



UNRN

Universidad Nacional
de Río Negro



| unrionegro

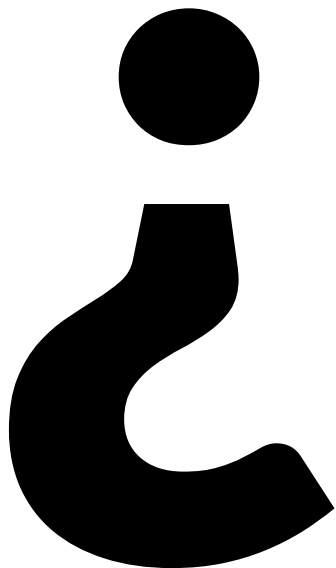
Programación 2

X

UNRN

Universidad Nacional
de **Río Negro**

Martín René Vilugrón
mrvilugron@unrn.edu.ar



Observaciones sobre excepciones



**Si solo hacen un
print/printStackTrace**

**Dejen que la
excepción siga su
camino**

**Ya que solo la
“silencian”**

**Con archivos es
particularmente
visibles**

¿Esta función puede cumplir con su objetivo siempre?

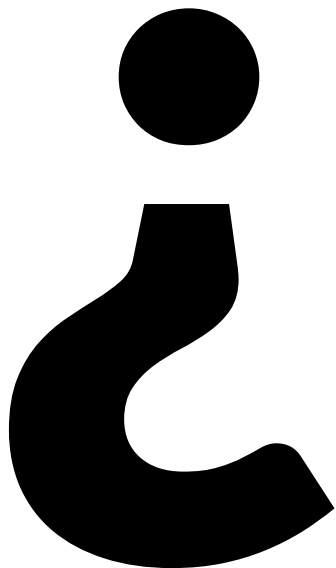
```
public static File crearArchivo(String nombre) {  
    File archivo = new File(nombre);  
    try {  
        archivo.createNewFile();  
    } catch (IOException exc) {  
        exc.printStackTrace();  
    }  
    return archivo;  
}
```


¿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;  
}
```

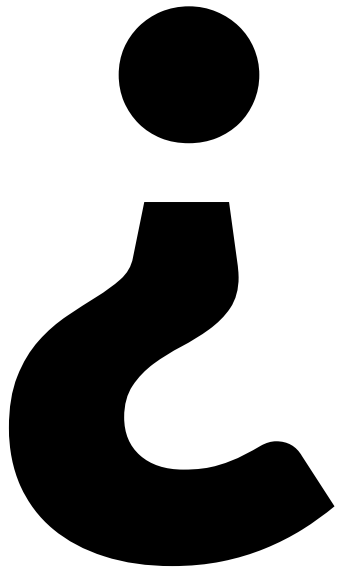
¿Y acá? ¿Que pasa si 'archivo' ya existía?

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



**¿Y si el archivo ya
existía?**





**¿Y si el archivo no
se puede
escribir?**



**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;
}
```

De esta manera cuando lleguen a escribir...

```
public static void escribirInforme(String archivo, String[] informe)
    throws IOException{
    File destino;
    try{
        destino = crearArchivo(archivo);
    }catch (FileNotFoundException exc){
        throw new ArchivoNoEncontrado(exc);
    }
    escribir(destino, informe);
}
```



¿Preguntas?

**Como vamos a usar
archivos a futuro,
este tema lo
podemos dejar para
luego**

Aparte

printStackTrace

es un `print` con pasos adicionales

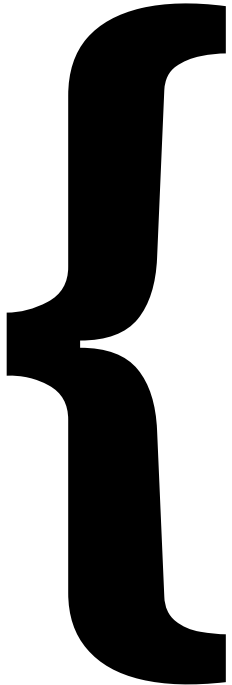
Y como tales, *tecnicamente* no van dentro de las funciones



**Mantengan
separado lo que
interactúa con el
archivo**



**De lo que hace el
trabajo con su
información**



**Es más fácil de
armar tests**



Son mas simples



**Y no siempre
necesitan
interactuar con
archivos**



Excepciones III

Una forma mas completa de armar excepciones

```
public static class LecturaArchivoException extends Exception{  
    public LecturaArchivoInvalidoException(){  
        super();  
    }  
    public LecturaArchivoInvalidoException(String razon){  
        super(razon);  
    }  
    public LecturaArchivoInvalidoException(Throwable excepcionOrigen){  
        super(excepcionOrigen);  
    }  
}
```



¿Preguntas?

Un uso netamente *erroneo*

Para 'suavizar' una excepción

```
try{  
    //código que puede fallar  
catch (IOException exc){  
    throw new RuntimeException(exc);  
}
```



¿Preguntas?

Hagamos Tiempo!

```
class Tiempo{  
    public Tiempo(int hora, int minutos, int segundos);  
    public Tiempo(int segundos);  
    public Tiempo sumar(int segundos);  
    public Tiempo sumar(Tiempo otro);  
    public Tiempo restar(int segundos);  
    public Tiempo restar(Tiempo otro);  
    public int comparar(Tiempo otro);  
}
```

Versión 1

```
class Tiempo{
    private int hora;
    private int minutos;
    private int segundos;

    public Tiempo(int hora, int minutos, int segundos);
    public Tiempo(int segundos);
    public void sumar(int segundos);
    public void sumar(Tiempo otro);
    public void restar(int segundos);
    public void restar(Tiempo otro);
    public int comparar(Tiempo otro);
}
```

Versión 2

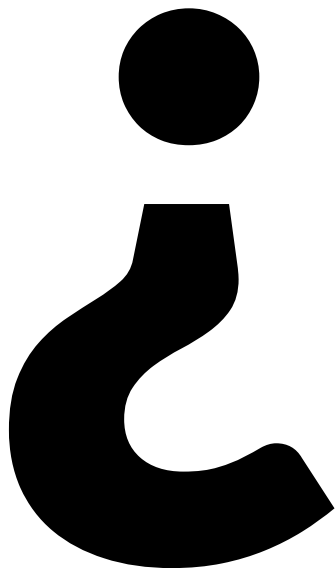
```
class Tiempo{  
    private int segundos;  
  
    public Tiempo(int hora, int minutos, int segundos);  
    public Tiempo(int segundos);  
    public void sumar(int segundos);  
    public void sumar(Tiempo otro);  
    public void restar(int segundos);  
    public void restar(Tiempo otro);  
    public int comparar(Tiempo otro);  
}
```

Versión 3

```
class Tiempo{  
    public Tiempo(int hora, int minutos, int segundos);  
    public Tiempo(int segundos);  
    public Tiempo sumar(segundos);  
    public Tiempo sumar(Tiempo otro);  
    public Tiempo restar(int segundos);  
    public Tiempo restar(Tiempo otro);  
    public int comparar(Tiempo otro);  
}
```

Versión 3

```
class Tiempo{
    public Tiempo(int hora, int minutos, int segundos);
    public Tiempo(int segundos);
    public Tiempo sumar(segundos);
    public Tiempo sumar(Tiempo otro);
    public Tiempo restar(int segundos);
    public Tiempo restar(Tiempo otro);
    public int comparar(Tiempo otro);
    public int aSegundos();
}
```



**Que cambió entre
ambas**



Un método como

`public Tiempo sumar(segundos);`

asume que no cambia

Que es **inmutable
(como String)**

**Útil para pasar
como argumento**

unrn.edu.ar

UNRN

Universidad Nacional
de **Río Negro**



| **unrionegro**