

# Archivos II

**UNRN**

Universidad Nacional  
de Río Negro

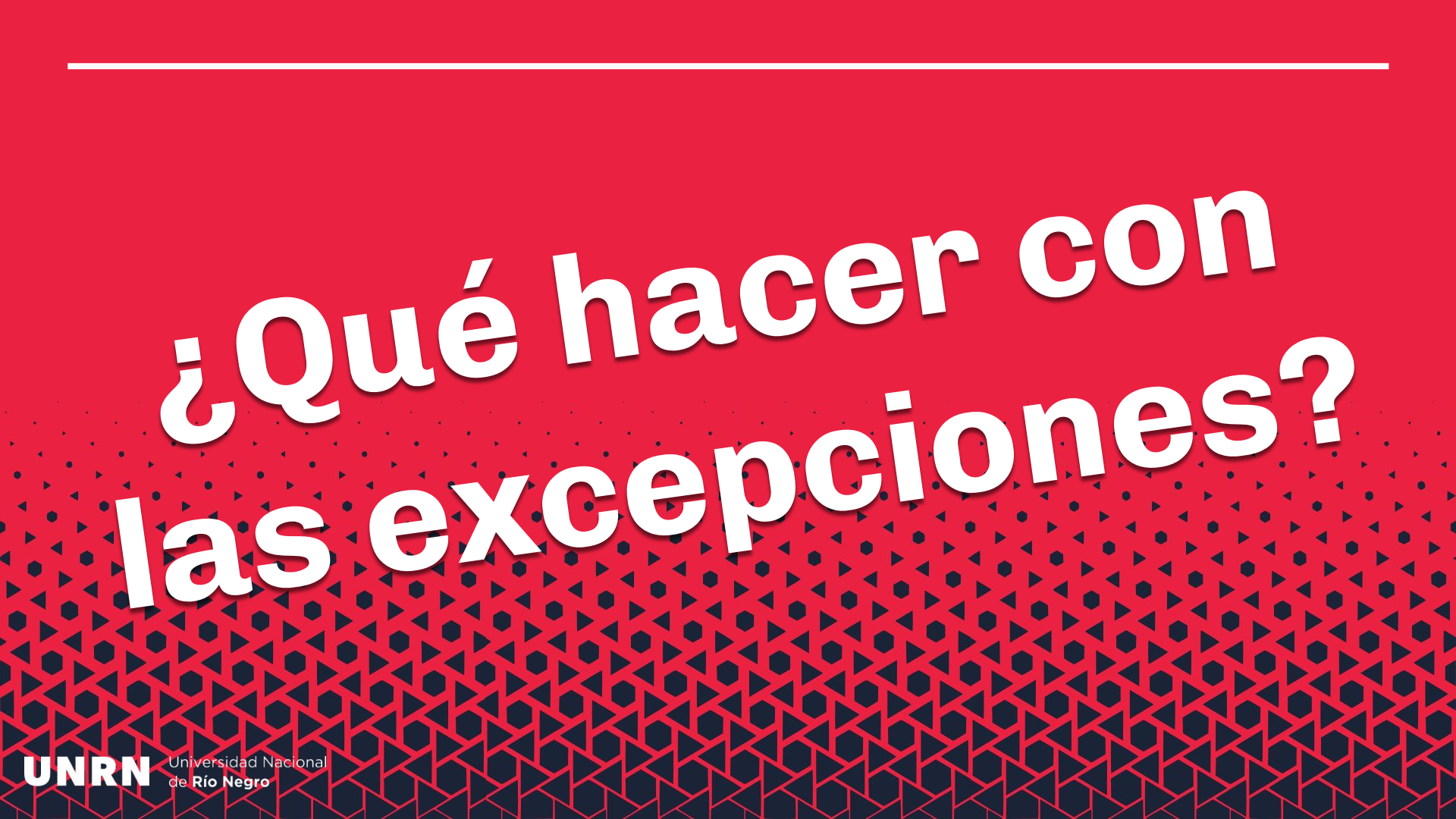
**VIII**

**2025**



---

# ¿Cómo armar los tests?



---

# ¿Qué hacer con las excepciones?

i

Nada...

!



# Preparativos y limpieza por caso

## @BeforeEach Se ejecuta antes de cada test

```
@BeforeEach  
void preparativosPorCasoDePrueba(){
```

**Esto *implica* variables compartidas**

# Podemos crear el archivo para cada test

```
class ArchivosTest {  
    Path archivoPrueba;  
    @BeforeEach  
    void preparativosPorCasoDePrueba() throws IOException {  
        archivoPrueba = Path.of(".", "temp.txt");  
        Files.create(archivoPrueba);  
    }  
}
```

## @AfterEach Se ejecuta antes de cada test

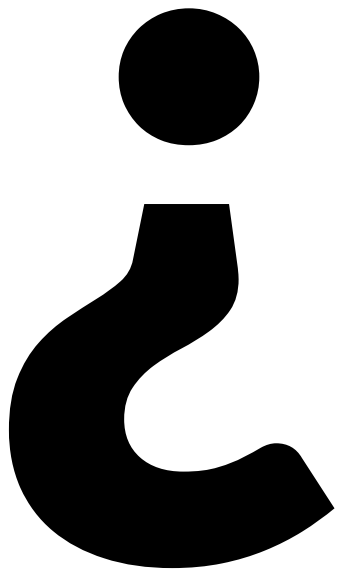
```
@AfterEach  
void limpiezaPorCasoDePrueba(){
```

**Para dejar todo ordenado al finalizar cada caso**



# Y borramos el archivo

```
class ArchivosTest {  
    Path archivoPrueba;  
    @AfterEach  
    void limpiezasPorCasoDePrueba() throws IOException {  
        Files.delete(archivoPrueba);  
    }  
}
```



**Por qué  
delegamos la  
excepción**



**No hay nada que  
hacer**

**más que leer la excepción y ver por que sucedió**



**¿Preguntas?**

---

# Documentación

**UNRN**

Universidad Nacional  
de Río Negro

# Etiqueta @throws de Javadoc

```
/**  
 * Devuelve la división entera de dos números.  
 * @param dividendo operando de la operación  
 * @param divisor que dividirá el dividendo  
 * @returns el dividendo divido por divisor  
 * @throws DivisionPorCeroException cuando el divisor es cero  
 */  
private static long division(long dividendo, long divisor) {
```

**¿De qué familia es `DivisionPorCeroException` ?**



—

**Para las de la familia  
RuntimeException se  
debe documentar  
como evitarla**

**Todas las  
excepciones que  
lancemos deben de  
estar documentadas  
@throws**

**Las excepciones de  
tiempo de  
ejecución deben  
documentar como  
evitar su  
lanzamiento**



**¿Preguntas?**

---

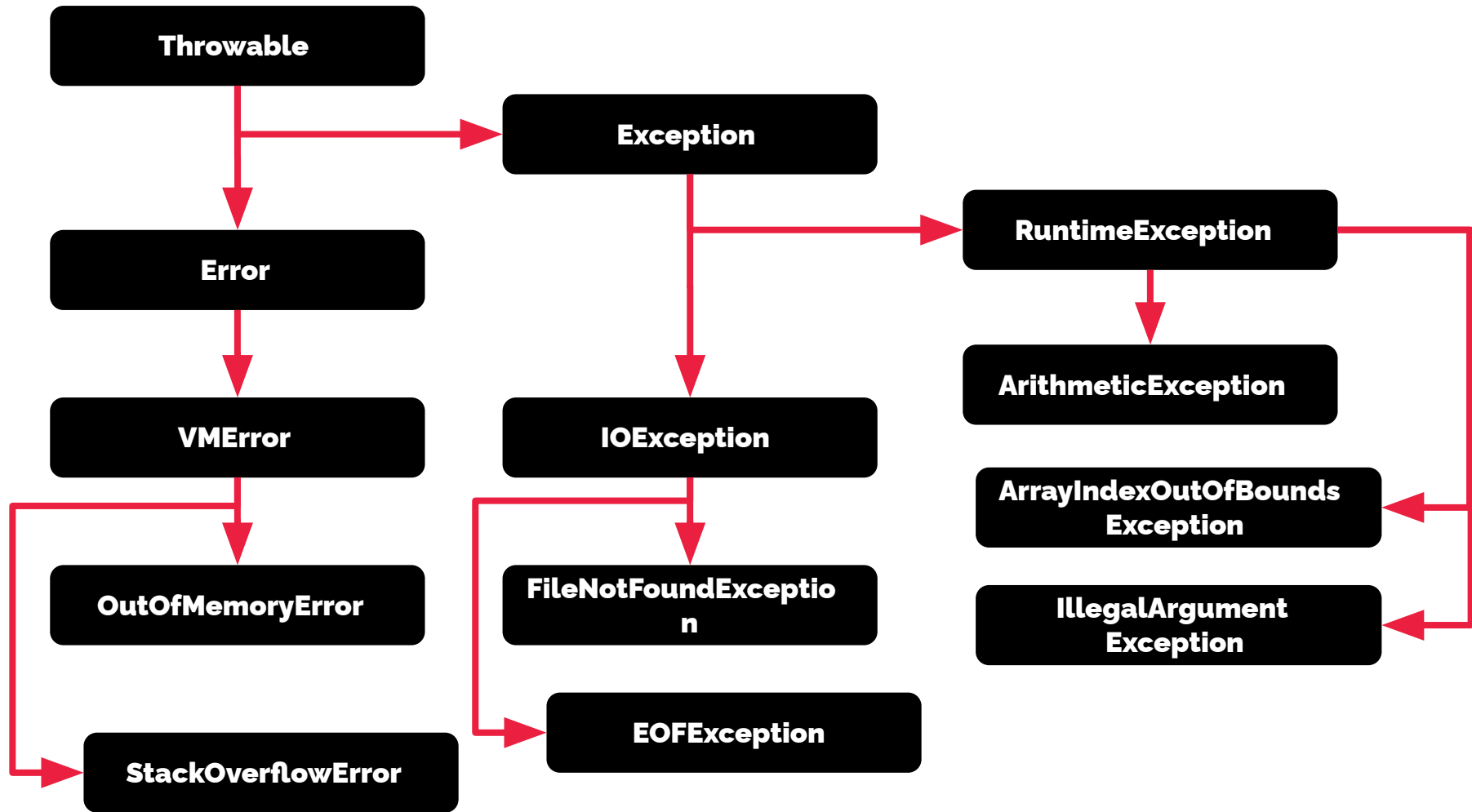
# Tipificación de excepciones

# No todas las situaciones son iguales

**Como  
han visto  
en el TP3**



**Podemos  
'clasificarlas' para  
darles tratamientos  
específicos**





**Esto es una  
relación  
"es un"**

**FileNotFoundException**  
**es una clase de**  
**IOException**

**Al atajar, lo  
hacemos desde  
donde indicamos  
hacia abajo.**

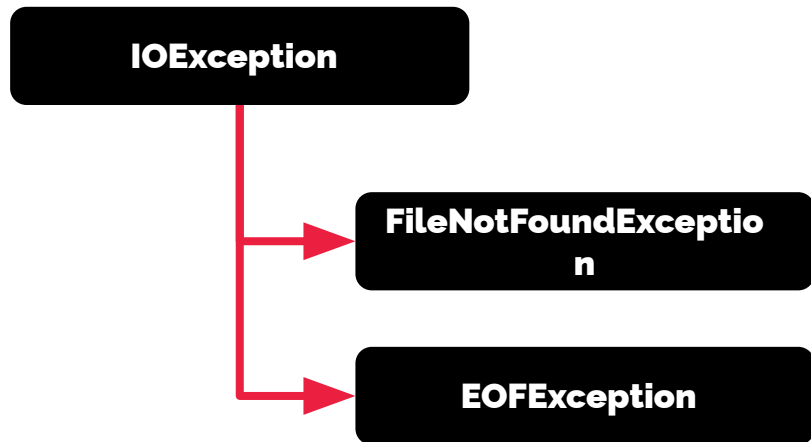


**Dentro de la  
misma familia**



---

# Gestión especializada



Podemos tratar todo de la misma manera como `IOException`

○

Pero podemos gestionar puntualmente los tipos de excepción.

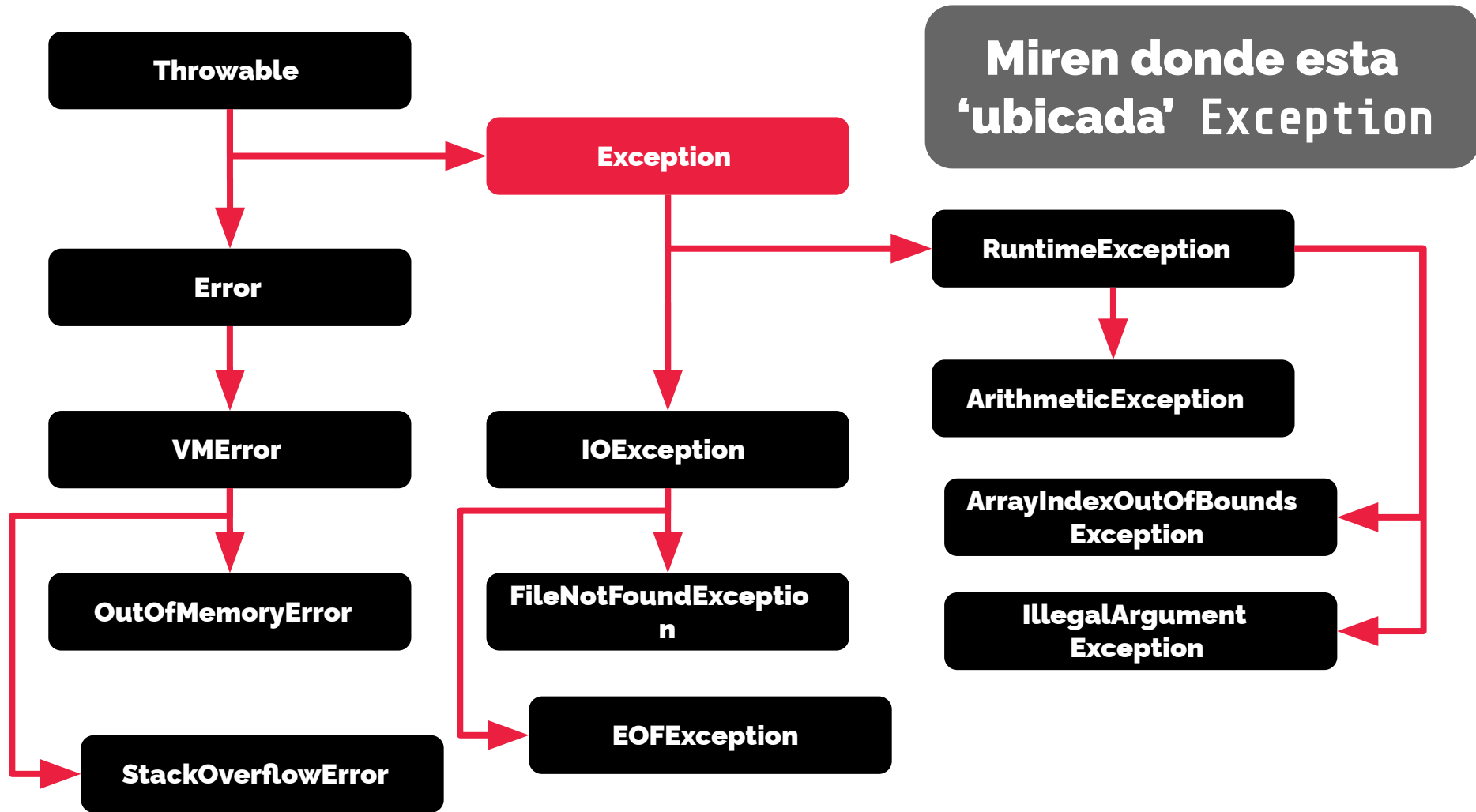


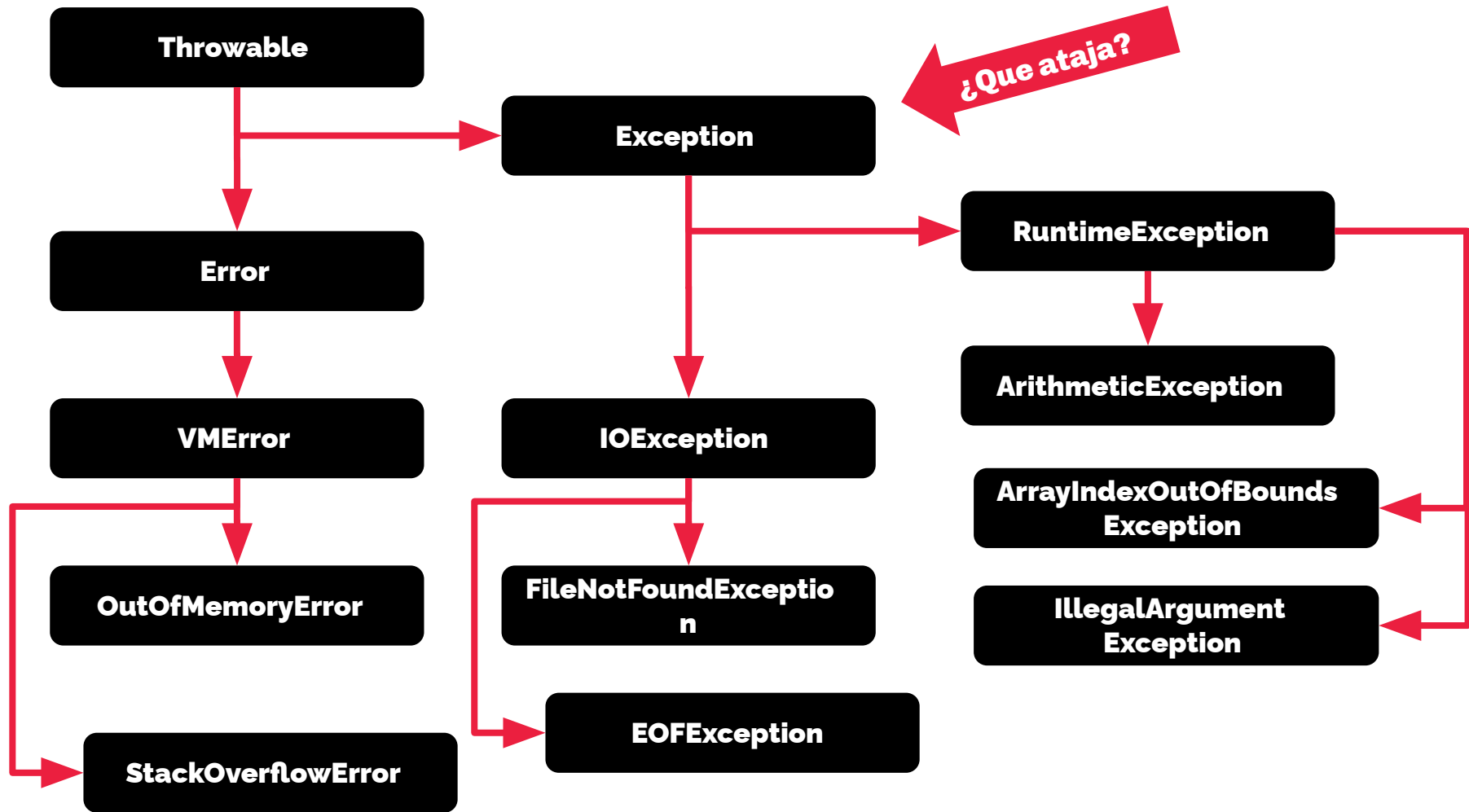
**Esto significa  
dos cuestiones  
importantes**

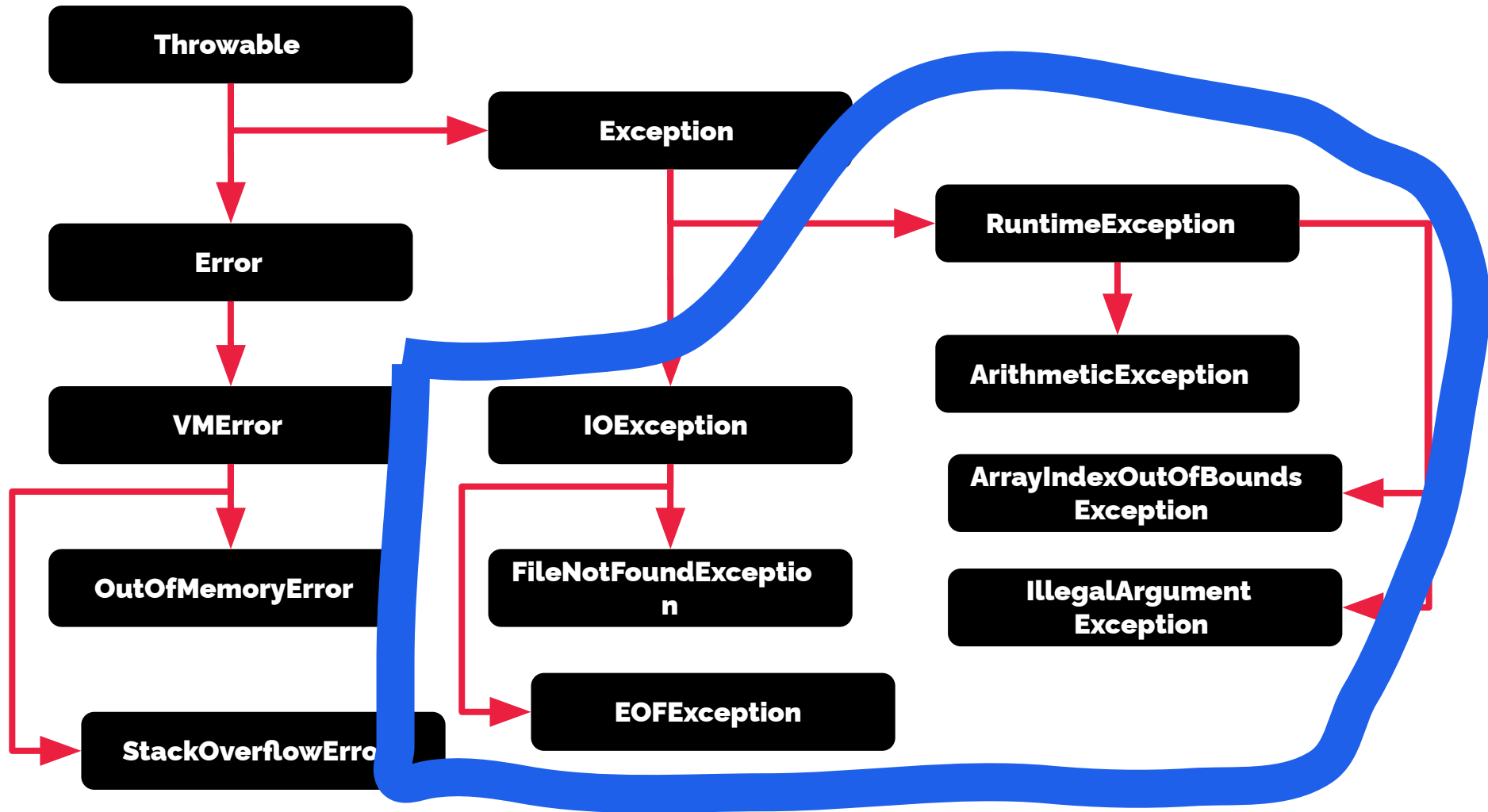
# ¿Qué pasa en este ejemplo?

1

```
try {  
    código con lanzamiento de excepción  
} catch (Exception exc) {  
    ¿Que se ataja en este punto?  
}
```





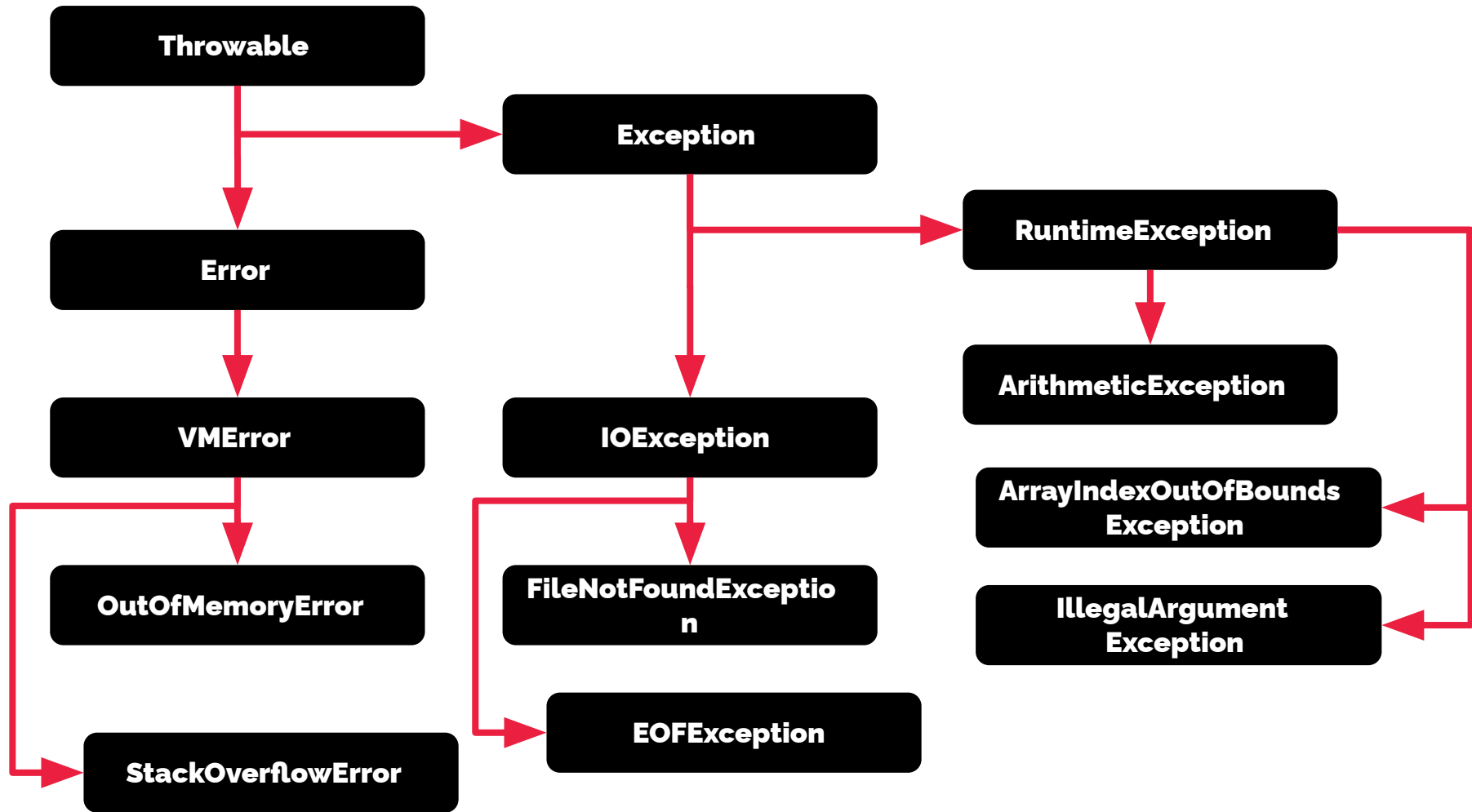




# Por lo que en este, ¿qué pasa?

2

```
try {  
    código con lanzamiento de excepción  
} catch (Exception exc) {  
    ¿Que se ataja en este punto?  
} catch (ArithmeticException exc) {  
    ¿y acá?  
}
```



**Sean específicos  
con lo que atajan,  
no está permitido atajar  
Exception o RuntimeException**



**¿Preguntas?**

---

# Al atajar,

**UNRN**

Universidad Nacional  
de Río Negro

—

**No es cuestión de  
hacer `try/catch`  
por todos lados**

**Y, sí solo hacen un  
print/printStackTrace**

**Dejen que la  
excepción siga su  
camino**



**Ya que solo la  
“silencian”**

**Con archivos es  
particularmente  
notorio**

# ¿Esta función puede cumplir con su objetivo siempre?

```
public static File crearArchivo(String nombre) {  
    File archivo = new File(nombre);  
    try {  
        archivo.createNewFile();  
    } catch (IOException exc) {  
        exc.printStackTrace();  
    }  
    return archivo;  
}
```

# ¿Esta función puede cumplir con su objetivo siempre?

```
public static File crearArchivo(String nombre) {  
    File archivo = new File(nombre);  
    try {  
        archivo.createNewFile();  
    } catch (IOException exc) {  
        exc.printStackTrace();  
    }  
    return archivo;  
}
```

# ¿Y acá? ¿Qué pasa si 'archivo' ya existía?

```
public static void escribirInforme(String archivo, String[] informe){  
    File destino = crearArchivo(archivo);  
    escribir(destino, informe);  
}
```

**Es importante  
pensar en el  
“usuario” de la  
función**

---

# uno mismo

Necesito saber **que** falló y **como** fallo para tomar la decisión correcta

# No apuren la captura de la excepción



# Da una **falsa** sensación de seguridad

```
public static File crearArchivo(String nombre) {  
    File archivo = new File(nombre);  
    try {  
        archivo.createNewFile();  
    } catch (IOException exc) {  
        exc.printStackTrace();  
    }  
    return archivo;  
}
```

---

**Simplemente,  
no hay nada  
que hacer**

# ¿Esta función puede cumplir con su objetivo siempre?

```
public static File crearArchivo(String nombre)
    throws IOException{
    File archivo = new File(nombre);
    archivo.createNewFile();
    return archivo;
}
```



**¿Preguntas?**

# Documentación de Excepciones

---

# Etiqueta @throws

```
/**  
 *  
 * @throws ArregloExcepcion ¿cuando se lanza?  
 * @throws IOException ¿Cuando se lanza?  
 */
```

# Por ejemplo

```
/**  
 *Pide un número entero, con un mensaje personalizado  
 * y una cantidad limitada de intentos  
 * @throws NoMasIntentosException cuando agotamos los intentos  
 */  
public static int pideInt(String mensaje, int intentos)  
    throws NoMasIntentosException {  
    código que resuelve el ejercicio  
}
```

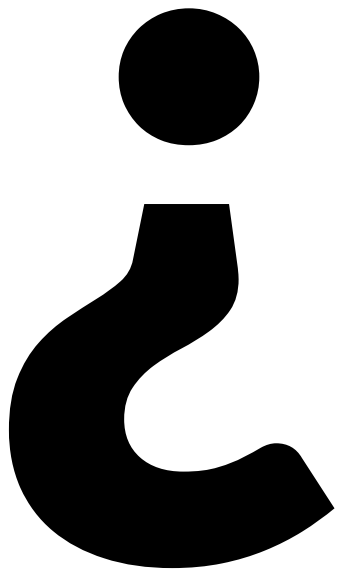
Solo para Exception

**Idealmente, se  
documenta todo lo  
que sepan que  
aparece.**





**¿Preguntas?**



**¿Podemos  
atajar y lanzar  
la misma  
excepción?**



i SÍ!

—

**Sí**empre y cuando  
sea para agregar  
información

# ¡Atajamos y lanzamos **casi** lo mismo!

```
public static int[] cadenasANumeros(String[] strings) {  
    int[] arreglo = new int[strings.length];  
    for (int i = 0; i < strings.length; i++) {  
        try {  
            arreglo[i] = Integer.parseInt(strings[i]);  
        } catch (NumberFormatException e) {  
            throw new NumberFormatException("Error al convertir el elemento "  
+ i + " a entero: " + strings[i]);  
        }  
    }  
    return arreglo;  
}
```

# Y su respectiva documentación

```
/**  
 * Convierte un arreglo de cadenas de texto a un arreglo de enteros.  
 *  
 * @param strings contiene las cadenas que se desea convertir.  
 * @return Un arreglo de enteros con los valores convertidos.  
 * @throws NumberFormatException Si alguna cadena no puede ser  
 *                               convertida a entero, indicando cuál falló.  
 */
```

—

**El error en sí es lo  
mismo, pero ahora  
tenemos mejor  
contexto**





**¿Preguntas?**

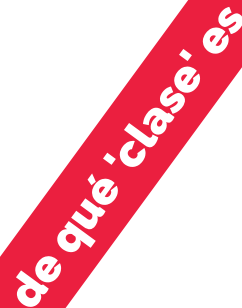


---

# Creación de Excepciones

# En un archivo separado

```
public class NoMasIntentosException extends Exception{  
    public NoMasIntentosException(){  
        super();  
    }  
}
```



de qué 'clase' es

# En un archivo separado

```
public class NoMasIntentosException extends RuntimeException{  
    public NoMasIntentosException(){  
        super();  
    }  
}
```



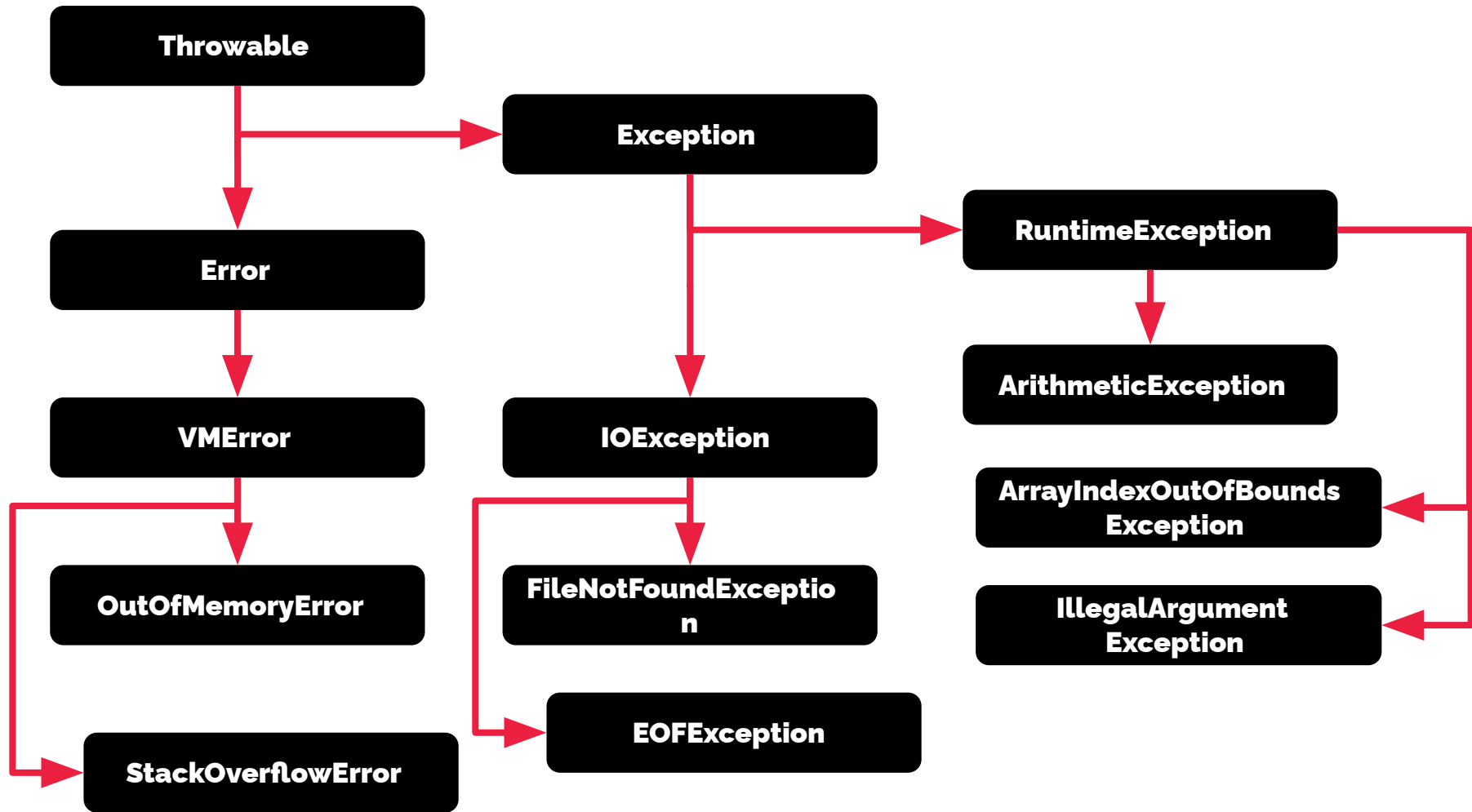
de qué 'clase' es

**El TP3 tiene un  
ejemplo **mucho**  
más completo.**

# Se pueden conectar a cualquier parte de la familia

```
public class ArchivoNoExisteException extends IOException{  
    public NoMasIntentosException(){  
        super();  
    }  
}
```





**Las excepciones  
propias deben  
pertenecer a una  
familia propia**

**unrn.edu.ar**

**UNRN**

Universidad Nacional  
de **Río Negro**



| **unrionegro**