

Resumen Guia 6

Manejo de Excepciones




Excepciones vs. error Java

Una **excepción** es un evento que ocurre durante la ejecución del programa que interrumpe el flujo normal de las sentencias.

Para evitar que esto ocurra, podemos manejar excepciones en nuestro programa y continuar con la ejecución del programa normalmente.

Un **error** es un evento irrecuperable que no se puede manejar. El programa corta abruptamente su ejecución.



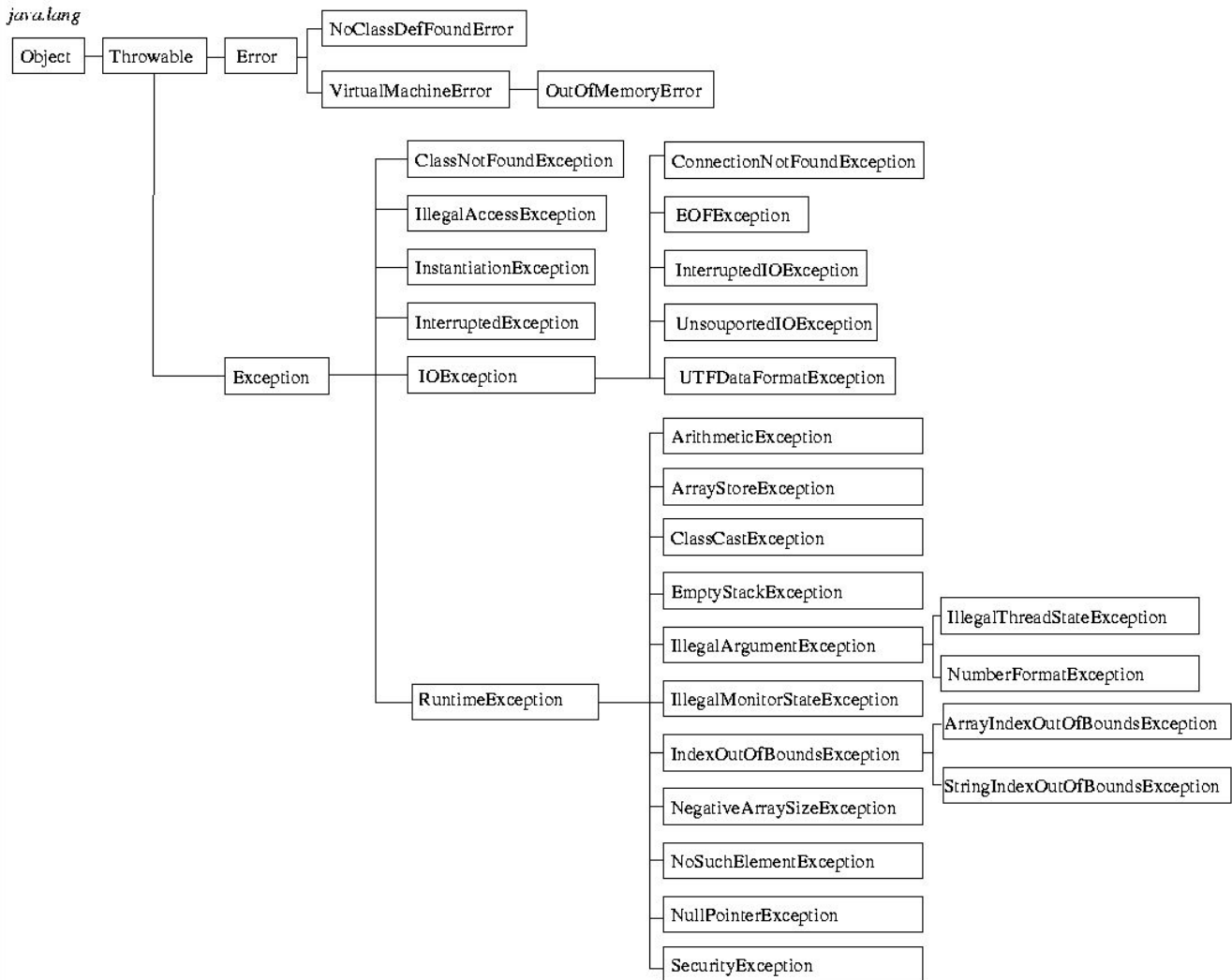
¿Cómo trabaja el manejador de excepciones?

El proceso de especificar la secuencia de pasos en un programa para manejar la excepción se llama **Manejo de excepciones**. Al proporcionar controladores de excepciones en un programa, **podemos garantizar el flujo normal de ejecución del programa**.

Ventajas:

- Separar el Manejo de Errores del Código "Normal".
- Propagar los errores sobre la Pila de Llamadas.
- Agrupar los Tipos de Errores y la Diferenciación de éstos.

java.lang





¿Qué ocurre cuando se produce una excepción?

- **Se crea un objeto exception y se lanza.** El objeto exception creado contiene información sobre el error. La ejecución normal del programa se detiene.
- Se busca en el método donde se ha producido la excepción un manejador de excepciones que capture ese objeto y que **gestione la excepción**.
- **Si este método no incluye un manejador de excepciones apropiado se busca en el método que llamó a este y así sucesivamente.** Se considera apropiado si el tipo de objeto excepción lanzado es compatible al tipo que puede manejar.
- Si no se encuentra un manejador de excepciones apropiado, Java muestra el error y acaba el programa.

Ejemplo sin manejador de excepciones

```
leerFichero {  
  abrir el fichero;  
  determinar su tamaño;  
  asignar suficiente memoria;  
  leer el fichero a la memoria;  
  cerrar el fichero;  
}
```

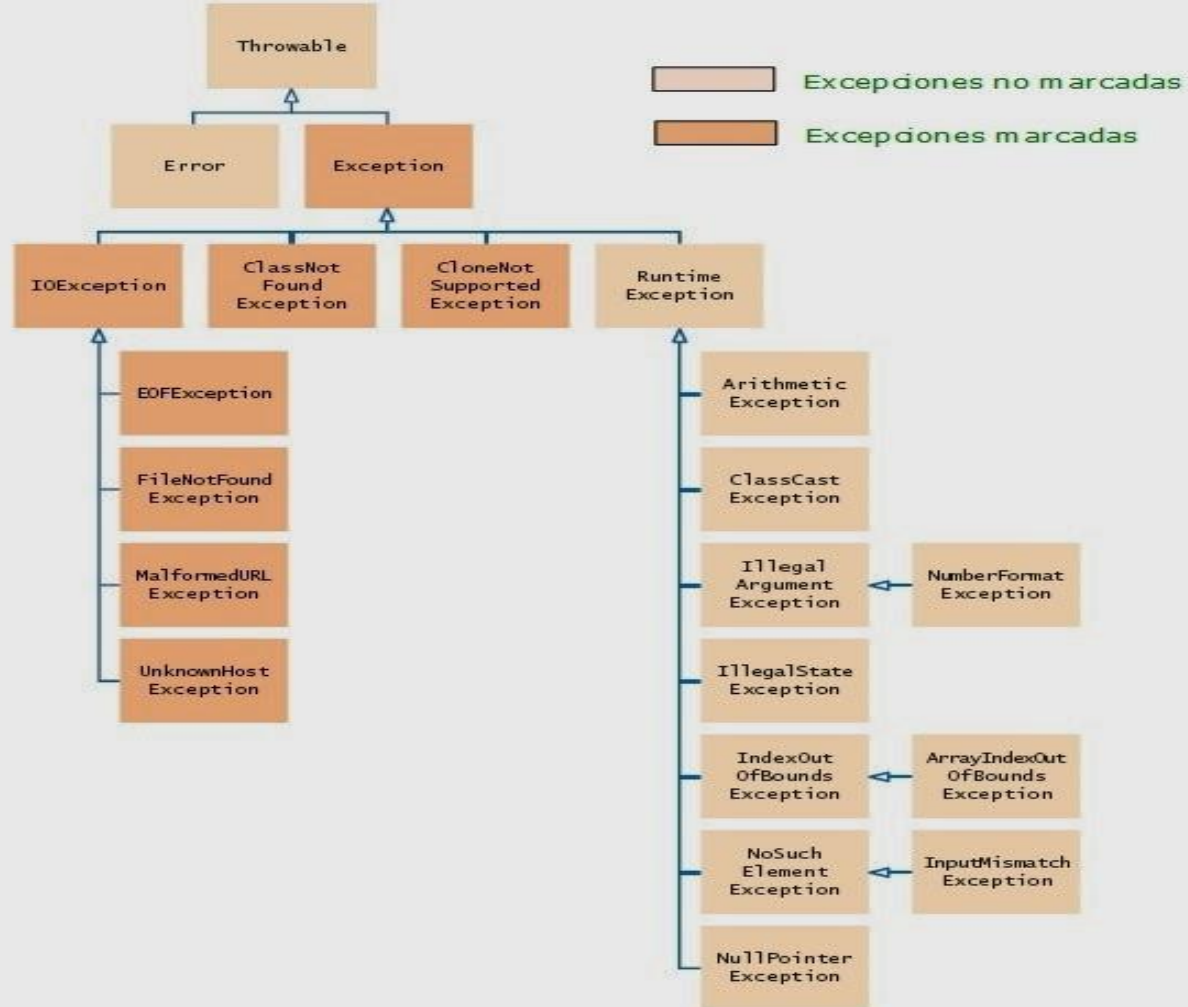
```
codigodeError leerFichero {  
  inicializar codigodeError = 0;  
  abrir el fichero;  
  if (ficheroAbierto) {  
    determinar la longitud del fichero;  
    if (obtenerLongitudDelFichero) {  
      asignar suficiente memoria;  
      if (obtenerSuficienteMemoria) {  
        leer el fichero a memoria;  
        if (falloDeLectura) {  
          codigodeError = -1;  
        }  
      } else {  
        codigodeError = -2;  
      }  
    } else {  
      codigodeError = -3;  
    }  
  }  
  cerrar el fichero;  
  if (ficheroNoCerrado && codigodeError == 0) {  
    codigodeError = -4;  
  } else {  
    codigodeError = codigodeError and -4;  
  }  
} else {  
  codigodeError = -5;  
}  
return codigodeError;  
}
```

Ejemplo con manejador de excepciones

```
leerFichero {  
    try {  
        abrir el fichero;  
        determinar su tamaño;  
        asignar suficiente memoria;  
        leer el fichero a la memoria;  
        cerrar el fichero;  
    } catch (falloAbrirFichero) {  
        hacerAlgo;  
    } catch (falloDeterminacionTamaño) {  
        hacerAlgo;  
    } catch (falloAsignaciondeMemoria) {  
        hacerAlgo;  
    } catch (falloLectura) {  
        hacerAlgo;  
    } catch (falloCerrarFichero) {  
        hacerAlgo;  
    }  
}
```

El manejo de excepciones no evita el esfuerzo de hacer el trabajo de detectar, informar y manejar errores. **Lo que proporciona el manejo de excepciones es una solución elegante al problema del tratamiento de errores.**

Las excepciones permiten escribir el flujo principal del código y tratar los casos excepcionales en otro lugar.





Tipos de Excepciones java

1. **Excepción marcada:** Durante la **compilación** del código, se comprueba la excepción marcada y **es obligatorio lanzar o gestionar la excepción**. Esta excepción se utiliza para separar el código de manejo de errores del código normal. Todas las excepciones marcadas están agrupadas y es útil para diferenciar los problemas.
2. **Excepción sin marcar:** Las excepciones sin marcar son las excepciones que se comprueban **en tiempo de ejecución**, **pueden evitarse mediante la programación adecuada**. La compilación de programas tendrá éxito.
3. **Error:** Error describe una situación que **no se puede manejar** y da como resultado que un programa se bloquee.

Bloque try - catch - finally

```
try {  
    // bloque de código  
}  
catch (Excepción1 e1) {  
    // tratamiento de la excepción Excepción1  
}  
catch (Excepción2 e2) {  
    // tratamiento de la excepción Excepción2  
}  
finally {  
    // bloque de código que siempre se ejecutará  
}
```

```
2  
3 public class ExcepcionesMain {  
4  
5     public static void main(String[] args) {  
6  
7         String nombre = null;  
8  
9         try {  
10             System.out.println("Nombre = " + nombre);  
11  
12         } catch (NullPointerException ex) {  
13             System.out.println("Error al imprimir la longitud de la cadena: " + ex.getMessage());  
14  
15         } finally {  
16             System.out.println("Esta línea siempre se va a imprimir");  
17         }  
18     }  
19 }
```

Bloque finally

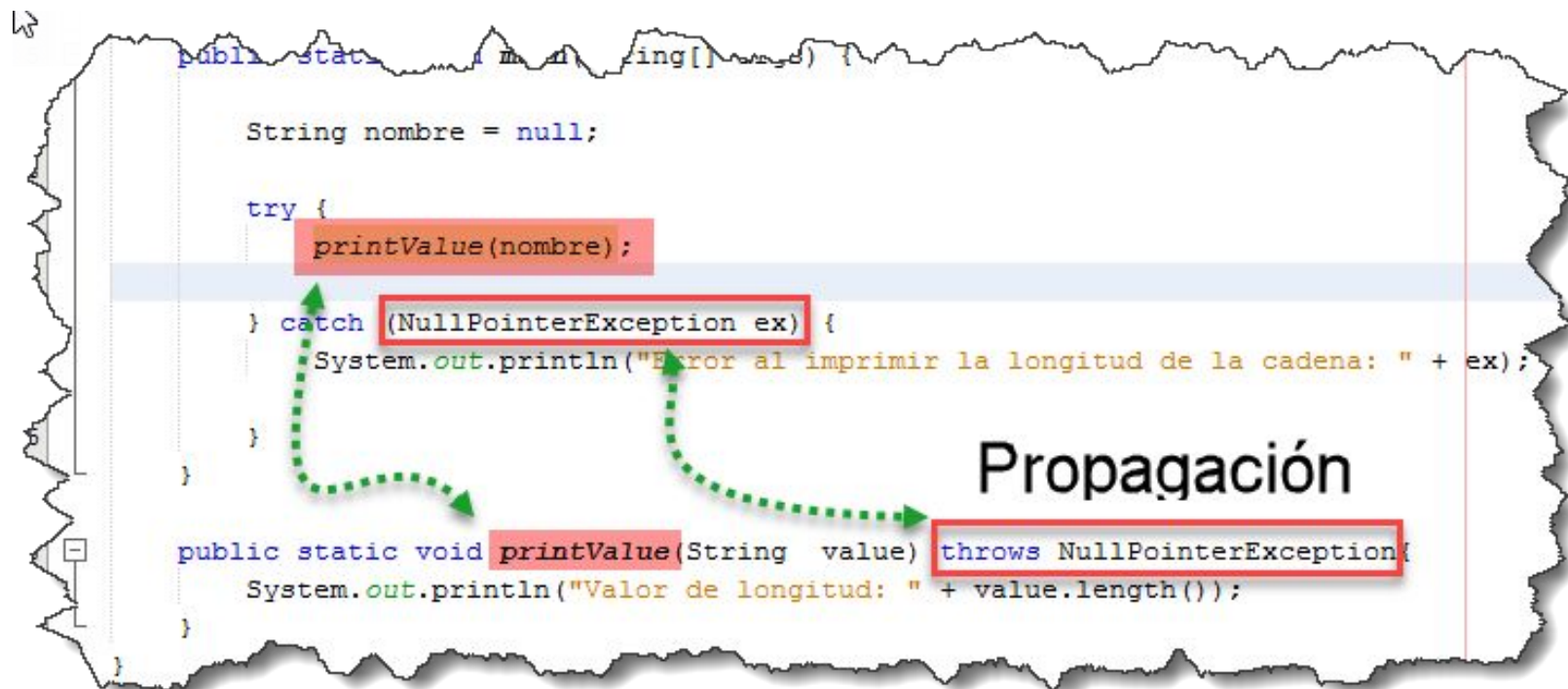
cursojava:excepciones.ExcepcionesMain > main > try > catch NullPointerException ex >

Output - HolaMundoProy (run) x

```
run:  
Error al imprimir la longitud de la cadena: java.lang.NullPointerException  
Esta línea siempre se va a imprimir  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Siempre se imprime

Propagación de excepciones - **throws**



Es necesario usarla con todas las excepciones marcadas.

Creación de excepciones

```
2  
3 public class MiExcepcion extends Exception {  
4  
5     public MiExcepcion() {  
6     }  
7  
8     public MiExcepcion(String mensaje) {  
9         super(mensaje);  
10    }  
11  
12    public MiExcepcion(String mensaje, Throwable causa) {  
13        super(mensaje, causa);  
14    }  
15  
16 }
```

The diagram illustrates the inheritance of the `super()` calls in the `MiExcepcion` class. It shows three constructors: a no-argument constructor, a constructor taking a `String` message, and a constructor taking a `String` message and a `Throwable` cause. The `super(mensaje)` and `super(mensaje, causa)` calls are highlighted in red. Green dotted arrows indicate the inheritance path from these calls to the `super` field in the `Exception` superclass, which is highlighted in a red box next to the `extends Exception` declaration.

Lanzar una excepción - **throw**

```
2
3 public class ExcepcionesMain {
4
5     public static void main(String[] args) {
6
7         String nombre = null;
8
9         try {
10
11             if(nombre == null){
12                 throw new MiExcepcion("Mi excepcion personalizada");
13             }
14
15         } catch (MiExcepcion ex) {
16             System.out.println("Error al imprimir la longitud de la cadena: " + ex);
17         }
18     }
19 }
20
```

Lanza excepción
personalizada

Cacha la excepción
personalizada