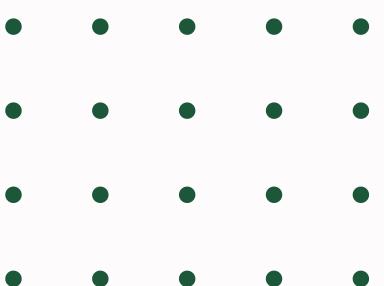


JAVA Exceptions

Arnau Burguera Calles
IT Academy

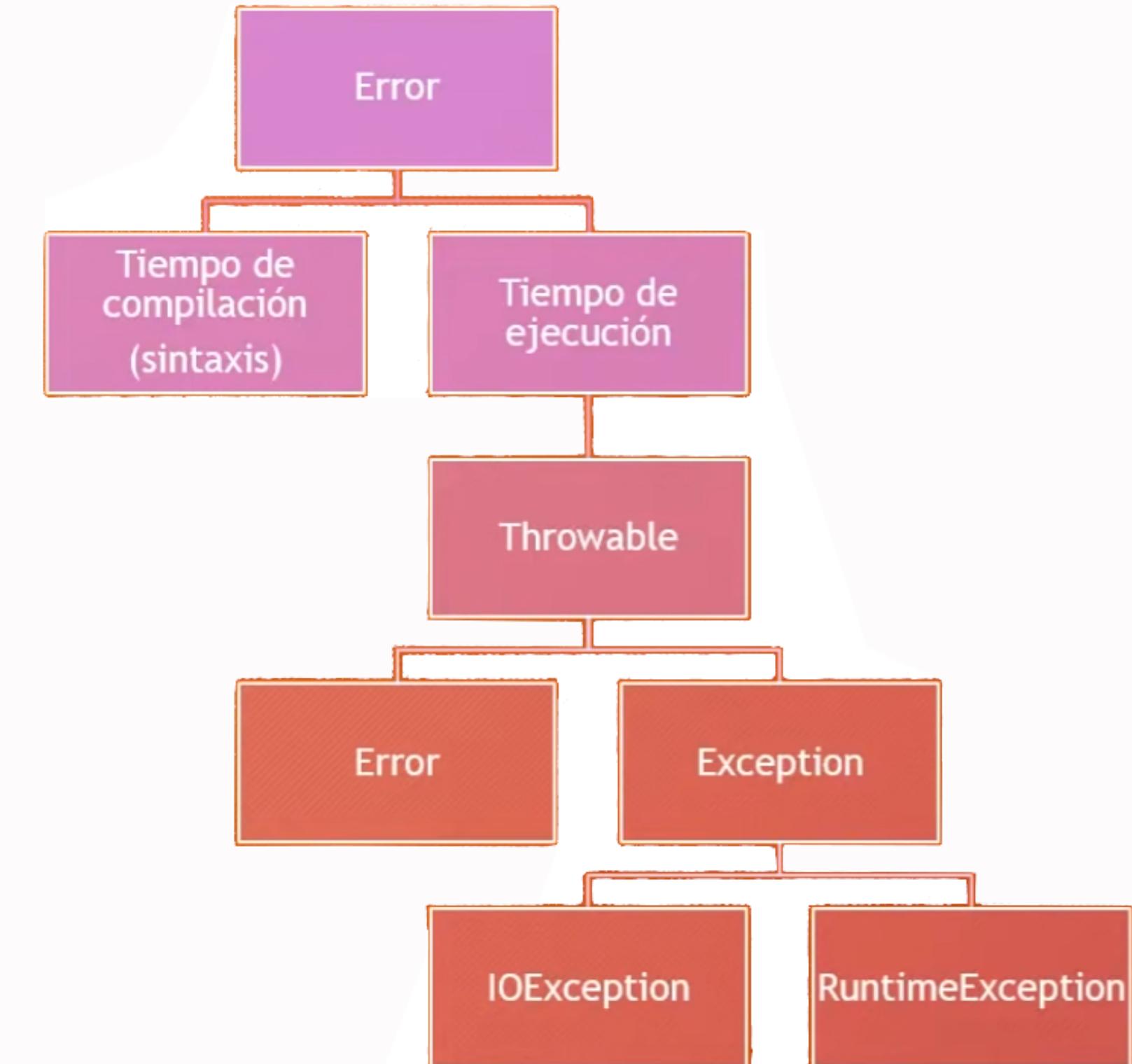


Errores

Errores de sintaxis (Syntax Errors): Estos errores ocurren cuando el código no sigue las reglas de sintaxis del lenguaje Java

Errores (Errors): Los errores en Java representan situaciones graves que normalmente están fuera del control del programador y que afectan al entorno de ejecución de la aplicación.

Excepciones (Exceptions): Las excepciones representan situaciones excepcionales o errores que pueden ocurrir durante la ejecución de un programa.



Qué son?

- Son situaciones excepcionales o errores que pueden ocurrir durante la ejecución de un programa.
- Nos permiten manejar errores y condiciones inesperadas de manera controlada, lo que ayuda a que los programas sean más robustos.
- Son objetos de clases que heredan de la clase base `java.lang.Throwable`.



Tipos

Checked Exceptions (Excepciones revisadas) //La culpa no es del programador

Son excepciones que el programador debe manejar explícitamente en su código. Estas excepciones derivan de la clase `java.lang.Exception`, pero no de `RuntimeException`. El manejo de estas excepciones generalmente se hace usando un bloque `try-catch` o propagándolas con ‘throws’.

Ejemplos: `IOException`, `SQLException` y `FileNotFoundException`.

Unchecked Exceptions (Excepciones no revisadas) //La culpa es del programador

No requieren un manejo explícito por parte del programador, pero es una buena práctica tratar de evitar estas excepciones escribiendo un código cuidadoso. Derivan de la clase `java.lang.RuntimeException`.

Ejemplos: `NullPointerException` , `ArrayIndexOutOfBoundsException` y `ArithmetricException`.

Checked Exceptions

(Revisadas)



Checked (Revisadas)

- Deben manejarse explícitamente en el código o declararse en la firma del método.
- Estas excepciones son subclases de `java.lang.Exception` y se utilizan para representar situaciones en las que un programa puede anticipar y manejar un error.

//En el código (bloque try-catch)

```
try {
    // Código que podría lanzar una Checked Exception
} catch (TipoDeExcepcion e) {
    // Manejar la excepción aquí
}
```

//En la firma del método (Se encarga quien llame al método)

```
void miMetodo() throws TipoDeExcepcion {
    // Código que podría lanzar una Checked Exception
}
```

Unchecked Exceptions

(No revisadas)

Unchecked (No revisadas)

- Estas excepciones son subclases de `java.lang.RuntimeException` y suelen representar errores de programación o condiciones impredecibles en tiempo de ejecución.
- Las excepciones no revisadas son signos de problemas en el código, como errores de lógica, acceso incorrecto a objetos o cálculos erróneos.

```
String texto = null;
int longitud = texto.length(); // Esto generará un NullPointerException

int[] arreglo = new int[3];
int valor = arreglo[4]; // Esto generará una ArrayIndexOutOfBoundsException

int resultado = 10 / 0; // Esto generará una ArithmeticException
```

- Mientras que las excepciones no revisadas no requieren manejo explícito, es importante escribir código sólido y seguro para evitar su aparición siempre que sea posible.

Bloque Try-catch

- Proporcionan un mecanismo para capturar y gestionar excepciones, lo que evita que el programa se detenga abruptamente debido a errores.

```
try {  
    FileInputStream archivo = new FileInputStream("archivo.txt");  
    // Código para leer el archivo  
} catch (IOException e) {  
    // Manejar la excepción, por ejemplo, mostrar un mensaje de error  
    System.err.println("Error de entrada/salida: " + e.getMessage());  
}
```

- Permiten manejar excepciones de manera adecuada, como mostrar mensajes de error o realizar acciones correctivas.

Bloque Try-catch

//Scope (alcance)

- El alcance (scope) de un bloque try-catch en Java está limitado al bloque de código que lo contiene. Esto significa que cualquier variable declarada dentro del bloque try o dentro de los bloques catch solo es visible y accesible dentro de ese bloque y sus subbloques.

```
//int numero = 10;

try {
    int numero = 10;
    System.out.println("Dentro del try: " + numero);

    // Generar una excepción
    int resultado = numero / 0;

} catch (ArithmeticException e) {
    System.out.println("Fuera del try-catch: " + numero);// Error de compilación

    System.out.println("Dentro del catch: " + e.getMessage());
}

System.out.println("Fuera del try-catch: " + numero);// Error de compilación
```

Bloque Try-catch

//más de un catch (varias excepciones)

- El código en el bloque catch que coincide con el tipo de excepción lanzada se ejecutará, y si ninguno de los catch coincide, se utilizará el bloque catch genérico al final. Esto permite manejar diferentes excepciones de manera específica.

```
try {  
    // Código que podría generar una excepción  
    int[] arreglo = new int[3];  
    int valor = arreglo[4]; // Esto generará una ArrayIndexOutOfBoundsException  
} catch (ArrayIndexOutOfBoundsException e) {  
    // Manejo de la excepción de índice fuera de rango  
    System.out.println("Se produjo una ArrayIndexOutOfBoundsException: " + e.getMessage());  
} catch (ArithmeticException e) {  
    // Manejo de la excepción de división por cero (nunca se ejecutará en este ejemplo)  
    System.out.println("Se produjo una ArithmeticException: " + e.getMessage());  
} catch (Exception e) {  
    // Manejo de excepciones genéricas (si ninguno de los otros catch captura la excepción)  
    System.out.println("Se produjo una excepción genérica: " + e.getMessage());  
}
```

Bloque Try-catch

//uso del pipe “|” para un solo catch)

- Se trata de un catch que maneja varias excepciones diferentes en un solo bloque. Esto puede ser útil si se desea realizar la misma acción de manejo de excepciones para varios tipos de excepciones.

```
try {  
    // Código que podría generar una excepción  
    int resultado = 10 / 0; // Esto generará una ArithmeticException  
} catch (ArithmeticException | NullPointerException e) {  
    // Manejo de excepciones comunes (ArithmeticException o NullPointerException)  
    System.out.println("Se produjo una excepción: " + e.getMessage());  
}
```

// (Windows) Alt Gr + 1

(Mac) Alt + Shift + L

Bloque Try-catch

//cláusula finally{}

- El bloque finally se ejecutará siempre, sin importar si se produce una excepción o no. Esto puede ser útil para liberar recursos o realizar acciones que deben ocurrir independientemente de si se generó una excepción o no.

```
try {  
    // Código que podría generar una excepción  
    int resultado = 10 / 0; // Esto generará una ArithmeticException  
} catch (ArithmeticException e) {  
    // Manejo de la excepción  
    System.out.println("Se produjo una ArithmeticException: " + e.getMessage());  
} finally {  
    // El código en este bloque se ejecutará sin importar si hay excepción o no  
    System.out.println("Este bloque se ejecuta siempre.");  
}
```

Bloque Try-catch

//try-with-resources

- Se introdujo en Java 7 y se utiliza para asegurarse de que los recursos, como flujos de entrada/salida (streams) o conexiones de bases de datos, se cierren adecuadamente al finalizar su uso, sin importar si se produce una excepción o no.

```
try (Recurso1 recurso1 = new Recurso1(); Recurso2 recurso2 = new Recurso2()) {  
    // Código que utiliza los recursos  
}  
catch (Excepcion e) {  
    // Manejo de excepciones  
}
```

- Recurso1 y Recurso2 son clases que implementan la interfaz AutoCloseable. Cuando se utiliza try-with-resources, Java abre y cierra estos recursos de manera automática, incluso si se lanza una excepción. Esto mejora la gestión de recursos y evita la necesidad de bloques finally para cerrar los recursos manualmente.

throws...

- La palabra clave throws se utiliza en la firma del método para indicar que un método podría lanzar una o varias excepciones de un tipo específico.
- Esto notifica a quienes utilicen el método que deben estar preparados para manejar esas excepciones o declararlas nuevamente en su propio código.

```
void miMetodo() throws TipoDeExcepcion1, TipoDeExcepcion2 {  
    // Código que podría lanzar TipoDeExcepcion1 o TipoDeExcepcion2  
}
```

- Proporciona información sobre las excepciones que pueden ocurrir, lo que facilita el manejo adecuado de errores.



Excepciones personalizadas

- Las excepciones personalizadas son excepciones creadas por el programador para representar situaciones específicas en una aplicación.
- A menudo se utilizan para encapsular información adicional sobre el error y proporcionar mensajes de error significativos.
- Proporcionan información detallada sobre el error, lo que facilita la depuración.
- Permite que el código que utiliza tu excepción identifique y maneje situaciones específicas de manera más precisa.
- Deben usarse con moderación y de manera apropiada para mejorar la calidad y la confiabilidad de tu código.



Excepciones personalizadas

//como se crean

- Para crear una excepción personalizada, debes extender una de las clases base de excepciones, como `Exception` o `RuntimeException`, según sea necesario.

```
public class MiExcepcionPersonalizada extends Exception {  
    public MiExcepcionPersonalizada(String mensaje) {  
        super(mensaje);  
    }  
}
```

```
public class MiClase {  
    no usages new *  
    public void metodo() throws MiExcepcionPersonalizada {  
        // Código que genera una situación especial  
        if /* alguna condición especial */ {  
            throw new MiExcepcionPersonalizada( mensaje: "Ocurrió una situación especial.");  
        }  
    }  
}
```

Tips y buenos hábitos

1. Utilizar Excepciones Revisadas cuando sea Apropiado
2. Ser Específico en el Manejo de Excepciones
3. Proporcionar Mensajes Informativos
4. Evitar Atrapar Excepciones de Manera Genérica
5. Utilizar Bloques "finally" para Liberar Recursos
6. No Ignorar Excepciones
7. Documentar el Manejo de Excepciones
8. Seguir las Convenciones de Nombres
9. Evitar Exceso de Bloques Try-Catch
10. Usar Excepciones para Situaciones Excepcionales

The background image shows a modern office space with a high ceiling featuring exposed pipes and ductwork. Large green plants are integrated throughout the room, hanging from the ceiling and growing in planters on the walls and desks. There are several wooden desks with black office chairs, and a large sofa area in the background. The overall atmosphere is bright and natural.

Resumen