

# Arreglos y excepciones

**UNRN**

Universidad Nacional  
de Río Negro



# Arreglos

**Una secuencia *mutable*\* y  
de elementos de *un tipo*  
con un largo fijo**

```
tipo[]
identificador;
```



```
int[] arr = {1, 2, 3};
```

Tamaño automático

```
int[] arr = new int[10];
```

Tamaño manual de diez elementos e inicializado en cero

---

# Con información adicional

```
int[] arr = new int[100];  
arr.length
```



---

# Uso con un lazo definido

```
int[] arr = new int[100];
for (int i = 0; i < arr.length; i++){
    arr[i] = i;
}
```

# Lazo 'for-each'

```
int[] arr = new int[100];
for (int valor : arr) {
    System.out.println(valor);
}
```

Pero solo permite **leer** del arreglo (y sin la posición)

# Admite inferencia de tipo

```
int[] arr = new int[100];
for (var valor : arr) {
    System.out.println(valor);
}
```

# Declaración e inicialización de matrices

```
int[][] matrixUno;  
int[][] matrixDos = {{1,2},{3,4}};  
var matrixTres = new int[3][4][8]; //¡3D!
```

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

**¿Preguntas?**



# Testing de arreglos

# Aserción de igualdad de arreglos

```
assertArrayEquals(primitivo[] esperado, primitivo[] actual, String mensaje)
```

# assertArrayEquals

```
@Test
public void igualdadArreglos() {
    char[] esperado = {'U','N','R','N',''};
    char[] resultado = "UNRN".toCharArray();

    assertArrayEquals(esperado, resultado);
}
```

Más sobre el  
acceso al  
contenido

---

# ¿Cuál es la salida acá?

```
public class ArregloApp {  
    public static void main(String[] args) {  
        int[] numeros = {1, 2, 3};  
        System.out.println(numeros[3]);  
    }  
}
```

# ¿Cuál es la salida acá?

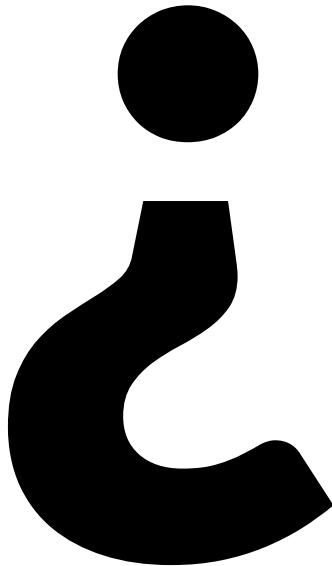
```
public class ArregloApp {  
    public static void main(String[] args) {  
        int[] numeros = {1, 2, 3};  
        System.out.println(numeros[3])  
    }  
}
```



# Vamos a ver algo como

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: Index 3 out of
bounds for length 3
at ar.unrn.ArregloApp.main(ArregloApp.java:21)
```

# Gestión de errores excepciones

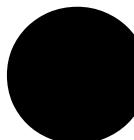


# Qué son



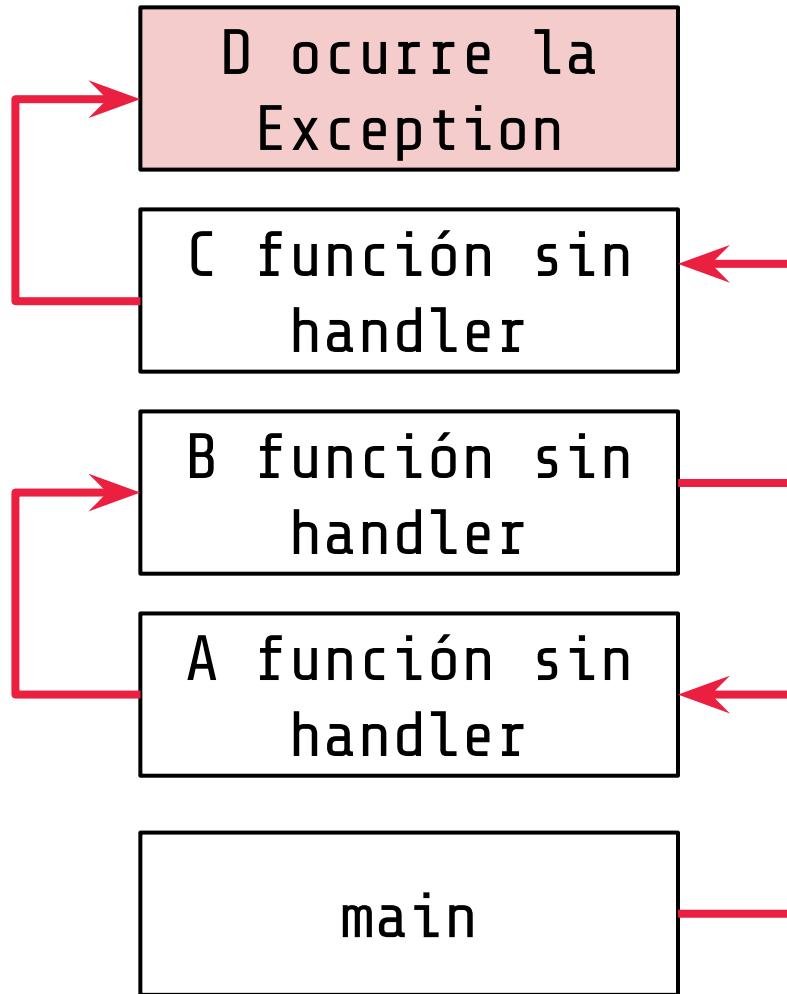
**Cualquier  
interrupción al flujo  
de instrucciones  
normal del  
programa**

**¿Que son?**

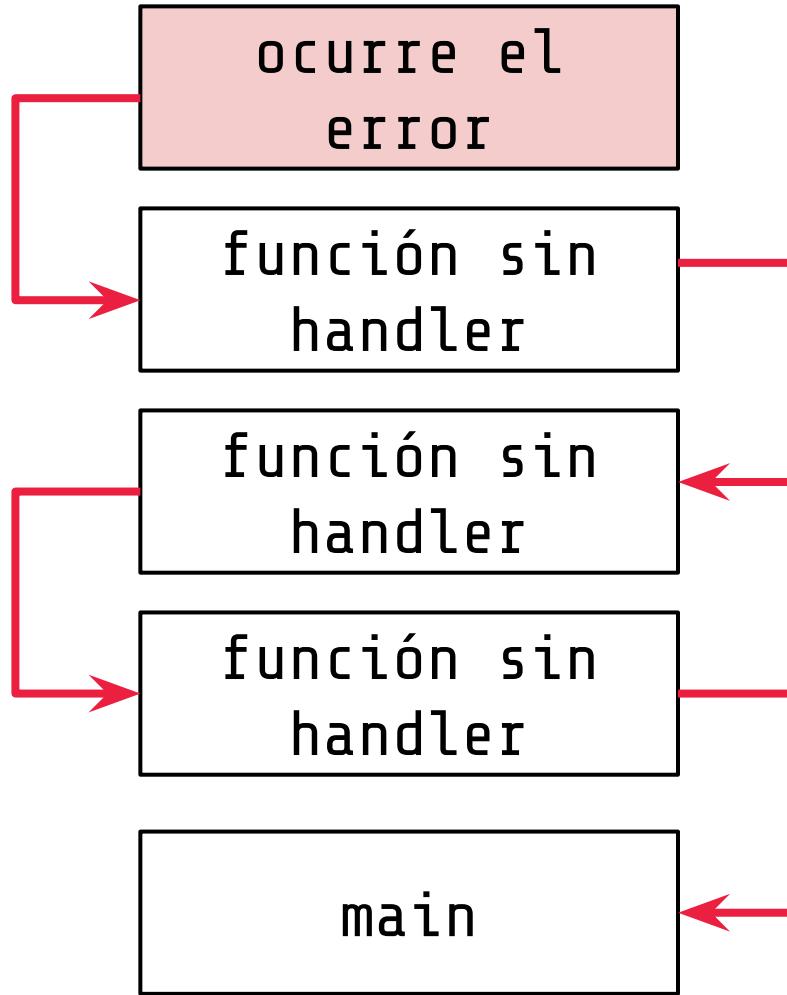


# Como funcionan

**Todo viene bien  
hasta que...**



# Buscando quien se haga cargo



# **Si nadie lo ataja**

# Un ejemplo concreto (y simple)

```
public class ExplosionApp {  
    public static void main(String[] args) {  
        a();  
    }  
    static void a() {  
        b();  
    }  
    static void b() {  
        c();  
    }  
    static void c() {  
        d();  
    }  
    static void d() {  
        1/0;  
    }  
}
```



¡BOOM!

# Esto es lo que veríamos

```
Exception in thread "main" java.lang.ArithmetricException: / by zero
at ar.unrn.ExplosionApp.d(ExplosionApp.java:23)
at ar.unrn.ExplosionApp.c(ExplosionApp.java:19)
at ar.unrn.ExplosionApp.b(ExplosionApp.java:15)
at ar.unrn.ExplosionApp.a(ExplosionApp.java:11)
at ar.unrn.ExplosionApp.main(ExplosionApp.java:7)
```



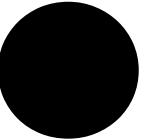
**Esto aplica a cualquier  
situación 'fuera de lo  
normal'**



**¿Preguntas?**



**Hasta ahí, es solo otro  
ejemplo de cómo vemos  
un error**



**Pero ¿qué  
podemos  
hacer?**

**1**

# **Atajar**

2

# Delegar

# Atajadas



# Como atajar Excepciones

# ¡Pavada de atajada!

```
try {  
    argentina.getWorldCup(2023).getFinal().ganar();  
} catch (MbappeException exc){
```



```
}
```

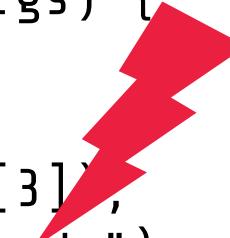
**Por ahora, para saber  
que falla es ver la  
documentación o  
*hacerlo fallar***

# Y ahora, ¿cuál es la salida?

```
public class ArregloApp {  
    public static void main(String[] args) {  
        int[] numeros = {1, 2, 3};  
        try {  
            System.out.println(numeros[3]);  
            System.out.println("Hola Mundo");  
        } catch (ArrayIndexOutOfBoundsException exc) {  
            System.out.println("Te pasaste");  
        }  
    }  
}
```

# Y ahora, ¿cuál es la salida?

```
public class ArregloApp {  
    public static void main(String[] args) {  
        int[] numeros = {1, 2, 3};  
        try {  
            System.out.println(numeros[3]);  
            System.out.println("Hola Mundo");  
        } catch (ArrayIndexOutOfBoundsException exc) {  
            System.out.println("Te pasaste");  
        }  
    }  
}
```

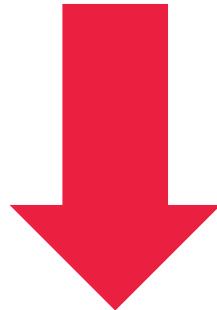


**Por donde va el  
programa es  
fundamental con las  
Excepciones**

# Atajar significa que se puede hacer algo

```
try {  
    // Código que puede fallar  
    // ¡Pero también lo que depende!  
}catch (TipoExcepction exc){  
    // Qué hacer cuando falle, intentamos recuperarnos  
}
```

Y sigue su curso



**Esta estructura  
admite variaciones**

# Cuando el código puede fallar de múltiples formas

```
try {
    argentina.getWorldCup(2023).getFinal().ganar();
} catch (FranceException exc) {
    // Messi
} catch (MbappeException exc) {
    // Dibu Martinez
}
```

# Cuando queremos responder de la misma forma \*

```
catch (MbeppException | FranceException exc) {
```



A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

**¿Preguntas?**



---

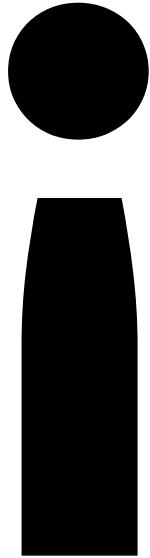
# Delegar

# En este caso, ¿qué podemos hacer?

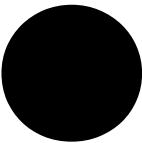
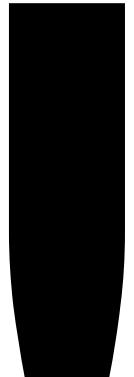
¿Pero quién  
puede hacer  
algo?

```
int divide(int dividendo, int divisor){  
    return dividendo / divisor;  
}
```

esta puede fallar



**Son una forma  
de dar más  
información**





**¿Preguntas?**

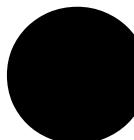


# Como regla general

**Es mejor prevenir  
que atajar**

**Un  
ArrayIndexOutOfBoundsException  
es un condicional  
que no funcionó**

**Ya vamos a ver situaciones que si o si necesitan ataje**



**Como se  
prueba esto**

# Como para mejorar la División Lenta

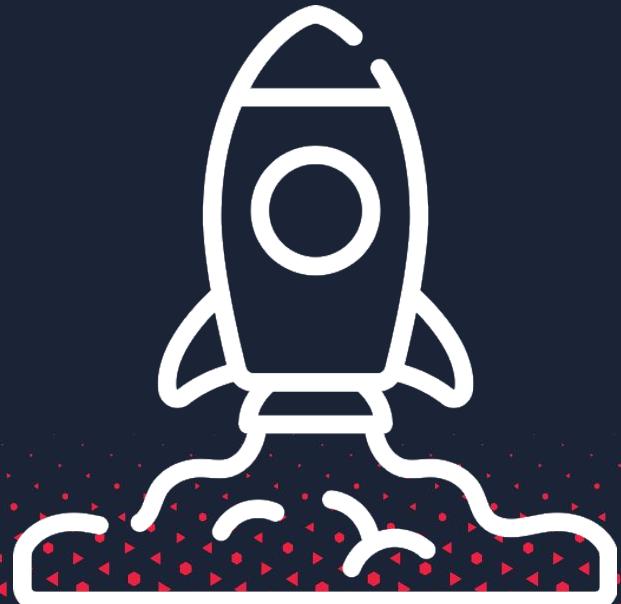
```
@Test
void debieraFallar(){
    try{
        var esperado = DivisionApp.dividir(1,0);
        fail("Debió saltar al catch");
    } catch (ArithmeticException exc){
        ; // un print también vá
    }
}
```



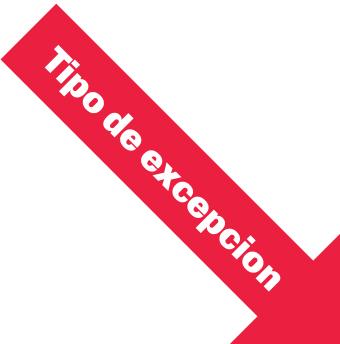
**¿Preguntas?**



# Lanzamiento de excepciones



# Lanzamiento



```
throw new ArithmeticException("division por cero");
```

Pero, *puede \* ser necesario* declararla

```
public static int leerArchivo(String nombreArchivo)  
    throws IOException
```

Ahorita vamos a ver cuáles requieren declaración y cuáles no.

**De momento  
vamos a usar  
Exceptions ya  
creadas**

**En el TP3 hay un  
ejemplo**

---

# Familias de excepciones

# De tiempo de ejecución (`RuntimeException`)

---

# Que son (a grandes rasgos) prevenibles

ArithmeticException

No dividir por cero

ArrayIndexOutOfBoundsException

No pasarse de los límites

StringIndexOutOfBoundsException

No convertir algo incorrecto

NumberFormatException

# Esto lanza una ArithmeticException

```
public static int division(int valor) {  
    return valor / 0;  
}
```

Que *simplemente* aparece

**Ver una de este tipo  
es *generalmente* un  
bug**

*Controladas  
'checked'*

# de tipo (Exception)

A grandes rasgos,  
inevitables y  
tenemos que estar  
preparados

**El uso de archivos\*  
es un buen ejemplo**

# **Es necesario declarar su lanzamiento**

```
public static String[] leerArchivo(String nombre) throws ArchivoException
```

**Que obliga a tenerla en cuenta**

**Podemos delegar,  
pero en algún  
momento**

---

# Alguien la tiene que atajar

```
try {  
    String[] datos = leerArchivo("dredd");  
} except (ArchivoException excepcion) {  
    // ¿podemos cambiar de archivo?  
}
```

**En algunos casos,  
no podemos hacer  
nada.**

**Que iremos viendo  
gradualmente**

# —

# Errores (Error)

**Situaciones en  
donde el programa  
completo no se  
puede recuperar**

*OutOfMemoryError*  
*StackOverflowError*



guess I'll die

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

**¿Preguntas?**



---

# ¿Qué tipo elegir?

**Qué tipo elegir  
es un tema de  
debate**

# Por esto mismo



A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

**¿Preguntas?**



---

# Testing de excepciones

# Un uso de Assertions.fail

```
@Test
public void testDivision(){
    int argumento1 = 4;
    int argumento2 = 0;
    try {
        int resultado = division_lenta(argumento1, argumento2);
        fail("La excepción no fue lanzada :-(");
    } catch (DivisionPorCero excepcion) {
        ; // no hay nada que hacer, la idea es que falle
    }
}
```

**¡El fallo es no lanzar la excepción!**

# Cuando la función lanza una Exception

```
@Test
public void testDivision() throws DivisionPorCero {
    int argumento1 = 4;
    int argumento2 = 2;
    int esperado = 2; // argumento1 + argumento2 esta bien
    int resultado = division_lenta(argumento1, argumento2);
    assertEquals(esperado, resultado, "no coincide");
}
```



Y no nos interesa, es necesario declararla (y simplificar el test)



**¿Preguntas?**



# TP3

## Arreglos y excepciones

# *Cuestiones de estilo*

**No atajar la  
excepción si no es  
possible tomar una  
decisión**

**El main de un  
programa no debe  
dejar pasar  
excepciones de tipo**

# Qué familia de excepciones se eligió debe de estar documentada

Declarar el  
lanzamiento de  
una Exception sin  
tipo es un error

**unrn.edu.ar**

**UNRN**

Universidad Nacional  
de Río Negro



| unrnionegro