y excepciones

- **Error**

  Un error es un evento que se produce durante el funcionamiento de un programa, provocando una interrupción en su flujo de ejecución. Al producirse esta situación, el error genera una excepción

- **Excepción**

  Una excepción es un objeto generado por un error, que contiene información sobre las características del error que se ha producido

Errores y excepciones

- **Ejemplos de errores:**
  - Indexación de un array fuera de rango
  - Referencia a ningún objeto (null)
  - Errores de formato
  - Errores en operaciones matemáticas (división por cero, etc…)
  - El archivo que queremos abrir no existe
  - Falla la conexión a una red
  - La clase que se desea utilizar no se encuentra en ninguno de los paquetes reseñados con import

Errores y excepciones

- **Si, tal como hemos hecho hasta ahora, nuestro programa no captura y gestiona la excepción:**
  - La excepción sale del programa, pasando al soporte de ejecución que:
    1. Informa en consola del problema (es decir, imprime la traza)
    2. Detiene la ejecución

Errores y excepciones. Ejemplo

- **Si ejecutamos**

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

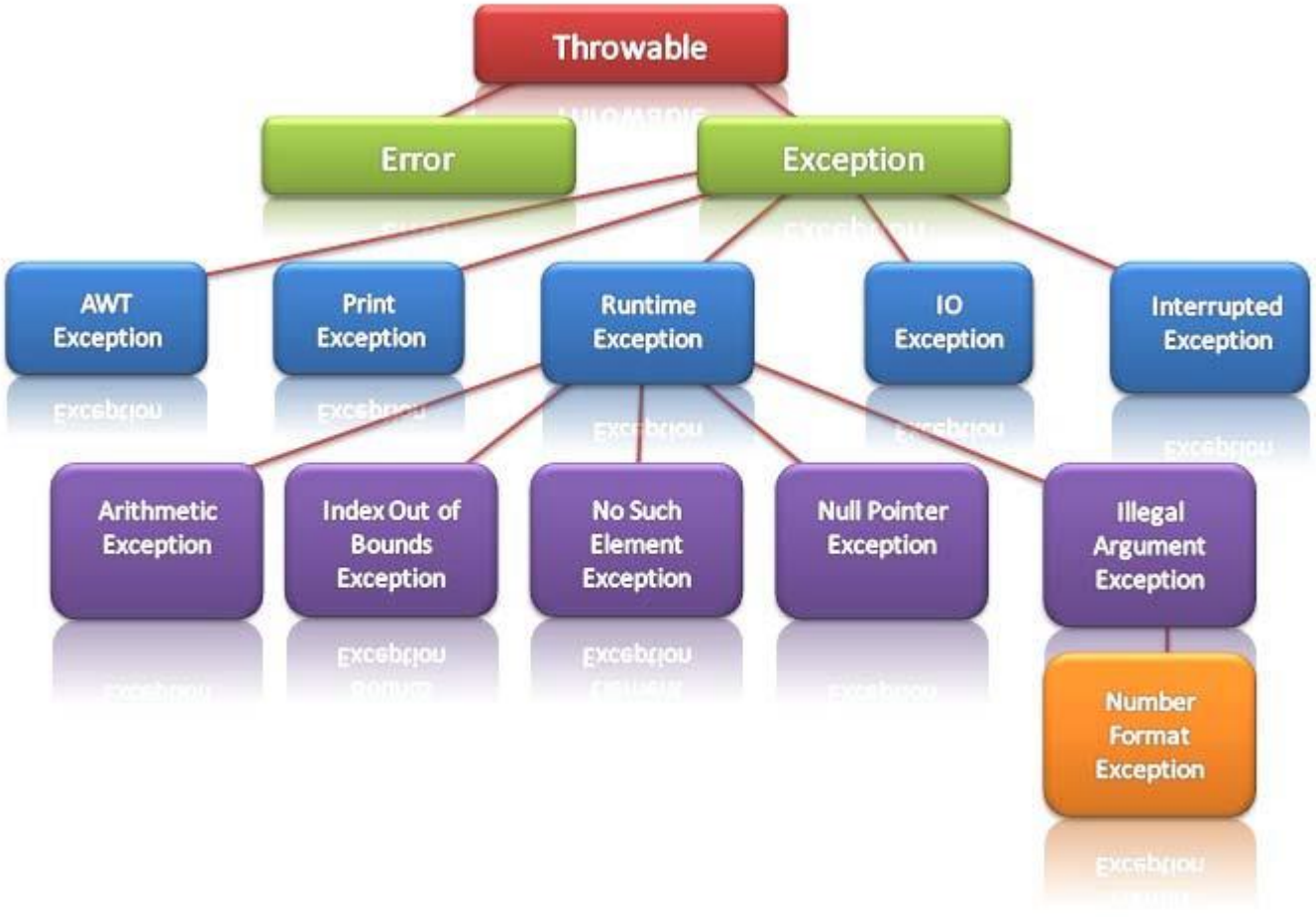
- **E introducimos caracteres no numéricos se lanza una excepción InputMismatchException**

```
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

- **Java lanza una excepción en respuesta a una situación poco usual**
  - Las excepciones en Java son objetos de clases derivadas de la clase base ***Exception***
  - Existen también los errores internos que son objetos de la clase ***Error*** que no estudiaremos
  - Ambas clases *Error* y *Exception* son clases derivadas de la clase base ***Throwable*** (java.lang.Throwable)
- **El programador también puede crear y lanzar sus propias excepciones**

Excepciones en Java



Excepciones en Java

- **Existe toda una jerarquía de clases derivada de la clase base *Exception***
- **Se dividen en dos grupos principales:**
  - **Implícitas o unchecked:**
    - Son las RuntimeException (o Excepciones en tiempo de ejecución)
    - JAVA las comprueba durante la ejecución lanzando de forma automática la clase de Excepción correspondiente
    - No es necesario realizar un tratamiento explícito
    - Por 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 no ha sido creado se lanza la excepción ***NullPointerException***
  - ***Excepciones explícitas o checked:***
    - Las demás clases derivadas de *Exception*
    - JAVA las comprueba en compilación
    - Es necesario tratarlas (o propagarlas)
    - Por ejemplo,
      - ***IllegalAccessException*** que indica que no se puede encontrar un método

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	cuando una condición aritmética excepcional ha ocurrido 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 (tal como un array, de una cadena o de un vector) está fuera de rango
NegativeArraySizeException	si una aplicación intenta crea un array con tamaño negativo
NullPointerException	cuando una aplicación intenta usar null donde se pide un objeto
InputMismatchException	Lanzada por Scanner para indicar que el valor recuperado no coincide con el patrón para el tipo esperado

Atención!

- **Las clases derivadas de *Exception pueden pertenecer a distintos packages (agrupación de* clases) de Java, dependiendo del tipo de Excepción que representen**
  - Package java.lang
  - Package java.io
  - Package java.util
  - …

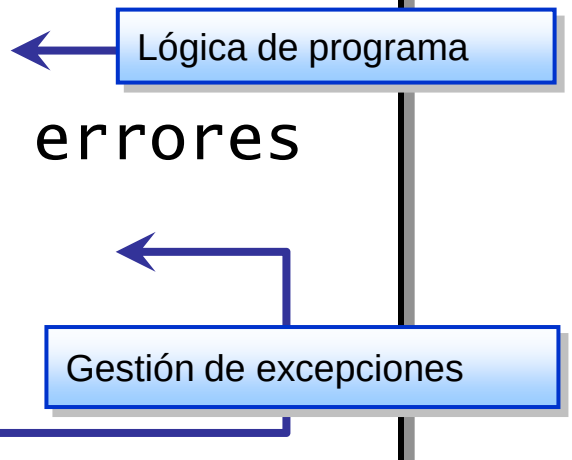
Gestión Estructurada de Excepciones

- **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

Cómo utilizar la instrucción try…catch

- Poner 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) {
      ....
}
```



Cómo utilizar la instrucción try…catch

- **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 debe ser el que capture excepciones genéricas y los primeros deben ser los más específicos
  - Si vamos a tratar igual a todas las excepciones (sean del tipo que sean) basta con un solo catch que capture objetos Exception

Cómo utilizar la instrucción try…catch

- **Ejemplo**

```
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);
          }
      }
```

Cómo utilizar la instrucción try…catch

- **Ejemplo**

```
try {
        System.out.print("Valor: ");
        int valor = lector.nextInt();
        int auxiliar = 8 / valor;
        System.out.println(auxiliar);
} catch (ArithmeticException e) {
        System.out.println("Division por cero");
 }
```

Cómo utilizar la instrucción try…catch

- **Ejemplo**

```
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();
 }
```

Cómo utilizar la instrucción try…catch

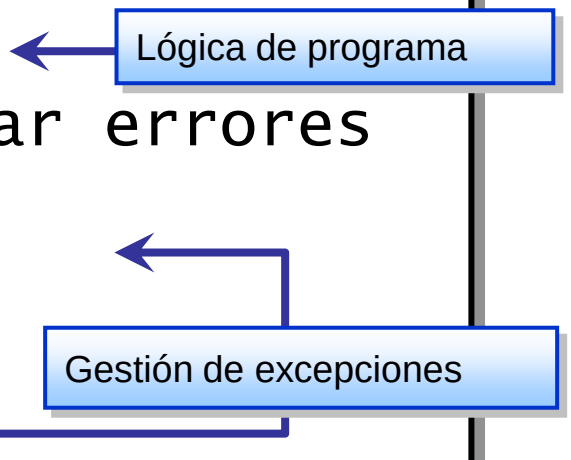
- **Ejemplo: Insistimos hasta que el valor introducido sea válido**

```
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);
```

Cómo utilizar el bloque finally

- Sección opcional; si se incluye, se ejecuta siempre
- Colocar código de limpieza, como el utilizado para cerrar archivos, en el bloque finally

```
try {
  'código que puede provocar errores
} catch (ExceptionA a) {
    ....
} catch (ExceptionB b) {
    ....
} finally {
 'código quese ejecuta siempre
}
```



Algunos métodos de Exception

- **String getMessage()**
  - Obtiene el mensaje descriptivo de la excepción o una indicación específica del error ocurrido
- **String toString()**
  - Escribe una cadena sobre la situación de la excepción. Suele indicar la clase de excepción y el texto de getMessage()
- **void printStackTrace()**
  - Escribe el método y mensaje de la excepción (la llamada información de pila)
  - El resultado es el mismo mensaje que muestra el ejecutor (la máquina virtual de Java) cuando no se controla la excepción

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

Directrices para el uso de la gestión estructurada de excepciones

No utilizar la gestión estructurada de excepciones para errores que se produzcan de modo rutinario. Utilizar otros bloques de código para abordar estos errores.

- if …else if, etc.

Organizar los bloques catch desde específicos hasta generales

Crear una objeto Excepción

- **El programador también puede crear y lanzar sus propias excepciones:**

- **Para crear un objeto de una clase excepción:**
  - Usaremos **new** (es decir, el constructor, como siempre)
  - Ejemplo:

```
new ArithmeticException( );
Exception error = new Exception( );
Exception correoErroneo = new Exception("E-MAIL ERRONEO !!!");
```

Crear una clase excepción

- **Atención: Aquí hay conceptos de herencia que aun no hemos visto (próximo tema)**
- **<u>Para crear una nueva clase excepción</u>:**
  - Debe ser de una subclase de Exception

    ```
    class MisErrores extends Exception { ... }
    ```

  - Entre sus campos suele llevar información de por qué se crea (o sea, de qué circunstancia excepcional es mensajero)

    ```
    class MisErrores extends Exception {
                public MisErrores  (String msg) {
                    super(msg);
                    }
    }
    ```

  - Un objeto de esta clase se crea con new (o sea, usando un constructor)

    ```
    MisErrores error = new MisErrores("Se ha producido un error… ");
    ```

Lanzar una excepción: throw

- **Para lanzar una excepción:**
  - Usamos la palabra reservada **throw**
  - El lanzamiento interrumpe la ejecución normal del código (es decir que la siguiente instrucción no se ejecuta)
  - Por ello el lanzamiento de la excepción suele estar en un bloque condicional

```
if (denominador  == false)
        throw new ArithmeticException( );
```

Propagar una excepción: throws

- **Cuando una excepción se lanza dentro de un método puede que este la capture o que no (que la propague)**
- **Si la excepción se propaga:**
  - En la cabecera del método se deben declarar el tipo de excepciones que puede generar y que deberán ser gestionadas por quien invoque a dicho método
  - Para ello utilizaremos la sentencia **throws** seguida de las excepciones que puede provocar el código del método
  - Si ocurre una excepción en el método, el código abandona ese método y regresa al código desde el que se llamó al método
  - Ejemplo:

```
void miMetodo (argumentos)  throws Exception {
        ...
        throw  new Exception( );
        ...
}
```

  - Si un método propaga una excepción, esta deberá ser considerada en el código llamante que, a su vez, puede capturarla o retransmitirla

Propagar una excepción: throws