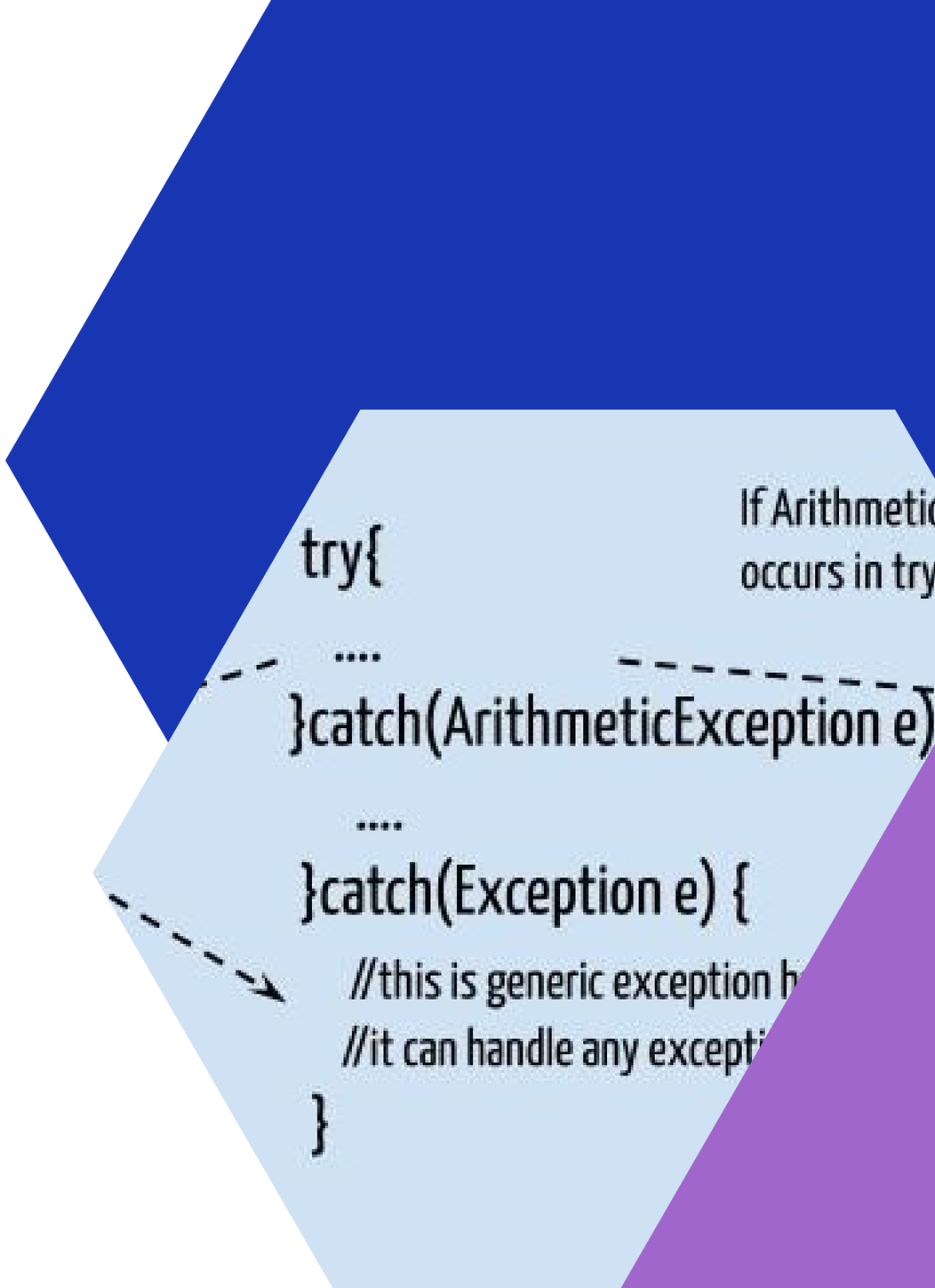
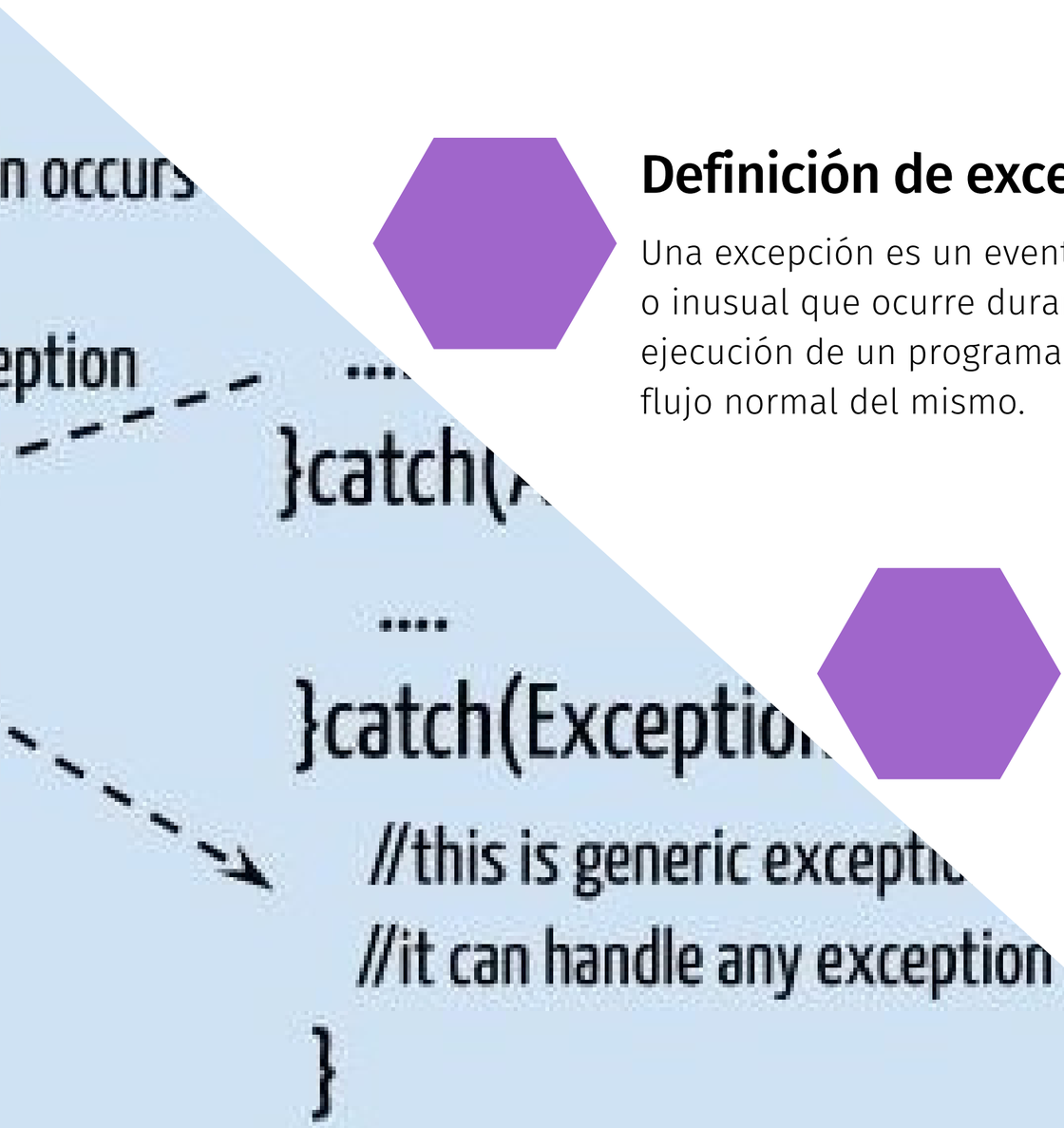
JAVA EXCEPCIONES





Definición de excepción

Una excepción es un evento imprevisto o inusual que ocurre durante la ejecución de un programa y perturba el flujo normal del mismo.

Importancia de su manejo

El manejo adecuado de excepciones ayuda a evitar que los programas se bloqueen o finalicen abruptamente en caso de errores, permitiendo una recuperación controlada.

Bloque "try-catch"

El bloque "try-catch" permite detectar y manejar las excepciones, proporcionando una forma de lidiar con situaciones excepcionales y continuar con la ejecución del programa.

Tipos de Excepciones en Java

Excepciones comprobadas (Checked Exceptions)

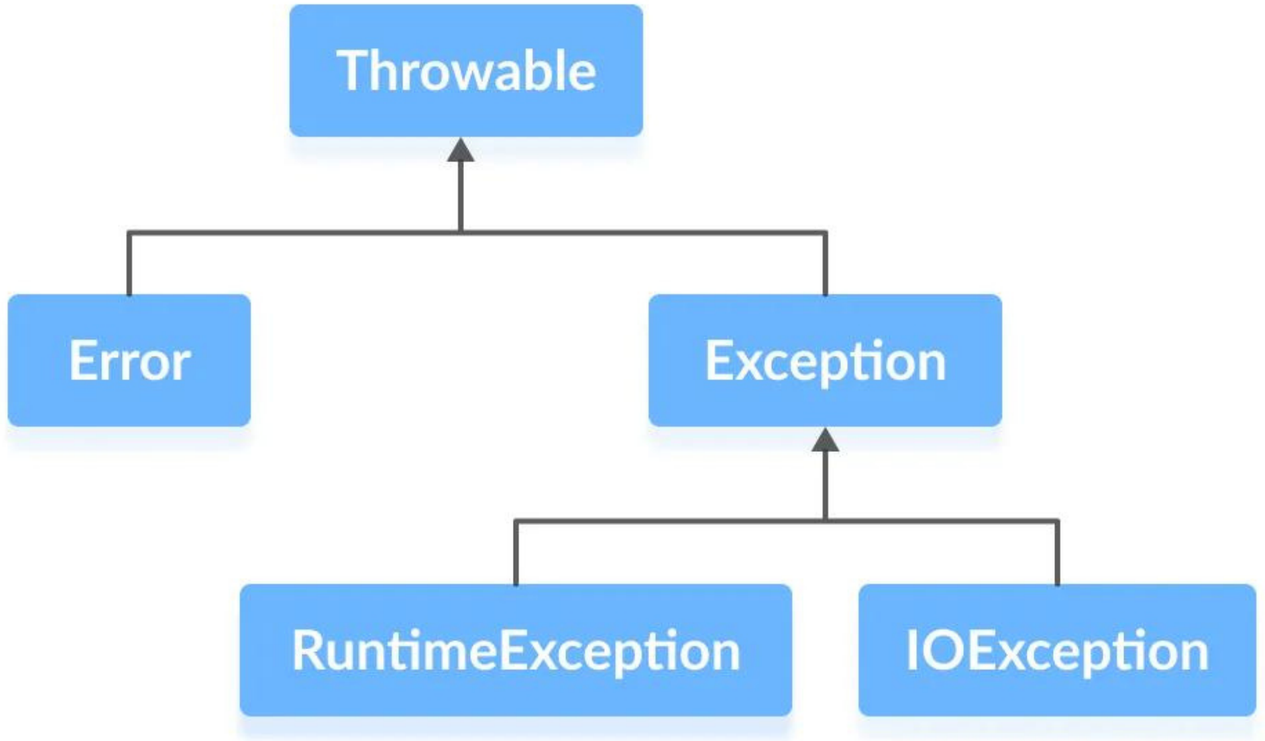
- ¿Qué son?: Son excepciones que el compilador obliga a manejar utilizando el bloque "try-catch" o declarando que el método las lanza.
- Ejemplos: FileNotFoundException (cuando un archivo no se encuentra), IOException (ocurren errores de entrada/salida).

Excepciones no comprobadas (Unchecked Exceptions)

- ¿Qué son?: Son excepciones que no requieren ser capturadas o declaradas, ya que son subclases de RuntimeException.
- Ejemplos: NullPointerException (cuando se intenta acceder a un objeto nulo), ArrayIndexOutOfBoundsException (cuando se accede a un índice inválido en un arreglo).

Errores (Errors)

- ¿Qué son?: Son problemas graves que generalmente están fuera del control del programador y que no deberían tratarse.
- Ejemplos: OutOfMemoryError (cuando la JVM se queda sin memoria), StackOverflowError (cuando ocurre una recursión infinita).



Bloque "try-catch" en Java

Estructura básica del bloque "try-catch":

```
try {
    // Código que puede generar una excepción
} catch (ExcepcionTipo1 e) {
    // Manejo de la excepción tipo 1
} catch (ExcepcionTipo2 e) {
    // Manejo de la excepción tipo 2
} finally {
    // Código opcional que se ejecuta siempre, con o sin excepción
}
```

El código dentro del bloque "try" se monitorea en busca de excepciones. Si se produce alguna excepción, el flujo del programa se desvía al bloque "catch" correspondiente.

Multi-catch en Java

Multi-catch permite capturar varias excepciones en un solo bloque "catch".

Sintaxis

```
try {
  // code
} catch (ExceptionType1 | Exceptiontype2 ex) {
  // catch block
}
```

Ejemplo de uso

```
class Main {
  public static void main(String[] args) {
    try {
      int array[] = new int[10];
      array[10] = 30 / 0;
    } catch (ArithmeticException | ArrayIndexOutOfBoundsException e) {
      System.out.println(e.getMessage());
    }
  }
}
```

Ejemplo de Multi-catch en Java

Se muestra un método que realiza una operación aritmética y puede generar dos tipos de excepciones.

El bloque "catch" captura tanto ArithmeticException como IllegalArgumentException, lo que permite un manejo más conciso de ambas excepciones.

```
public int dividir(int numerador, int denominador) throws
ArithmeticException, IllegalArgumentException {

if (denominador == 0) {

throw new IllegalArgumentException("El denominador no puede ser
cero");
    }

    return numerador / denominador;
}
```

Try with Resources

Try with Resources es una forma más elegante de manejar recursos que implementan la interfaz AutoCloseable.

Sintaxis

```
try (resource declaration) {
  // use of the resource
} catch (ExceptionType e1) {
  // catch block
}
```

Ejemplo Try with Resources en Java

Al finalizar el bloque "try", el recurso BufferedReader se cerrará automáticamente, sin necesidad de escribir un bloque "finally".

```
try (BufferedReader br = new BufferedReader(new
FileReader("archivo.txt"))) {

    String linea;
    while ((linea = br.readLine()) != null) {
        System.out.println(linea);
    }
} catch (IOException e) {
    System.out.println("Error al leer el archivo: " + e.getMessage());
}

}
```

¡Gracias!