


Compiladores e Intérpretes

Guía para los Testers de JUnit de la Catedra

Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación
2023



Introducción

- En esta **guía** se presentará cómo **utilizar** las **herramientas de testing** de JUnit que utiliza la cátedra, como parte de la **evaluación** de los proyectos
- La guía está armada respecto a **IntelliJ IDEA** pero los pasos se pueden **adaptar a otros IDEs**

Introducción

- En esta **guía** se presentará cómo **utilizar** las **herramientas de testing** de JUnit que utiliza la cátedra, como parte de la **evaluación** de los proyectos
- La guía está armada respecto a **IntelliJ IDEA** pero los pasos se pueden **adaptar a otros IDEs**

Aclaración Importante: Las herramientas **no incluyen** los **casos de prueba** necesarios para **verificar adecuadamente** el compilador. El desarrollo de los casos es **responsabilidad del estudiante**. *Ademas, estas no son las únicas herramientas que utiliza la cátedra para la evaluación*

Contenido del paquete

- Este documento
- Dos testers:
 - *TesterDeCasosSinErrores.java* usado para **probar los casos de prueba sin errores**, y
 - *TesterDeCasosConErrores.java* usado para **probar los casos con errores**
- La **carpeta** *resources* con dos **subcarpetas**:
 - *sinErrores*: carpeta para los casos de prueba sin errores
 - *conErrores*: carpeta para los casos de prueba con errores

Contenido del paquete

- **Este documento**

- **Dos testers:**

- *TesterDeCaso*
de prueba sin errores

- *TesterDeCasosConErrores*
con errores

- La **carpeta** *resources* con dos **subcarpetas**:

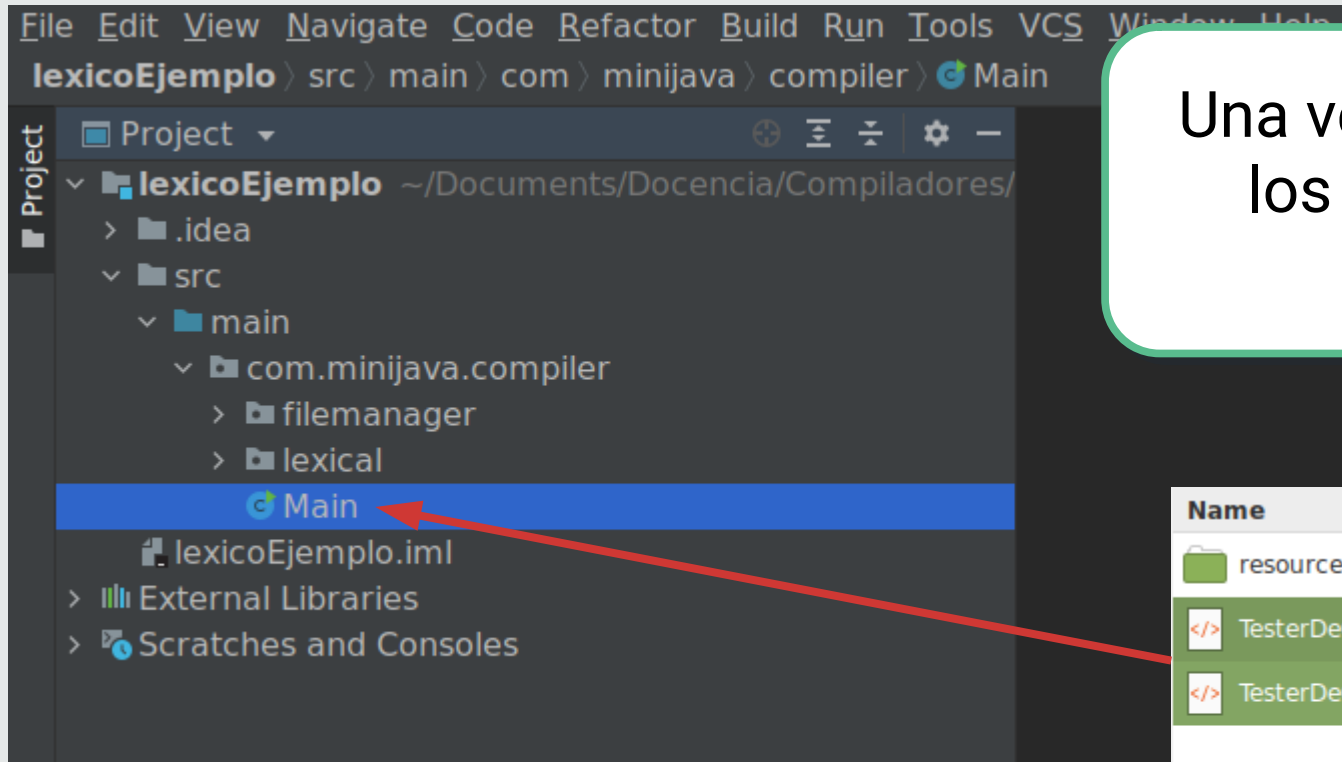
- *sinErrores*: carpeta para los casos de prueba sin errores

- *conErrores*: carpeta para los casos de prueba con errores

Estas carpetas contienen, a modo de ejemplo, dos casos cada una. Es **responsabilidad del estudiante desarrollar todos los casos** de prueba con y sin errores **necesarios para verificar** el adecuado funcionamiento de su compilador

carpeta para **probar los casos**

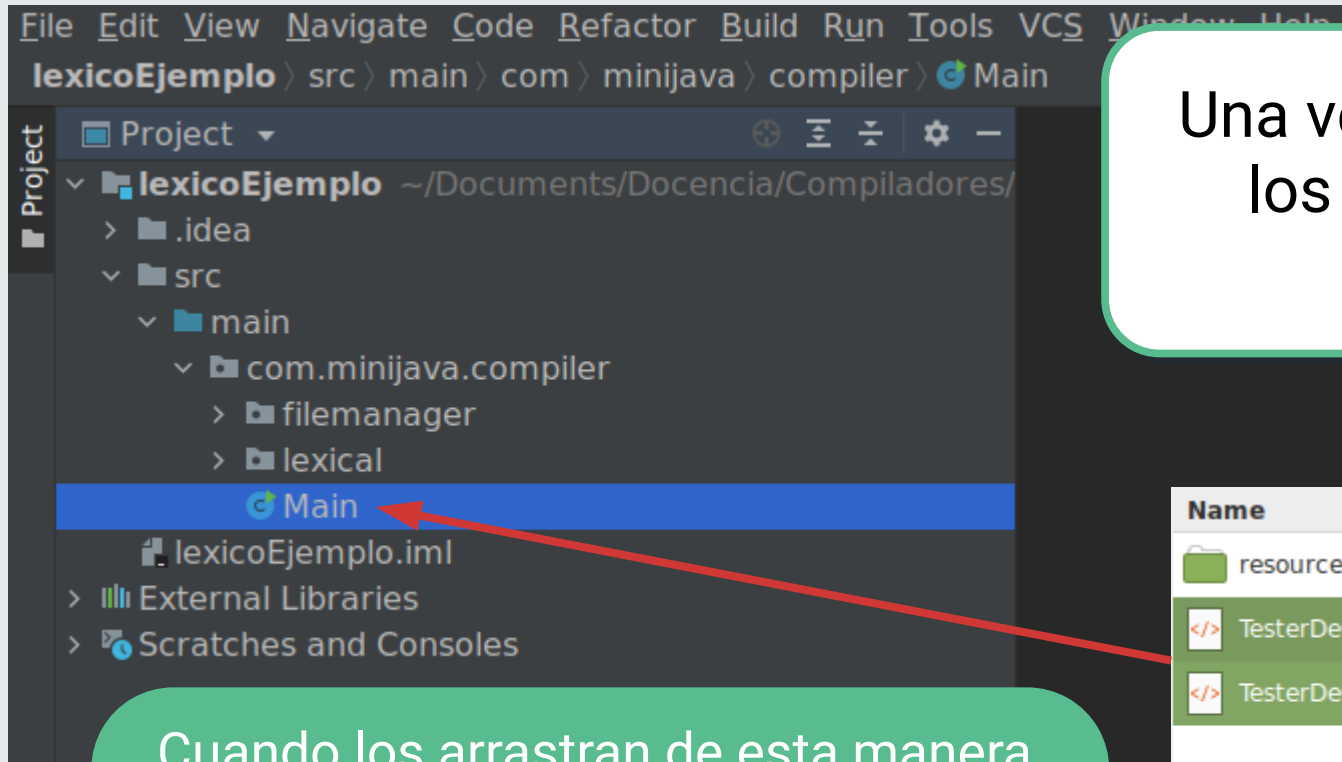
Agregando las Clases con los Testers



Una vez abierto el proyecto en el IDE arrastrar los ***dos testers*** a donde este la clase que contenga el método main

Name	Size	Type	Date Modified ^
resources	4,1 kB	folder	Today
TesterDeCasosConErrores.java	2,5 kB	Java source code	Today
TesterDeCasosSinErrores.java	2,1 kB	Java source code	Today

Agregando las Clases con los Testers



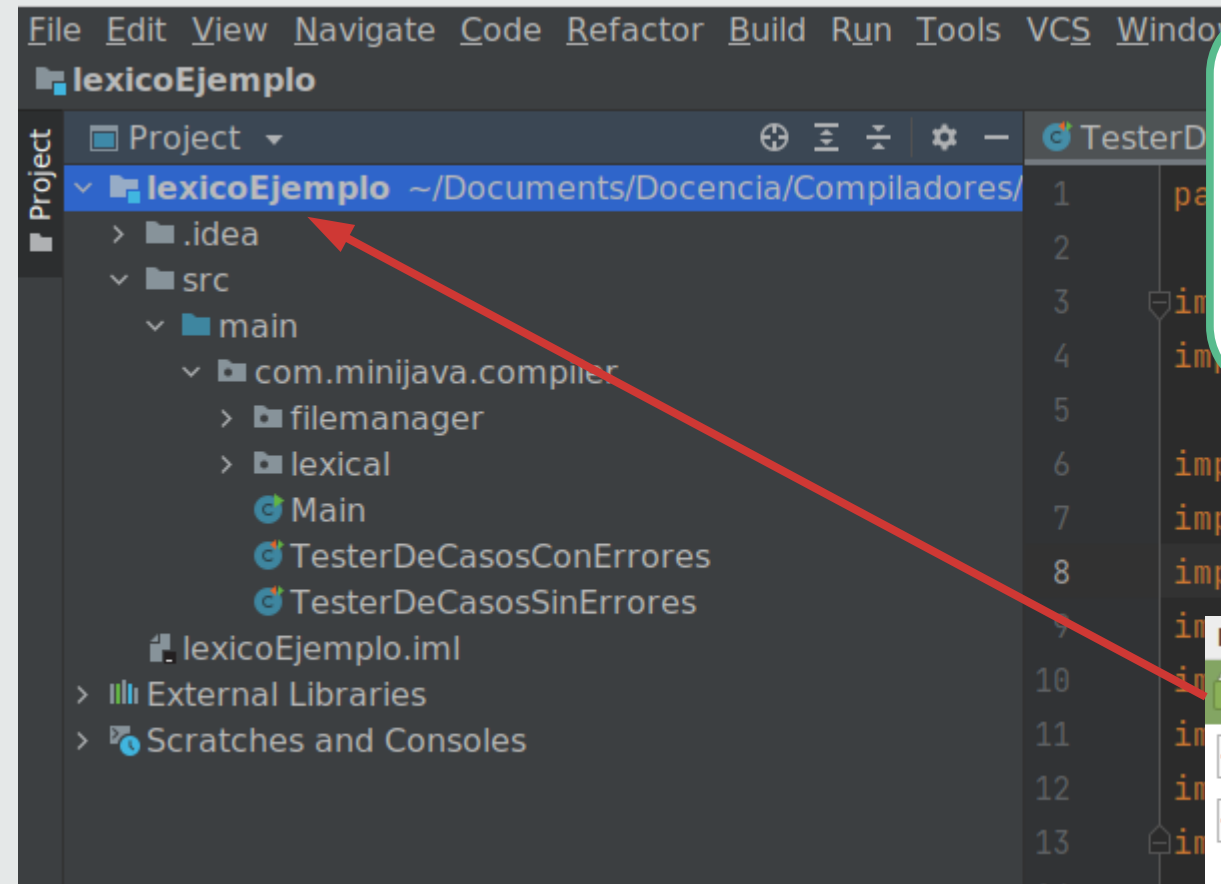
Una vez abierto el proyecto en el IDE arrastrar los **dos testers** a donde este la clase que contenga el método main



Cuando los arrastran de esta manera, IntelliJ les **agrega automáticamente el package**. Si no llegase a pasar esto (o estan usando otro IDE) lo tienen que agregar a mano en cada Tester

Name	Size	Type	Date Modified ^
resources	4,1 kB	folder	Today
TesterDeCasosConErrores.java	2,5 kB	Java source code	Today
TesterDeCasosSinErrores.java	2,1 kB	Java source code	Today

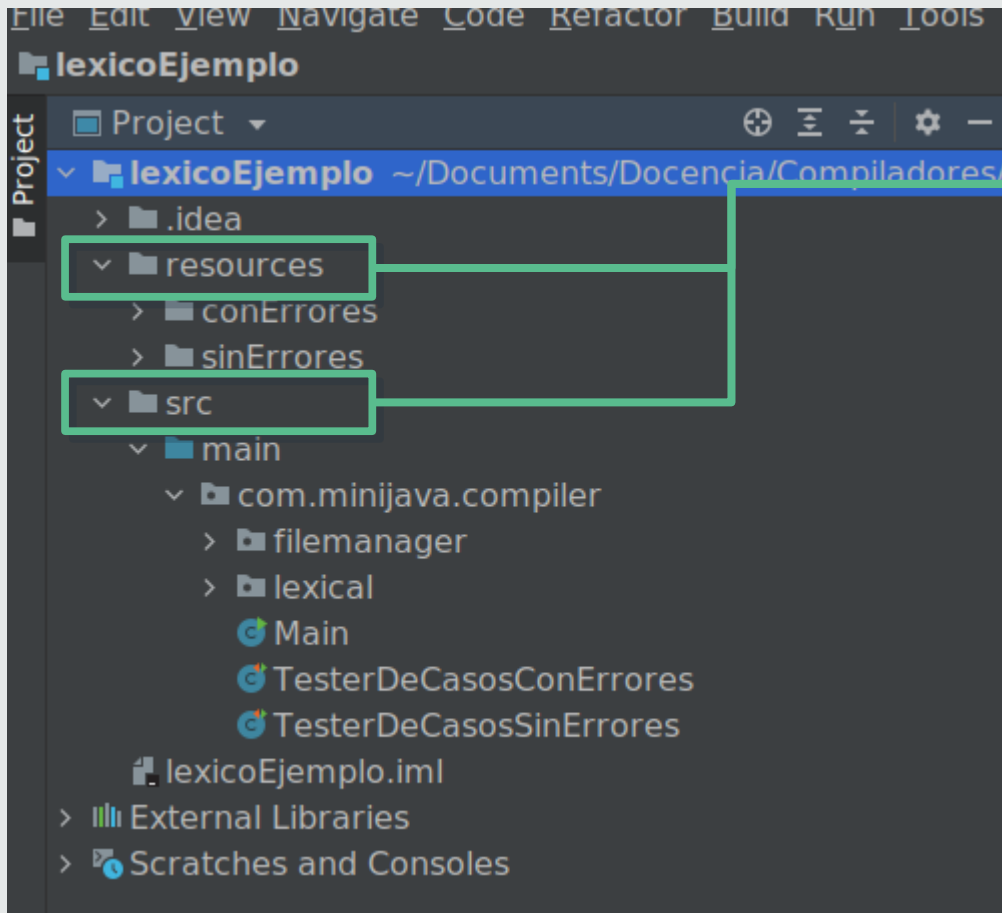
Agregando la carpeta de casos de prueba

También hay que llevar la carpeta **resources** con los casos de prueba a la carpeta raíz del proyecto, de forma que quede a la misma altura que la usual carpeta “src”



Name	Size	Type	Date Modified ^
resources	4,1 kB	folder	Today
 TesterDeCasosConErrores.java	2,5 kB	Java source code	Today
 TesterDeCasosSinErrores.java	2,1 kB	Java source code	Today

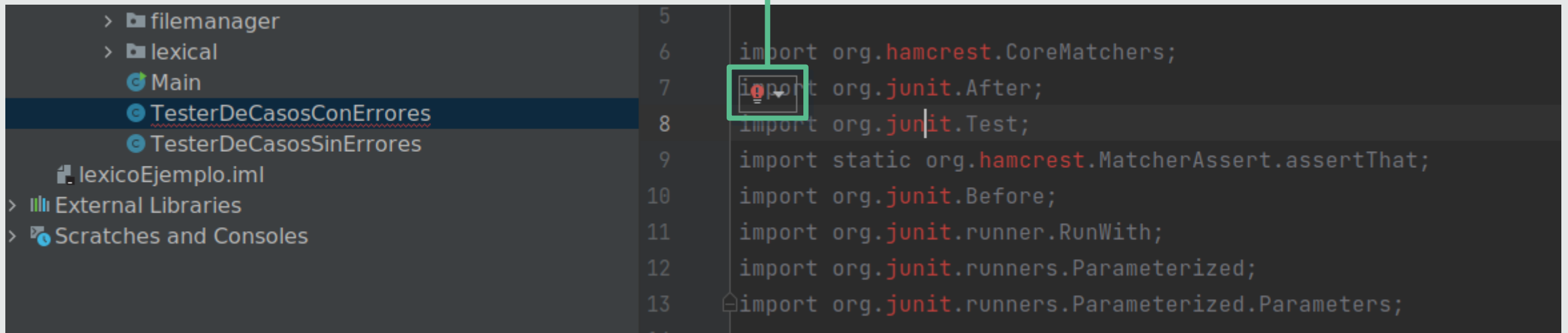
Agregando la carpeta de casos de prueba



También hay que llevar la carpeta **resources** con los casos de prueba a la carpeta raíz del proyecto, de forma que quede a la misma altura que la usual carpeta “src”

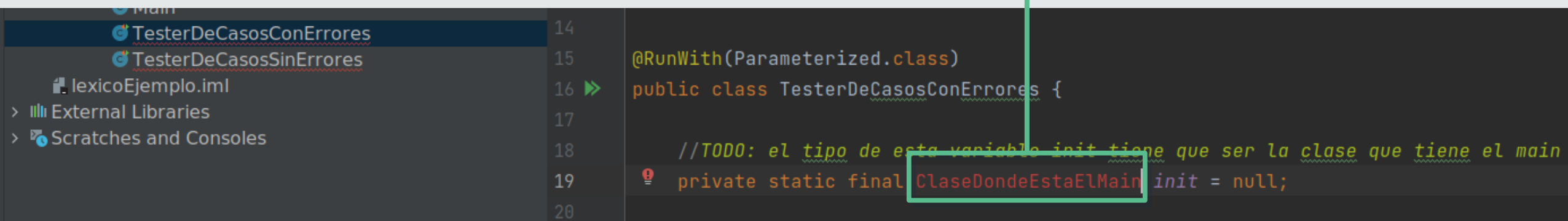
Dependencia con JUnit 4

Los Testers usan Junit 4, así que (al menos en IntelliJ) hay que agregar esa dependencia al classpath



Indicando la clase con el método main

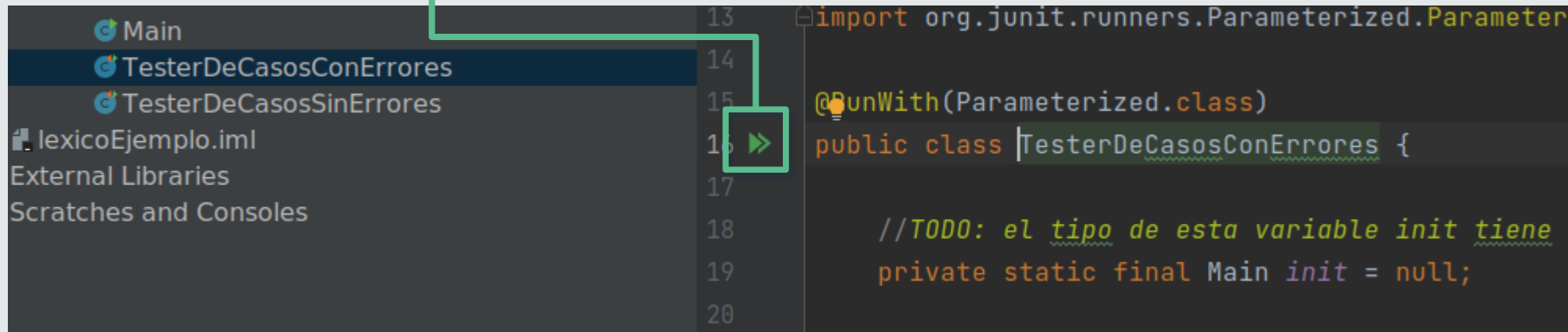
En ambos testers hay que **acomodar** la clase del atributo **init**, indicando la **clase** que tiene el método **main** del proyecto



```
14  
15  
16 >> public class TesterDeCasosConErrores {  
17  
18     //TODO: el tipo de esta variable init tiene que ser la clase que tiene el main  
19     private static final ClaseDondeEstaElMain init = null;  
20
```

Corriendo los Casos de Prueba

Para correr todos los tests de los casos con errores, hay que correr la clase `TesterDeCasosConErrores` (para los casos sin errores se usa la otra clase)

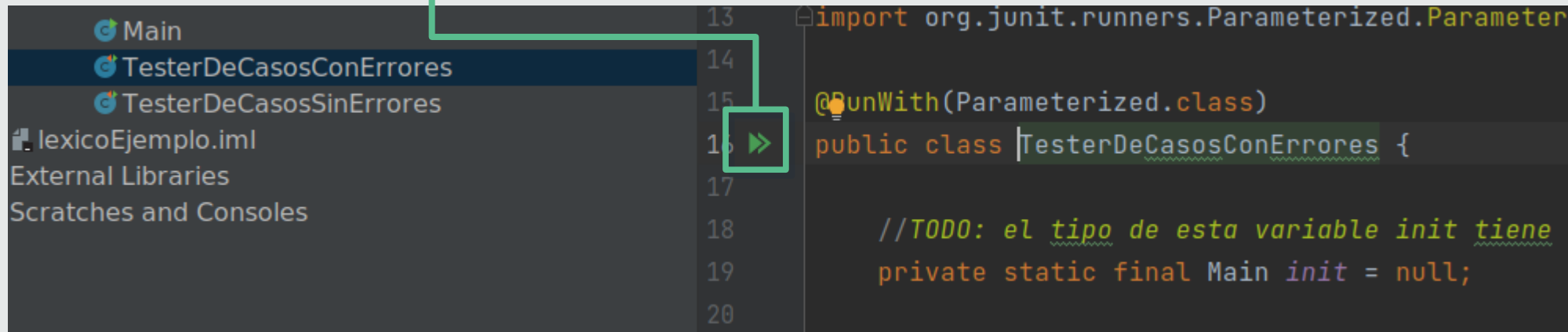


The screenshot shows an IDE interface. On the left, the 'Project' view displays a tree structure with the following items: 'Main', 'TesterDeCasosConErrores' (highlighted), 'TesterDeCasosSinErrores', 'lexicoEjemplo.iml', 'External Libraries', and 'Scratches and Consoles'. A green line connects the 'TesterDeCasosConErrores' item to the code editor on the right. The code editor shows the following Java code:

```
13 import org.junit.runners.Parameterized.Parameter
14
15 @RunWith(Parameterized.class)
16 public class TesterDeCasosConErrores {
17
18     //TODO: el tipo de esta variable init tiene
19     private static final Main init = null;
20
```

Corriendo los Casos de Prueba

Para correr todos los tests de los casos con errores, hay que correr la clase `TesterDeCasosConErrores` (para los casos sin errores se usa la otra clase)



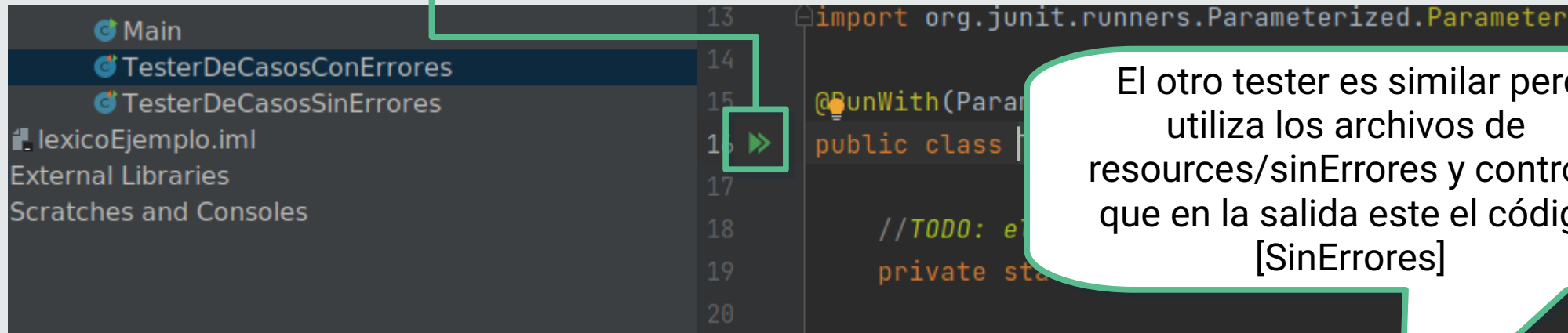
The screenshot shows an IDE with a project structure on the left and the source code of the `TesterDeCasosConErrores` class on the right. In the project structure, the `TesterDeCasosConErrores` class is selected, and a green box highlights the run button (a green play icon) next to it. The source code on the right shows the following:

```
13 import org.junit.runners.Parameterized.Parameter
14
15 @RunWith(Parameterized.class)
16 public class TesterDeCasosConErrores {
17
18     //TODO: el tipo de esta variable init tiene
19     private static final Main init = null;
20 }
```

Este **tester** invoca al main con cada archivo de la carpeta `resources/conErrores` y contrasta que en la salida se reporte el código de error adecuado (en lo que sigue veremos como se especifica)

Corriendo los Casos de Prueba

Para correr todos los tests de los casos con errores, hay que correr la clase `TesterDeCasosConErrores` (para los casos sin errores se usa la otra clase)



The screenshot shows an IDE with a project structure on the left and code on the right. The project structure includes 'Main', 'TesterDeCasosConErrores', and 'TesterDeCasosSinErrores'. The code on the right is for 'TesterDeCasosConErrores' and includes the following lines:

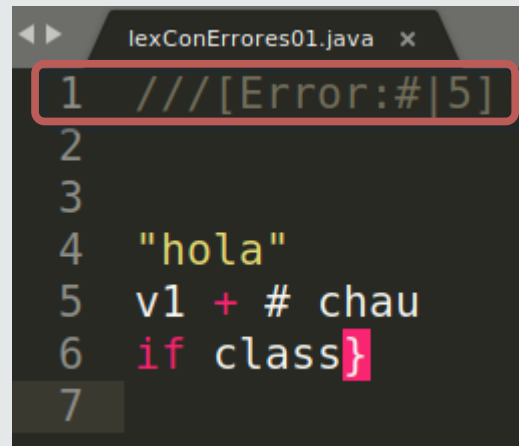
```
13 import org.junit.runners.Parameterized.Parameter
14
15 @RunWith(Parameterized.class)
16 public class TesterDeCasosConErrores {
17
18     //TODO: e
19     private sta
20
```

El otro tester es similar pero utiliza los archivos de `resources/sinErrores` y controla que en la salida este el código `[SinErrores]`

Este **tester** invoca al main con cada archivo de la carpeta `resources/conErrores` y contrasta que en la salida se reporte el código de error adecuado (en lo que sigue veremos como se especifica)

Formato de los casos de prueba con errores

- En los **casos** de prueba **con errores** la primer linea debe ser el código de error que esperado antecedido por “///”



```
lexConErrores01.java x
1 ///[Error:#|5]
2
3
4 "hola"
5 v1 + # chau
6 if class}
7
```

- Ese es el **código** que utilizará el tester para ver si la **salida** del compilador es la **esperada**

Formato de los casos de prueba sin errores

- En los **casos** de prueba **con errores** la primer linea debe ser el código [SinErrores] que esperado antecedido por “///”
- Ese es el **código** que utilizará el tester para ver si la **salida** del compilador es la **esperada**

Formato de los casos de prueba

- Recomendamos ver los **casos de prueba** (con y sin errores) **provistos por la cátedra** como ejemplos
 - Ahí van a poder ver **ejemplificaciones** de como son estos formatos
- Ante cualquier duda **consultanos!**