UD9. Gestión de excepciones

Módulo: Programación

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Germán Gascón Grau
g.gascongrau@edu.gva.es

Contenidos

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

Errores vs Excepciones

- **Errores**

  Desde el punto de vista de Java, 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`.

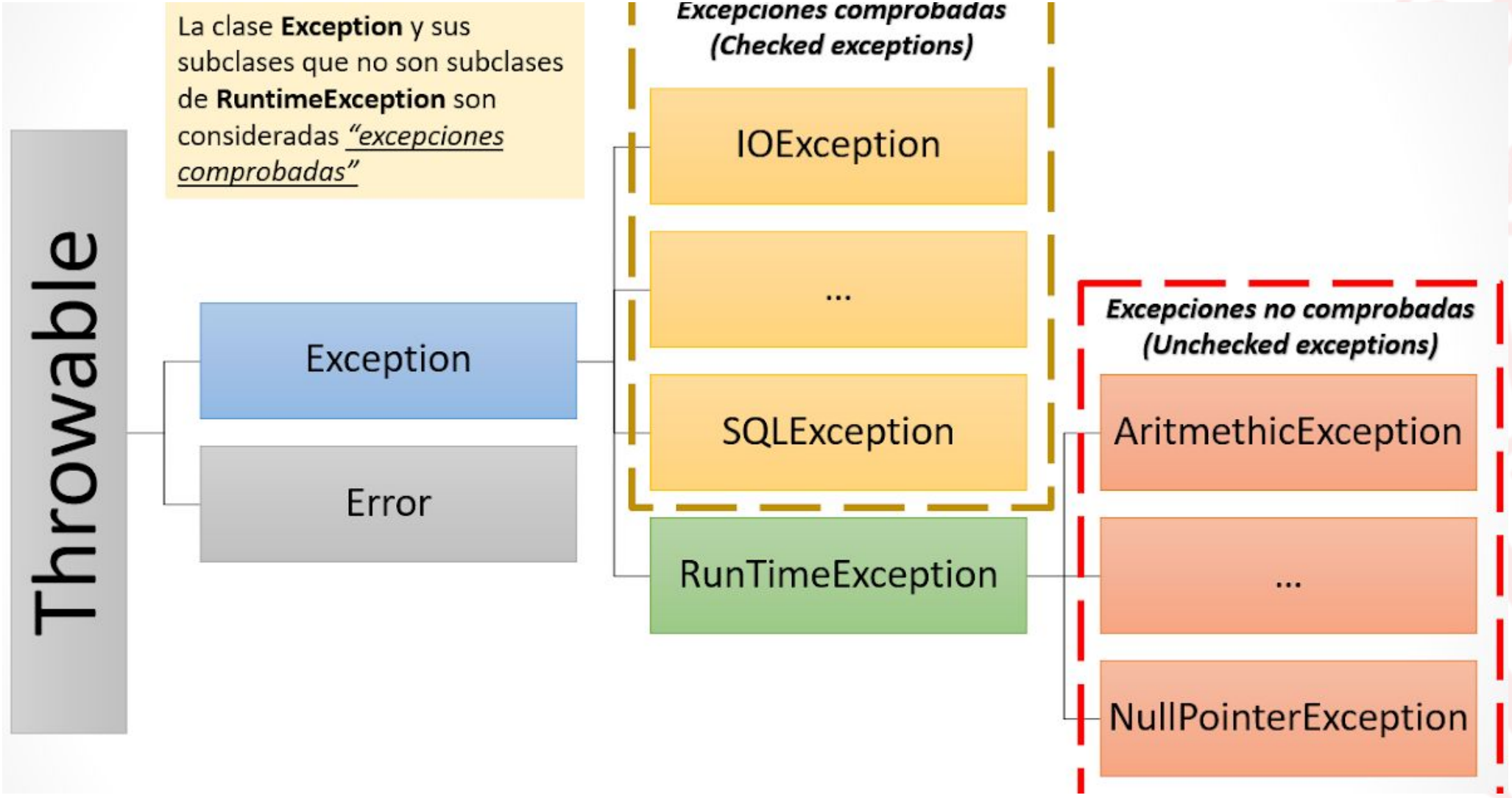
- **Excepciones**

- Una excepción es un objeto de la clase `Exception` o de alguna de sus clases derivadas, generado por un error, que **contiene información** sobre las características **del error** que se ha producido.

Ejemplos de Excepciones

- Algunas de las situaciones en las que se genera una `Exception` son:
  - 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

Jerarquía de Excepciones



Jerarquía de Excepciones

- 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)

Jerarquía de Excepciones

- 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` a su vez deriva de la clase base `Throwable` (java.lang.Throwable)
- Las excepciones pueden dividirse en:
  - Excepciones comprobadas (Checked Exceptions)
  - Excepciones no comprobadas (Unchecked Exceptions)
    - Derivan de `RuntimeException`

Excepciones comprobadas

- Son excepciones que **deben ser tratadas obligatoriamente** o de lo contrario provocarán un **error de compilación**.
- Algunos ejemplos de excepciones comprobadas son:
  - `IOException` cuando se produce un error de entrada/salida, por ejemplo al leer un fichero.
  - `FileNotFoundException` cuando intenta accederse a un archivo que no existe.
  - `InterruptedException` cuando se interrumpe un hilo de ejecución.
  - `SQLException` cuando se produce un error al acceder a una base de datos.

1100111001011001101010110011011010011001101011110011100011011001011001110010110011010101100110110100110011010111

Excepciones no comprobadas

- Son excepciones que **derivan** de la clase `RuntimeException`.
- Este tipo de excepciones son **comprobadas a lo largo de la ejecución** del programa (tiempo de ejecución) y Java las lanza de forma **automática**.
- **No es obligatorio tratarlas** pero **proporcionan información útil** al programador para que éste intente solucionar el problema.
- Si se produce una excepción de este tipo y no es tratada, el programa finalizará abruptamente.

Ejemplos de excepciones no comprobadas

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.

¿Qué ocurre si se produce una excepción?

- Cuando se produce una excepción y no es tratada:
  1. Se muestra por la salida estándar de error la traza de la pila de llamadas que ha provocado el error.
  2. Se detiene la ejecución del programa.
- Por este motivo nuestros programas finalizaban al producirse una `Exception`.

Ejemplo de excepción no comprobada

- Dado el siguiente código:

```
Scanner lector = new Scanner(System.in);
System.out.print("Valor: ");
int valor = lector.nextInt();
System.out.print("Hemos leído : "+valor);
lector.close();
```

- Si lo ejecutamos 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)
```

Directrices para el tratamiento de excepciones

- Todas las **excepciones comprobadas** deberán ser tratadas o de lo contrario producirán un error en tiempo de compilación.
- Las **excepciones no comprobadas** deberán ser tratadas en los siguientes casos:
  - El error es recuperable
  - El error no es directamente recuperable pero podemos transicionar a un estado que sí lo es.
  - El error no es recuperable pero podemos minimizar las consecuencias del error.
- **Nunca** debemos utilizar el tratamiento de excepciones para “solucionar” errores de programación, es decir, errores propios del programador o errores que pueden ser fácilmente comprobados mediante bloques if … else if, etc.

  Tratar excepciones cuyo origen son errores de programación, ocultan errores que posteriormente pueden ser desastrosos.

Tratamiento de excepciones

- Tratar una excepción es capturar/interceptar el objeto que contiene información sobre el error que se ha producido.
- Para capturar una excepción utilizaremos bloques de código **try** … **catch** … **finally**
  - Dentro del bloque **try** pondremos el código que queremos ejecutar.
  - En bloques **catch** pondremos cada una de las excepciones que queremos capturar y el código para tratar el error.
  - El bloque **finally** es opcional y lo utilizaremos para código que deberá ejecutarse siempre independientemente de si se ha producido una excepción o no.

Tratamiento de excepciones

- 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) {
  // código para tratar la ExceptionA
} catch (ExceptionB b) {
  // código para tratar la ExceptionB
} finally { // Opcional
  // código a ejecutar siempre
}
```

Tratamiento de excepciones con recursos

- Existe otra versión de try llamada **try-with-resources** que permite abrir recursos en el bloque try y cerrarlos automáticamente al finalizar el bloque.
- En el bloque try se pone entre paréntesis las sentencias que abren recursos y Java los cerrará automáticamente.

```
try (Scanner lector = new Scanner(System.in)) {
    System.out.print("Valor: ");
    int valor = Integer.parseInt(lector.nextLine());
    System.out.printf("El valor leído es %d\n", valor);
} catch (NumberFormatException nfe) {
    System.out.println("Introduzca sólo números por favor");
}
```

De esta forma no hace falta cerrar el Scanner ya que Java lo hará automáticamente.

- En el tema de archivos veremos este tipo de try en detalle.

Funcionamiento de los bloques 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 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.

Ejemplo tratamiento de excepción

```
Scanner lector = new Scanner(System.in);
double valor = Double.NEGATIVE_INFINITY;
boolean valido = false;
while (!valido) {
    try {
        System.out.print("Valor: ");
        valor = Double.parseDouble(lector.nextLine());
        valido = true;
    } catch (NumberFormatException nfe) {
        System.out.println("Introduzca sólo números por favor");
    }
}
System.out.println("Hemos leído: " + valor);
lector.close();
```

¿Que problema ves en el siguiente código?

```
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);
    }
}
```

Algunos métodos de la clase `Exception`

- `String getMessage()`

  Recupera el mensaje decriptivo de la excepción o una indicación específica del error producido.

- `String toString()`

  Convierte a cadena de caracteres la información del error. Normalmente contiene la clase de excepción y el texto de `getMessage()`

- `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.

Propagación de excepciones

- Cuando se produce un error:
  - Se puede capturar en el método en el cual se ha producido
  - O bien, 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 la JVM (Java Virtual Machine).
- Solo se propagan los errores que derivan de la clase RuntimeException. El resto no se propagan y se deben capturar o elevar/lanzar.

Ejemplo

```
public class Main {
    public static void main(String[] args) {
        try {
            leerNumero();
        } catch (NumberFormatException e){
            System.out.println("Captura de la excepción desde el main.");
        }
    }

    public static int leerNumero(){
        Scanner reader = new Scanner(System.in);
        int num = 0;
        try {
            System.out.print("Indica 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;
    }
}
```

¿Qué muestra este programa si introducimos una letra?

Ejemplo

```
public class Main {
    public static void main(String[] args) {
        try {
            leerNumero();
        } catch (NumberFormatException e){
            System.out.println("Captura de la excepción desde el
main.");
        }
    }

    public static int leerNumero() {
        Scanner reader = new Scanner(System.in);
        int num = 0;
        System.out.print("Indica un número: ");
        num = Integer.parseInt(reader.nextLine());
        System.out.println("Número correcto");
        System.out.println("Solo se pueden introducir números.");
        return num;
    }
}
```

¿Qué muestra este programa si introducimos una letra?

Delegar el tratamiento de excepciones

- Acabamos de ver que el mecanismo de propagación de excepciones va elevando la excepción en la pila de llamadas hasta llegar al método main quien finalmente, la propagará a la JVM y provocará que nuestro programa finalice.
- La palabra reservada **throws** permite indicar de forma explícita que un método no va a tratar una o varias excepciones.
- De esta forma, delegará en los métodos que invoquen a el método que lleva el **throws** para que sean ellos quien traten la excepción.
- La sintaxis es la siguiente:

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

Delegar el tratamiento de excepciones

- Puede darse el caso que dentro de un método queramos controlar un error, pero también queramos controlarlo fuera de ese método.
- Para ello debemos:
  - 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` objetoTipoExcepción;
  - 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.

Ejemplo delegación de excepciones

```
public class ApuntesMain {
    public static void main(String[] args) {
        try {
            leerNumero();
        } catch (Exception e){
            System.out.println(e.getMessage());
        }
    }
    public static int leerNumero() throws Exception {
        Scanner reader = new Scanner(System.in);
        int num = 0;
        try {
            System.out.print("Indica un número: ");
            num = Integer.parseInt(reader.nextLine());
            System.out.println("Número correcto");
        } catch (NumberFormatException e) {
            throw new Exception("Sólo se pueden introducir números");
        }
        return num;
    }
}
```

Excepciones personalizadas

- Podemos crear nuestras propias excepciones personalizadas creando clases derivadas (subclases) de la clase `Exception`.
- Al crear una excepción personalizada es importante elegir la jerarquía correcta, al menos para dejar claro si se trata de excepciones comprobadas o no comprobadas.
- Por tanto, para crear excepciones personalizadas comprobadas crearemos una clase que herede de `Exception` o de cualquier subclase que no sea `RuntimeException`.
- Para excepciones personlizadas no comprobadas heredaremos de `RuntimeException` o alguna de sus subclases.

Ejemplo de excepción personalizada

- Al crear una excepción personalizada, como mínimo deberemos sobreescribir su constructor que recibe como parámetro un String que es el mensaje de la excepción.
- Por ejemplo para definir una excepción personalizada para indicar que el NIF no es válido podríamos hacer:

```
public class InvalidNIFException extends RuntimeException {
    public InvalidNIFException(String msg) {
        super(msg);
    }
}
```

Ejemplo de excepción personalizada

```
public String solicitarNIF() throws InvalidNIFException {
    Scanner reader = new Scanner(System.in);
    System.out.println("NIF: ");
    String nif = reader.nextLine();
    if (validarNIF(nif)) {
        return nif;
    }
    throw new InvalidNIFException(nif + " no es un NIF válido");
}

public char letraNIF(int dni) {
    String tabla="TRWAGMYFPDXBNJZSQVHLCKE";
    int modulo= dni % 23;
    return tabla.charAt(modulo);
}

public boolean validarNIF(String nif) {
    if (nif == null)
        return false;
    nif = nif.toUpperCase();
    StringBuilder dniString = new StringBuilder();
    if (nif.length() >= 2) {
        char letra = nif.charAt(nif.length() - 1);
        char c;
        for (int i = 0; i < nif.length(); i++) {
            c = nif.charAt(i);
            if (Character.isDigit(c)) {
                dniString.append(c);
            }
        }
        return (!dniString.isEmpty()) && letra == letraNIF(Integer.parseInt(dniString.toString()));
    }
    return false;
}
```