

Excepciones



Curso 2024/25

Java

Manejo de excepciones

Introducción

Una excepción es un **evento** que puede ocurrir durante la ejecución de un programa y que **interrumpe** la ejecución normal del mismo.

Cuando se genera una excepción dentro de un método, el método **crea** un objeto y lo lanza. Dicho objeto se denomina `exception object` y contiene información sobre el error, incluyendo el **tipo** de error y el **estado** del programa cuando ocurrió el error.

Las excepciones se pueden **capturar** para tratarlas de forma adecuada y dejar a la aplicación en un estado **estable**.

Introducción

Existen tres tipos fundamentales de errores:

- **Errores:** Indican un error **irrecuperable**. Lo normal es que el programa se interrumpa.
- **Excepciones no comprobables:** Se generan en **tiempo de ejecución**. Por ejemplo, invocar a un método usando una variable a `null`.
- **Excepciones comprobables:** Constituyen **fallos menores**, como leer de un fichero que no existe.

Introducción

Existen tres tipos fundamentales de errores:

- **Separar el manejo de errores del código normal:** el código susceptible de generar excepciones se **separa** del resto de código. Fácilmente localizable.
- **Propagar los errores sobre la pila de llamadas:** Si un método lanza una excepción, o bien la **captura** o bien la **relanza** sobre la pila de llamadas.
- **Agrupar los errores y diferenciarlos:** Permite tratar de forma similar errores **similares**.

Introducción

Una excepción se puede producir por varios motivos:

- Porque un método reciba datos de entrada que **no sabe** procesar.
- Porque se produzca una situación excepcional, como por ejemplo invocar a un método con un objeto a null: `NullPointerException`.
- Porque se produzca un **error de entrada/salida**. Por ejemplo intentar leer un fichero que no existe: `IOException`.

Introducción

Cuando un método lanza una excepción, puede:

- **Tratarla** de forma que la aplicación siga estable.
- **Lanzarla** para que otro método (siguiendo la pila de llamadas) la trate.

Lanzamiento

Para lanzar una excepción se utiliza:

```
throw new NombreException(...)
```


Lanzamiento

Para lanzar una excepción se utiliza:

```
throw new NombreException(...)
```

Si un método lanza una excepción, su cabecera **debe** indicarlo.

```
visibilidad tipo nombreMetodo throws NombreException {...}
```

Lanzamiento

```
package arithmetics;
import excepciones.ExceptionDivisionByZero;

public class Arithmetic {
    private int a;
    private int b;
    public Arithmetic(int a, int b){
        this.a = a;
        this.b = b;
    }
    public int division() throws ExceptionDivisionByZero {
        if (this.b == 0)
            throw new ExceptionDivisionByZero();
        else
            return this.a / this.b;
    }
}
```

```
public class ExceptionDivisionByZero extends Throwable {
    public ExceptionDivisionByZero() {
        super();
    }
}
```

Lanzamiento

Cuando se lanza una excepción con `throw`, lo que se lanza realmente es un **objeto** de una clase que representa una excepción.

La clase correspondiente a la excepción debe de **existir** y puede contener distintas constructoras y/o métodos.

Un método puede lanzar **distintos** tipos de excepciones.

Cuando la excepción se lanza, esta **sube** por la pila de llamadas hasta que es **capturada** o bien **interrumpe** abruptamente la aplicación.

Lanzamiento

```
public static void main(String[] args) throws ExceptionDivisionByZero {  
    Arithmetic a = new Arithmetic(2,0);  
    int z = a.division(); // Lanza excepción "División entre 0"  
    System.out.println(z);  
}
```

Lanzamiento

```
public static void main(String[] args) throws ExceptionDivisionByZero {  
    Arithmetic a = new Arithmetic(2,0);  
    int z = a.division(); // Lanza excepción "División entre 0"  
    System.out.println(z);  
}
```



Exception in thread "main" excepciones.ExceptionDivisionByZero
at excepciones.Arithmetic.division(Arithmetic.java:9)
at excepciones.Main.main(Main.java:8)

Lanzamiento

```
public int division() throws ExceptionDivisionByZero, ExceptionNegativeNumber {  
    if (this.a < 0 || this.b < 0)  
        throw new ExceptionNegativeNumber("Error: Negative number");  
    else if (this.b == 0)  
        throw new ExceptionDivisionByZero("Error: Division By Zero");  
    else return  
        this.a / this.b;  
}
```

Lanzamiento

```
public class ExceptionDivisionByZero extends Throwable {  
    public ExceptionDivisionByZero() {  
        super();  
    }  
    public ExceptionDivisionByZero(String message) {  
        super(message);  
    }  
}
```

```
public class ExceptionNegativeNumber extends Throwable {  
    public ExceptionNegativeNumber(String message) {  
        super(message);  
    }  
}
```

Lanzamiento

```
public class Main {  
    public static void main(String[] args) throws ExceptionDivisionByZero, ExceptionNegativeNumber {  
        Arithmetic a = new Arithmetic(-2,4);  
        int z = a.division();  
        System.out.println(z);  
    }  
}
```


Lanzamiento

```
public class Main {  
    public static void main(String[] args) throws ExceptionDivisionByZero, ExceptionNegativeNumber {  
        Arithmetic a = new Arithmetic(-2,4);  
        int z = a.division();  
        System.out.println(z);  
    }  
}
```



Exception in thread "main" excepciones.ExceptionNegativeNumber: Error: Negative number
at excepciones.Arithmetic.positiveDivision(Arithmetic.java:16)
at excepciones.Main.main(Main.java:8)

Lanzamiento

```
public class Main {  
    public static void main(String[] args) throws ExceptionDivisionByZero, ExceptionNegativeNumber {  
        Arithmetic a = new Arithmetic(2,0);  
        int z = a.division();  
        System.out.println(z);  
    }  
}
```

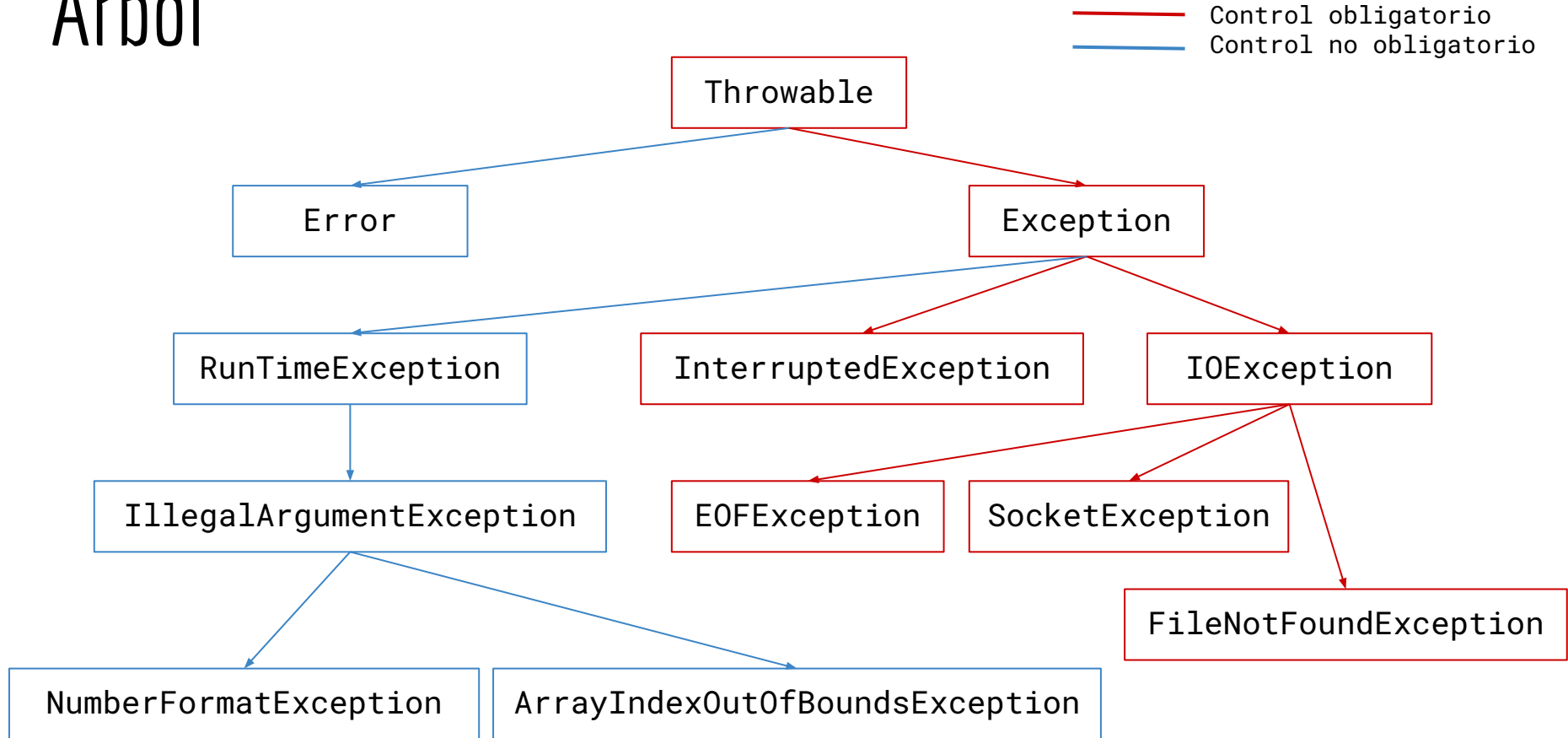
Lanzamiento

```
public class Main {  
    public static void main(String[] args) throws ExceptionDivisionByZero, ExceptionNegativeNumber {  
        Arithmetic a = new Arithmetic(2,0);  
        int z = a.division();  
        System.out.println(z);  
    }  
}
```



Exception in thread "main" excepciones.ExceptionDivisionByZero: Error: Division By Zero
at excepciones.Arithmetic.division(Arithmetic.java:9)
at excepciones.Main.main(Main.java:8)

Árbol



Definición

Cuando un método es susceptible de lanzar excepción, **siempre que sea posible** conviene darle un tratamiento adecuado. Para ello se usa `try / catch`.

Si un método tiene un bloque `try / catch` capaz de capturar todas las posibles excepciones que genere su ejecución, entonces el método ya **no lanza** excepción.

Definición

Cuando un método es susceptible de lanzar excepción, **siempre que sea posible** conviene darle un tratamiento adecuado. Para ello se usa `try / catch`.

Si un método tiene un bloque `try / catch` capaz de capturar todas las posibles excepciones que genere su ejecución, entonces el método ya **no lanza** excepción.

```
try {  
    // Código susceptible de lanzar excepción  
}  
catch (ClaseExcepcion_1) { // Tratamiento de la excepción }  
catch (ClaseExcepcion_2) { // Tratamiento de la excepción }  
...  
[finally { // A ejecutar en cualquier caso }]
```

Funcionamiento

Consideramos la clase `Arithmetic`. Además, asumimos que las clases `ExceptionNegativeNumber` y `ExceptionDivisionByZero` extienden de la clase `Error`.

```
public static void main(String[] args){  
    try {  
        Arithmetic a = new Arithmetic(-2,3);  
        int z = a.division();  
        System.out.println(z);  
    }  
    catch (ExceptionDivisionByZero e){  
        System.out.println(e.getMessage());  
    }  
    catch (ExceptionNegativeNumber e){  
        System.out.println(e.getMessage());  
    }  
    System.out.println("La ejecución sigue a partir de aquí");  
}
```

Funcionamiento

Consideramos la clase `Arithmetic`. Además, asumimos que las clases `ExceptionNegativeNumber` y `ExceptionDivisionByZero` extienden de la clase `Error`.

```
public static void main(String[] args){  
    try {  
        Arithmetic a = new Arithmetic(-2,3);  
        int z = a.division();  
        System.out.println(z);  
    }  
    catch (ExceptionDivisionByZero e){  
        System.out.println(e.getMessage());  
    }  
    catch (ExceptionNegativeNumber e){  
        System.out.println(e.getMessage());  
    }  
    System.out.println("La ejecución sigue a partir de aquí");  
}
```

¿Qué muestra
por pantalla?

Funcionamiento

Debemos considerar **todas** las excepciones que un código puede lanzar.

```
public static void main(String[] args){  
    try { Arithmetic a = new Arithmetic(-2,3);  
        int z = a.division();  
        System.out.println(z);  
    }  
    catch (ExceptionDivisionByZero e){  
        System.out.println(e.getMessage());  
    }  
    System.out.println("La ejecución sigue a partir de aquí");  
}
```

Funcionamiento

Debemos considerar **todas** las excepciones que un código puede lanzar.

```
public static void main(String[] args){  
    try { Arithmetic a = new Arithmetic(-2,3);  
        int z = a.division();  
        System.out.println(z);  
    }  
    catch (ExceptionDivisionByZero e){  
        System.out.println(e.getMessage());  
    }  
    System.out.println("La ejecución sigue a partir de aquí");  
}
```

¿Y ahora?

Funcionamiento

Debemos considerar **todas** las excepciones que un código puede lanzar.

```
public static void main(String[] args){
    Arithmetic a = new Arithmetic(-2,3);
    try {
        int z = a.division();
        System.out.println(z);
    }
    catch (ExceptionDivisionByZero e){
        System.out.println(e.getMessage());
    }
    catch (ExceptionNegativeNumber e){
        System.out.println(e.getMessage());
    }
    finally {
        System.out.println("En último lugar entro aquí");
    }
    System.out.println("La ejecución sigue a partir de aquí");
}
```

Funcionamiento

Debemos considerar **todas** las excepciones que un código puede lanzar.

```
public static void main(String[] args){
    Arithmetic a = new Arithmetic(-2,3);
    try {
        int z = a.division();
        System.out.println(z);
    }
    catch (ExceptionDivisionByZero e){
        System.out.println(e.getMessage());
    }
    catch (ExceptionNegativeNumber e){
        System.out.println(e.getMessage());
    }
    finally {
        System.out.println("En último lugar entro aquí");
    }
    System.out.println("La ejecución sigue a partir de aquí");
}
```

¿Y ahora...?

Aclaraciones

Un bloque `try / catch` puede contener tantos `catch` como se desee.

Los `catch`, en caso de que exista una jerarquía de excepciones, deben ir **ordenados** de forma correcta, esto es, desde **debajo** de la jerarquía **hacia** arriba.

Si no están relacionadas, entonces el orden es **irrelevante**.

Como norma siempre indicaremos en la cabecera de un método (cuando sea posible) el tipo de la excepción concreta que lanza. Intentaremos **evitar** poner super clases.

Da igual que la excepción se capture o no se capture; si se ejecuta un bloque `try`, entonces su `finally` correspondiente **siempre** se ejecuta.

Fuentes adicionales

Para reforzar los conceptos vistos hasta ahora, vamos a echar un vistazo a los siguientes blogs y volver a repasar algunos (y otros) ejemplos útiles:

<https://www.programiz.com/java-programming/exception-handling>

https://www.w3schools.com/java/java_try_catch.asp

Java

Ejercicios de excepciones

Ejercicio I

Crea un pequeño proyecto que contenga la clase `AgeValidation` y `InvalidAgeException`. La segunda clase, que ya te la doy, lo único que hace es, como puedes ver, tener una constructora que crea un objeto (que extiende de `Exception`) con el mensaje que le pasemos por parámetro. Por otro lado, la primera clase debe hacer lo siguiente:

- Tendrá el método `main()` y el método `checkAge()`. En el segundo método, recibe por parámetro la edad y, si esta es menor de 18 o mayor de 100, entonces crea y lanza una excepción del tipo `InvalidAgeException` con el mensaje **La edad debe estar entre 18 y 100 años**.
- Al mismo tiempo, en el primer método debes crear un scanner, donde preguntarás al usuario que introduzca su edad. Entonces, llamas al método `checkAge()` para comprobar la edad (el cual, recuerda, puede lanzar una excepción). Si la edad es correcta, entonces dirá **La edad es válida**. Y si no, pues muestra el mensaje mencionado anteriormente. En cualquiera de los casos, se debe cerrar el scanner con el método `close()`.

```
class InvalidAgeException extends Exception {  
    public InvalidAgeException(String message) {  
        super(message);  
    }  
}
```


Ejercicio II

Comenta línea a línea lo que
hace el siguiente código.

```
public class FileReadingExample {
    public static void main(String[] args) {
        BufferedReader reader = null;
        try {
            System.out.print("Enter the file name: ");
            BufferedReader inputReader = new BufferedReader(new InputStreamReader(System.in));
            String fileName = inputReader.readLine();

            reader = new BufferedReader(new FileReader(fileName));
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("An I/O error occurred: " + e.getMessage());
        } finally {
            try {
                if (reader != null) {
                    reader.close();
                }
            } catch (IOException e) {
                System.out.println("Error closing the file: " + e.getMessage());
            }
        }
    }
}
```

Ejercicio III

Crea un pequeño proyecto que maneje una matriz 4x4 de enteros `int[][]`. Debes pedir al usuario que inserte un nuevo valor en la columna i-ésima y fila j-ésima (es decir, también debes pedirle la posición donde desea insertar el nuevo valor). Hecho esto, el programa actualizará el valor de la matriz en dicha posición. Si se introduce una posición fuera de límites, entonces se creará una excepción con un mensaje personalizado que deberás capturar y mostrar por pantalla. De cualquier otro modo, se mostrará un mensaje **Posición actualizada** y se cerrará el scanner. Por último, debes imprimir nuevamente la matriz.

Excepciones



Curso 2024/25