

# Archivos

**UNRN**

Universidad Nacional  
de Río Negro

**VII**

**2024**





**¿Preguntas?**

---

# Archivos

## new style

**UNRN**

Universidad Nacional  
de Río Negro

# **Como guardar información más allá del cierre del programa**

---

# rutas

`java.nio.Paths / java.nio.Path`

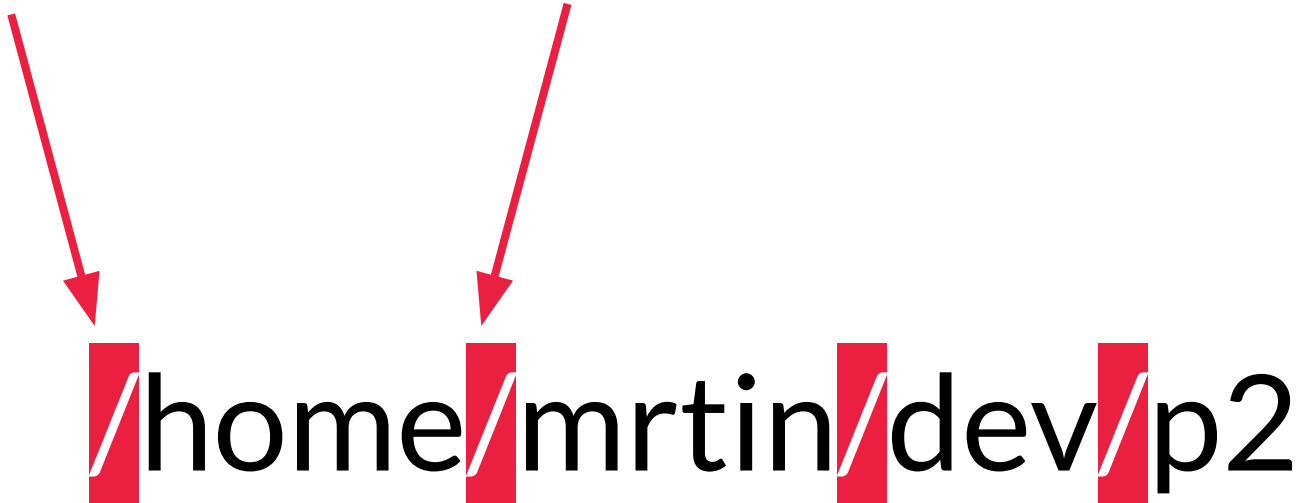


Una ubicación dentro de la estructura de directorios.

Puede apuntar a un archivo o directorio.

**Raíz**

**separador**



**/home/mrtin/dev/p2**

The diagram shows the file path **/home/mrtin/dev/p2**. Four red vertical rectangles are placed over the slashes: the first under the root slash, and the others under the slashes separating 'home', 'mrtin', and 'dev'. A red arrow points from the label 'Raíz' to the first rectangle. Another red arrow points from the label 'separador' to the second rectangle. A third red arrow points from the label 'directorio' to the 'home' directory name.

**directorio**

**Raíz**

**separador**

**C:\home\mrtin\dev\p2**

**unidad**

**directorio**



---

# Hay dos formas de expresar rutas

**absoluta**

`C:\users\usuario\`

`/home/usuario`

**relativa**

`Descargas\archivo`

`documentos/subdirectorio`

# Rutas relativas

```
$> pwd
```

```
/home/mrtin/dev
```

```
$> ls ../
```

```
Descargas, dev, Documentos, Música, Videos
```

```
$> ls p2/tp2/build/reports
```

```
junit.xml, checkstyle.xml
```

```
$> ls ../Descargas/peliculas
```

```
Mi vecino Totoro, John Wick 4
```

**Para navegar en el disco**

# iA la terminal!



---

# Detalle en Windows

```
String ruta = "C:\\home\\martin\\dev\\p2"
```

**ino es valido!**

---

# La \ es un símbolo del lenguaje

```
String ruta = "C:\\home\\martin\\dev\\p2"
```

# File.separator

Contiene el separador correcto al sistema operativo para armar cadenas con rutas.

\ Windows

/ Linux

***Por si tenemos que armar una ruta***



**¿Preguntas?**

# java.nio.file.Paths

**Da una ruta a partir de una expresión,  
*¡puede no existir!***

`Path ← Paths.get(String primero, ...)`

RT



# java.nio.file.Path

```
Path pwd = Paths.get(".");  
Path inex = Paths.get(".", "noexiste");
```



# java.nio.file.Path

```
Path pwd = Paths.get(".");  
sout(pwd.toAbsolutePath());
```

# Mención especial

**Ojo con usar en get  
lo que viene directo  
del usuario**



# Que pasa si...

```
public static void main(String[] args) {  
    ...codigo...  
    Files.delete(Paths.get(args[0]));  
}
```

```
$> java programa "C:\\windows\\System32"
```

**¡Una lectura puede ser igual de dañina!**

---

# Archivos

java.nio.file.Files

# Consultas

`Files.exist(path)`

`Files.notExists(path)`

`Files.isRegularFile(path)`

`Files.isReadable(path)`

`Files.isWritable(path)`

`Files.isSameFile(path1, path2)`

`Files.size(path)`

# Contenido en una ubicación

```
DirectoryStream<Path> stream =  
    Files.newDirectoryStream(path);
```

E



# Para recorrer un directorio

```
Path pwd = Paths.get(".");  
try {  
    DirectoryStream<Path> contenido  
        = Files.newDirectoryStream(pwd);  
    for (Path ruta : contenido) {  
        System.out.println(ruta.toString());  
    }  
    stream.close();  
} catch (IOException exc) {  
    exc.printStackTrace();  
}
```



**Cuando el archivo  
entra entero en  
memoria (~2gb)**

**IOException  
OutOfMemoryError**

# Leer el archivo

```
String ← Files.readString(Path path)
```

**Lee todo el archivo en un String**

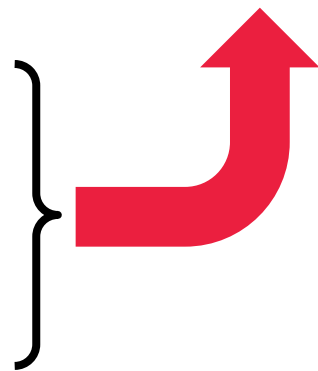
# Escribir a un archivo

```
Files.writeString(path, cadena, modo);
```

```
StandardOpenOption.APPEND
```

```
StandardOpenOption.WRITE
```

```
StandardOpenOption.CREATE
```



# Escritura y lectura todo junto

```
Path ruta = Paths.get(".", "test.txt");
try {
    String cadena = "Hola Mundo!";
    Files.writeString(ruta, cadena, StandardOpenOption.CREATE);
    String contenido = Files.readString(ruta);
    System.out.println(contenido);
} catch (IOException exc) {
    exc.printStackTrace();
}
```

# Todas las operaciones de archivos lanzan IOException



**¿Preguntas?**



# **Formas de crear una cadena**



# java.util.Formatter

```
StringBuilder builder = new StringBuilder();  
Formatter salida = new Formatter(builder);
```

```
salida.format(...igual que printf...);
```

```
salida.close();
```

# Se puede usar con un Path

```
Path ruta = Paths.get(".", "test.txt");
try {
    Formatter salida = new Formatter(ruta.toFile());
    salida.g5(...igual que printf...);
} catch (IOException exc ) {
    gestionamos errores
}
```

# Excepciones con try/catch/finally

# try/catch/finally

```
try {  
    Código que puede fallar  
} catch (excepción) {  
    Gestión de errores  
    O delegación  
} finally {  
    Esto se ejecuta en cualquiera de los dos casos  
}
```

¡Con **catch** opcional!

# try/finally con Formatter

```
Path ruta = Paths.get(".", "test.txt");
Formatter salida = null;
try {
    salida = new Formatter(ruta.toFile());
    salida.format(...igual que printf...);
} catch (IOException exc ) {
    gestion de errores
} finally {
    if (salida != null) {
        salida.close();
    }
}
```

**Aunque es menos compacto que el try/resources**

# try/finally con Scanner

```
Path ruta = Paths.get(".", "test.txt");
Scanner scanner = null;
try {
    scanner = new Scanner(ruta.toFile());
    while (scanner.hasNext()) {
        System.out.println(scanner.nextLine());
    }
} catch (IOException e) {
    gestion de errores
} finally {
    if (scanner != null) {
        scanner.close();
    }
}
```



**Hay una opción  
más simple**

# Try con recurso

Al usar algo que requiera 'cierre' y pueda fallar

```
try ( construcción y asignación del recurso ) {  
    uso del recurso  
} catch (excepciones) {  
    gestion de errores  
}
```

Llama **close** automáticamente al finalizar



# Try with resources

```
Path ruta = Paths.get(".", "test.txt");
try (Scanner scan = new Scanner(ruta)) {
    while (scanner.hasNext()) {
        System.out.println(scanner.nextLine());
    }
} catch (IOException e) {
    gestion de errores
}
```

**Llama close automáticamente**

# Del ejemplo de Path

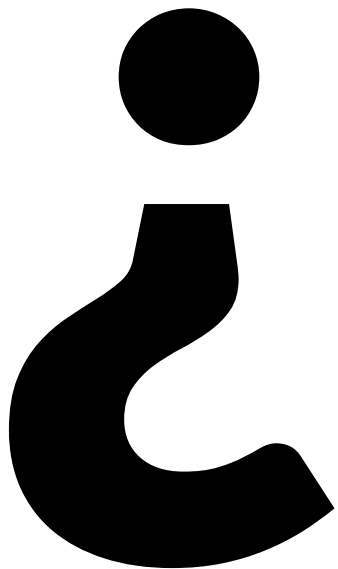
```
Path ruta = Paths.get(".", "test.txt");  
try (Formatter salida = new Formatter(ruta.toFile())) {  
    salida.format(...igual que printf...);  
} catch (IOException exc ) {  
    gestión de errores  
}
```

**Este garantiza que el Formatter se cierre**

# ¡Admite varios!

```
Path rutaOrigen = Paths.get(".", "origen.txt");
Path rutaDestino = Paths.get(".", "destino.txt");
try (Scanner origen = new Scanner(rutaOrigen);
     Formatter destino = new Formatter(rutaDestino.toFile())) {
    while (origen.hasNext()) {
        String linea = origen.nextLine();
        destino.format("%s%n", linea);
    }
} catch (IOException excepcion) {
    excepcion.printStackTrace();
}
```

**Llama close automáticamente**



**Cuando se  
usa uno u otro**



# **Cuando necesitamos cerrar el recurso si o si**

**El ingreso mejorado no funciona si se cierra automáticamente**



**¿Preguntas?**

---

# Excepciones II

---

# Con un breve repaso

**UNRN**

Universidad Nacional  
de Río Negro



# Ejemplo I

```
public class FailApp{  
  
    public static void main(String[] args) {  
        a();  
    }  
    static void a() {  
        b();  
    }  
    static void b() {  
        throw new RuntimeException();  
    }  
}
```

# Ejemplo II

```
public class FailApp{

    public static void main(String[] args) {
        a();
    }
    static void a() {
        try{
            b();
        } catch (RuntimeException exc){
            System.out.println("Ouch");
        }
    }
    static void b() {
        throw new RuntimeException();
    }
}
```

# ¿Como es el flujo del programa?

```
public class FailApp{

    public static void main(String[] args) {
        funcion();
    }

    static void funcion() {
        try {
            System.out.println("1");
            throw new RuntimeException();
            System.out.println("2");
        } catch (RuntimeException exc) {
            System.out.println("3");
        }
    }
}
```

# Usarlas de esta forma no es correcto



```
public class FailApp{

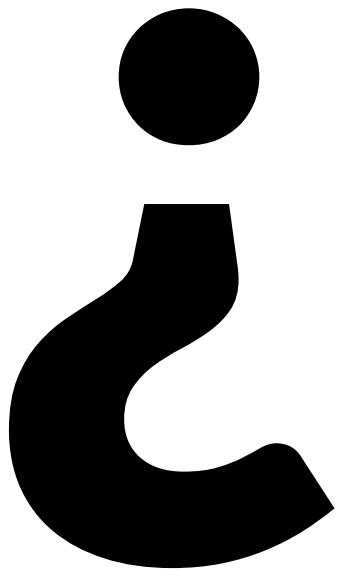
    public static void main(String[] args) {
        funcion();
    }

    static void funcion() {
        try {
            System.out.println("1");
            throw new RuntimeException();
            System.out.println("2");
        } catch (RuntimeException exc) {
            System.out.println("3");
        }
    }
}
```

**No atajar una  
excepción lanzada  
en el mismo bloque**



**¿Preguntas?**



**Qué implica que  
una excepción  
sea con tipo**



**¡Su código **no puede**  
dejar excepciones  
*con tipo sin atajar!***



# ¿Pero qué podemos hacer en el main?

```
public class FailApp{  
  
    public static void main(String[] args){  
        try{  
            metodoFail();  
        } catch (UnaExcepcion exc){  
            exc.printStackTrace();  
        }  
    }  
    static void metodoFail() {  
        throw new UnaExcepcion();  
    }  
}
```



**Probablemente** , de lo poco que  
podemos hacer

**Aparte de escribir un mensaje propio**

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

**Pero también,  
incluyendo**

# Relanzamiento 'suavizado'



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



Esto, **no** es hacerse cargo de la excepción

**No convertir  
excepciones con  
tipo a sin tipo**



**¿Preguntas?**

# **Una observación extra**

---

# printStackTrace

es un `print` con pasos adicionales  
Y como tales, *técnicamente* no van dentro de las funciones





**¿Preguntas?**

# TP4

## Archivos

**unrn.edu.ar**

