
PROMETEO
PROGRAMACIÓN
Control y manejo de Excepciones

Índice

01

Concepto

El manejo de excepciones es una característica fundamental de Java que permite gestionar errores y condiciones anómalas de forma controlada. Esto mejora la robustez y estabilidad de los programas al evitar fallos inesperados durante la ejecución.

1. Excepciones: Concepto

2. **¿Qué es una Excepción?**
   - Una **excepción** es un evento que interrumpe el flujo normal de ejecución de un programa debido a un error o condición inesperada.
   - Se genera cuando el programa encuentra un problema, como dividir por cero, acceder a índices fuera de rango o intentar abrir un archivo inexistente.
3. **Características Principales**:
   - Las excepciones son **objetos** que representan un error o un estado excepcional.
   - Java proporciona un mecanismo estructurado para **capturar, manejar y recuperar** de estos errores.
4. **Manejo de Excepciones en Java**:
   - Java utiliza las palabras clave **try**, **catch**, **finally**, y **throw** para manejar excepciones.

02

Flujo de control

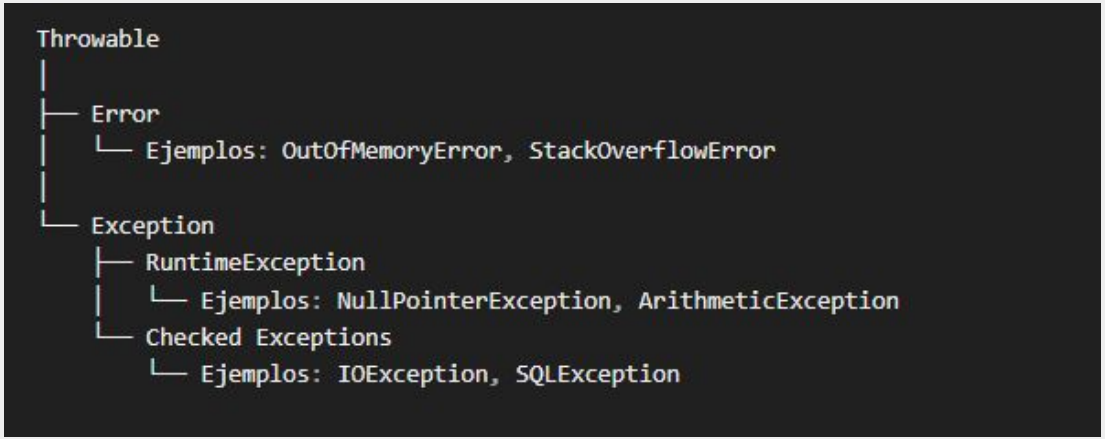
Bloque	Descripción
`try`	Contiene el código que puede generar una excepción.
`catch`	Captura y maneja la excepción generada en el bloque `try`.
`finally`	Contiene código que se ejecuta siempre, independientemente de si hubo excepción.
`throw`	Lanza manualmente una excepción en el programa.

03

Jerarquías

Java organiza las excepciones en una jerarquía basada en clases. Todas las excepciones derivan de la clase base Throwable.

Diagrama Simplificado de la Jerarquía de Excepciones:



03

Jerarquías

Tipos de Excepciones

1. **Errores (`Error`)**:
   - Representan problemas graves relacionados con el entorno del programa.
   - No deben manejarse directamente en la mayoría de los casos.
   - Ejemplo: **`OutOfMemoryError`**.

2. **Excepciones (`Exception`)**:
   - Representan problemas que pueden ser manejados y recuperados.
   - Se dividen en:
     1. **Excepciones Verificadas (`Checked Exceptions`)**:
        1. Son verificadas en tiempo de compilación.
        2. Ejemplo: **`IOException`**.
     2. **Excepciones No Verificadas (`Unchecked Exceptions`)**:
        1. Son errores en tiempo de ejecución.
        2. Ejemplo: **`NullPointerException`**, **`ArithmeticException`**.

03

Jerarquías

Captura de Excepciones Múltiples

Es posible capturar diferentes tipos de excepciones utilizando varios bloques catch.

```java
public class Main {
    public static void main(String[] args) {
        try {
            int[] numeros = {1, 2, 3};
            System.out.println(numeros[5]); // Genera ArrayIndexOutOfBoundsException
        } catch (ArithmeticException e) {
            System.out.println("Error aritmético.");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Índice fuera de rango.");
        } catch (Exception e) {
            System.out.println("Ocurrió un error.");
        }
    }
}
```

03

Jerarquías

Relación Entre Excepciones

Cuando se manejan excepciones, es importante capturarlas desde las más específicas a las más generales. Esto asegura que el bloque correcto maneje la excepción.

```
try {
    int resultado = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Error específico: División por cero.");
} catch (Exception e) {
    System.out.println("Error general.");
}
```

04

Excepciones personalizadas

Además de utilizar las excepciones estándar de Java, podemos crear excepciones personalizadas para manejar situaciones específicas en nuestras aplicaciones. Además, las palabras clave **throw** y **throws** permiten controlar explícitamente la generación y propagación de excepciones.

¿Qué son las Excepciones Personalizadas?

Las **excepciones personalizadas** son clases definidas por el usuario que extienden la clase base **Exception** o **RuntimeException**. Estas excepciones permiten modelar errores específicos de una aplicación, proporcionando más claridad y control sobre el manejo de errores.

04

Excepciones personalizadas

Creación de Excepciones Personalizadas

- **Extender `Exception`**:
  - Usar cuando se requiere que la excepción sea verificada (checked).
  - El compilador obliga a manejarla o declararla con throws.
- **Extender `RuntimeException`**:
  - Usar cuando se requiere una excepción no verificada (unchecked).
  - El compilador no obliga a manejarla.

04

Excepciones personalizadas

Ejemplo con Excepción Verificada

```
public class Main {
    public static void verificarEdad(int edad) throws MiExcepcion {
        if (edad < 18) {
            throw new MiExcepcion("La edad mínima es 18 años.");
        }
    }

    public static void main(String[] args) {
        try {
            verificarEdad(16);
        } catch (MiExcepcion e) {
            System.out.println("Excepción capturada: " + e.getMessage());
        }
    }
}
```

04

Excepciones personalizadas

Ejemplo con Excepción No Verificada

```
public class Main {
    public static void verificarSaldo(double saldo) {
        if (saldo < 0) {
            throw new MiExcepcionRuntime("El saldo no puede ser negativo.");
        }
    }

    public static void main(String[] args) {
        verificarSaldo(-50); // Lanza excepción sin necesidad de try-catch
    }
}
```

05

Uso de Throw y Throws

Palabra Clave throw

La palabra clave **throw** se utiliza para lanzar explícitamente una excepción en un programa.

Palabra Clave throws

La palabra clave **throws** se utiliza en la declaración de un método para indicar que puede lanzar una o más excepciones verificadas.

Aspecto	`throw`	`throws`
Uso	Se utiliza para lanzar una excepción.	Se utiliza para declarar excepciones que puede lanzar un método.
Momento	Se encuentra dentro del cuerpo del método.	Se encuentra en la declaración del método.
Tipos	Lanza una única instancia de excepción.	Declara una o más excepciones.


PROMETEO