

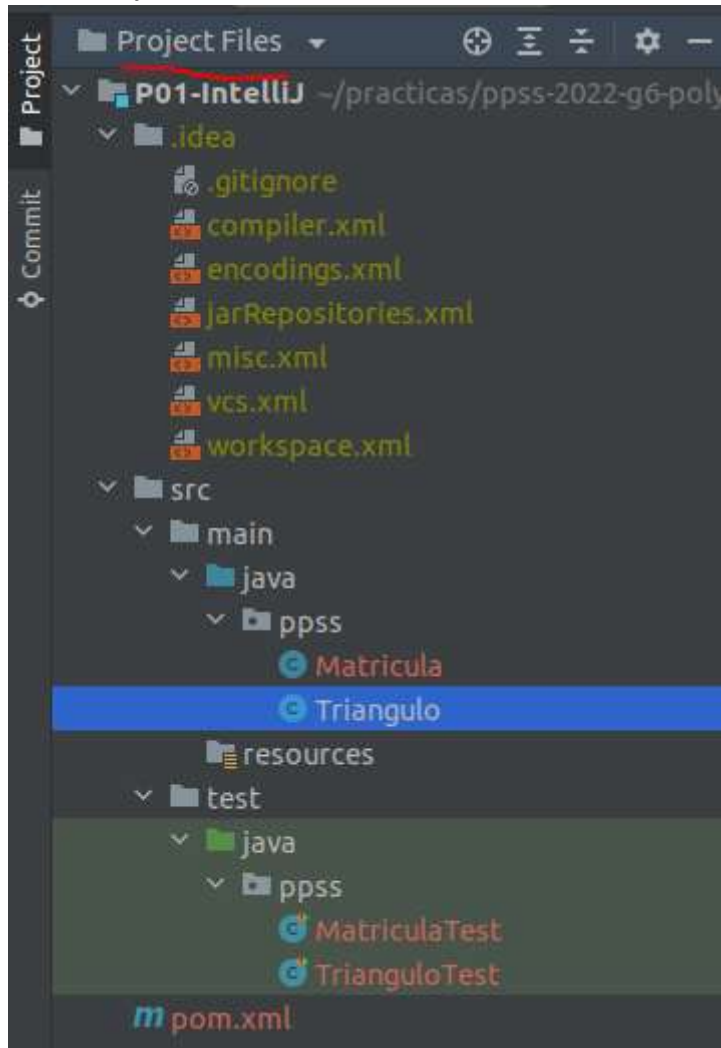
Practica 01A

Nikita Polyanskiy

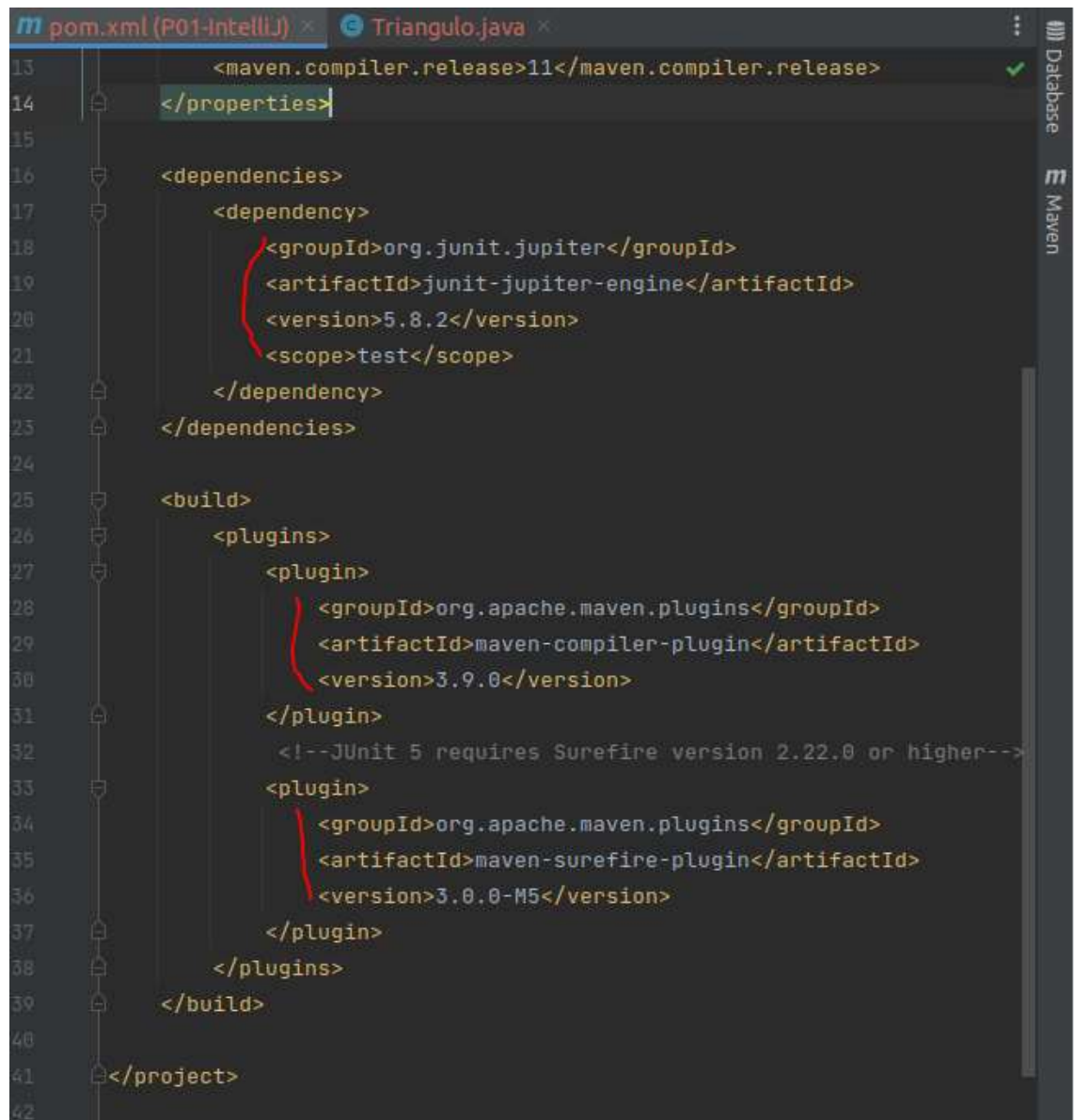
Y4441167L

Ejercicio 1:

- A) Para ver la estructura del directorio (las carpetas y archivos físicos), hay que visualizar como "Project Files":



- B) Podemos ver 3 artefactos, 1 usado en Dependencies, y 2 en Build:



```
13      <maven.compiler.release>11</maven.compiler.release>
14    </properties>
15
16    <dependencies>
17      <dependency>
18        <groupId>org.junit.jupiter</groupId>
19        <artifactId>junit-jupiter-engine</artifactId>
20        <version>5.8.2</version>
21        <scope>test</scope>
22      </dependency>
23    </dependencies>
24
25    <build>
26      <plugins>
27        <plugin>
28          <groupId>org.apache.maven.plugins</groupId>
29          <artifactId>maven-compiler-plugin</artifactId>
30          <version>3.9.0</version>
31        </plugin>
32        <!--JUnit 5 requires Surefire version 2.22.0 or higher-->
33        <plugin>
34          <groupId>org.apache.maven.plugins</groupId>
35          <artifactId>maven-surefire-plugin</artifactId>
36          <version>3.0.0-M5</version>
37        </plugin>
38      </plugins>
39    </build>
40
41  </project>
42
```

- C) La clase Triangulo.java nos proporciona un conjunto especificado de valores del triángulo.

```

4 public class Triangulo {
5
6     public String tipo_triangulo(int a, int b, int c) {
7         String result=null;
8
9         if ((a <1) && (a > 200)) {
10             return "Valor a sobrepasa el rango permitido";
11         }
12         if ((b <1) && (b > 200)) {
13             return "Valor b sobrepasa el rango permitido";
14         }
15         if ((c <1) && (c > 200)) {
16             return "Valor c sobrepasa el rango permitido";
17         }
18
19         if (a >= (b+c)) {
20             return "No es un triangulo";
21         }
22
23         if (b >= (a+c)) {
24             return "No es un triangulo";
25         }
26
27         if (c >= (a+b)) {
28             return "No es un triangulo";
29         }
30
31         if ((a != b) && (a!= c) && (b!= c)) {
32             result = "Escaleno";
33         } else {
34             if (a==b) {
35                 if (a==c) {

```

D)

Id	Dato1 (a)	Dato2 (b)	Dato3 (c)	Resultado esperado
C1	1	1	1	Equilátero
C2	1	1	11	No es un triangulo
C3	1	2	0	Valor c sobrepasa el rango permitido*
C4	14	10	10	Isósceles

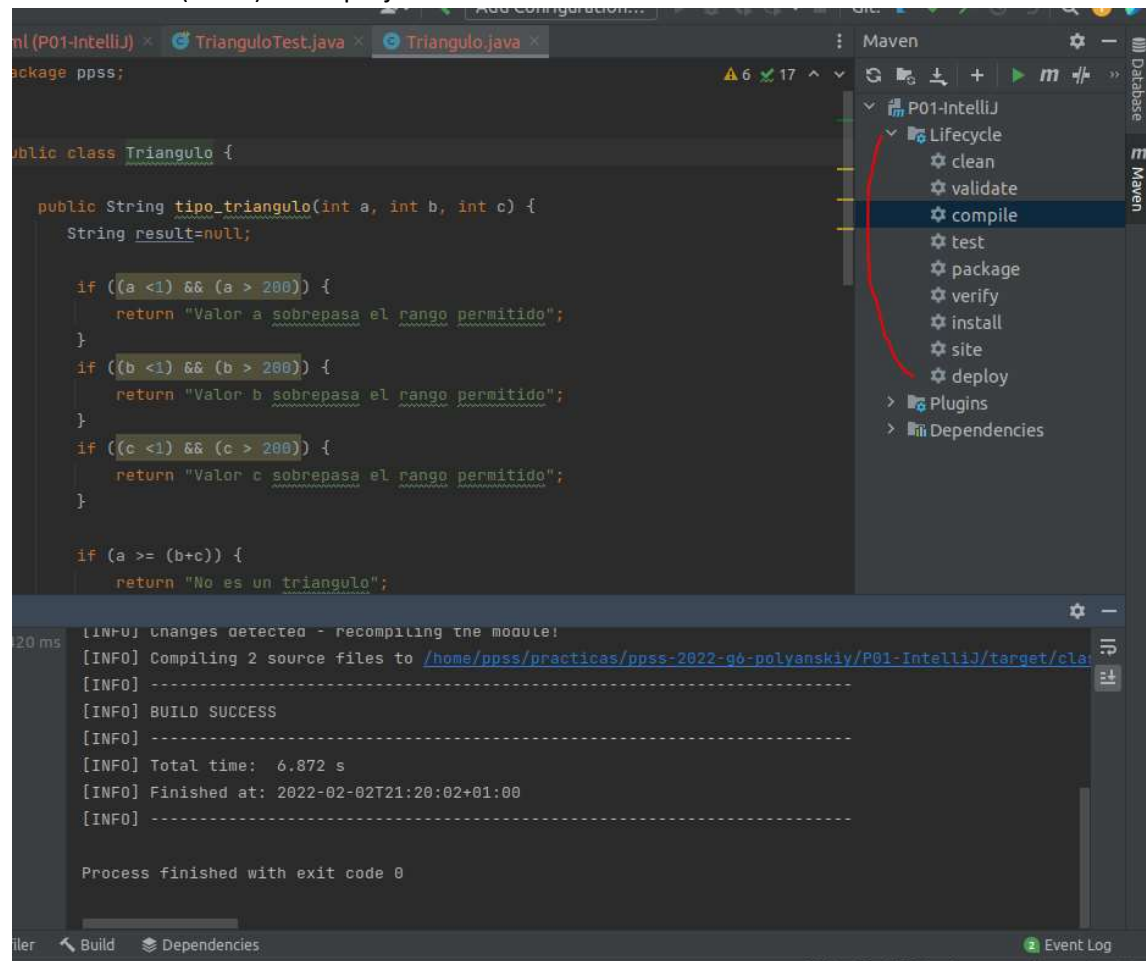
*arreglado después del ejercicio 2C

Algoritmo:

0. Función aparte para cada caso de prueba, antes del nombre de la función se pone `@Test`.
1. Asignar valores de prueba a las variables a utilizar.
2. Asignar el resultado esperado a otra variable.
3. Ejecutar el programa utilizando las variables de prueba, y guardar el resultado.
4. Comparar el resultado obtenido con el esperado, utilizando `assertEquals(esperado,real)`, este será el resultado de nuestro test.

Ejercicio 2:

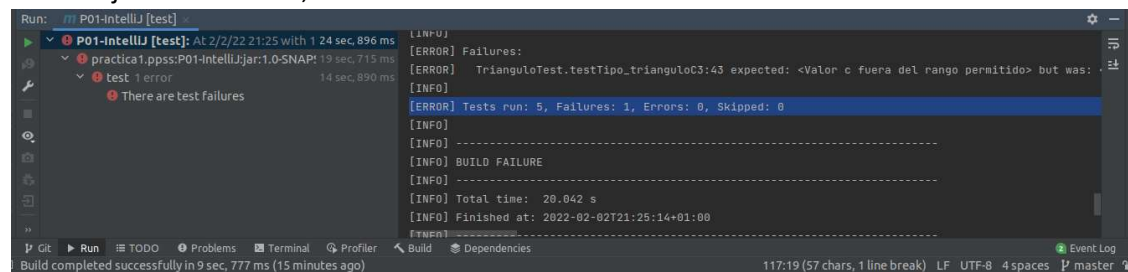
Ciclos de vida (Fases) de un proyecto Maven:



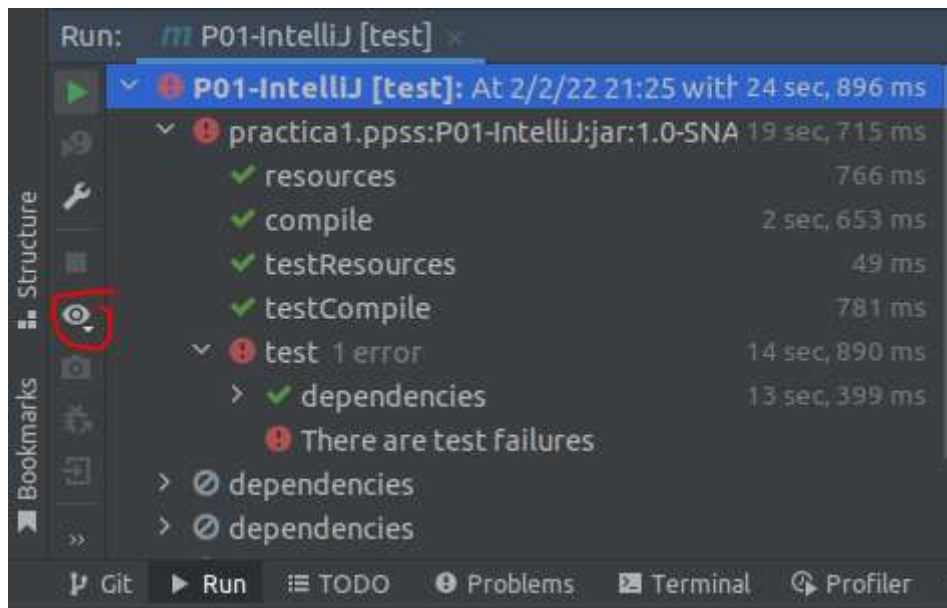
- A) El directorio Target se utiliza como directorio de salida para Maven, el contenido de todas las fuentes se guarda ahí, además de los archivos necesarios para los artefactos.



B) Se han ejecutado 5 tests, uno de ellos ha fallado:



Apretando el botón con el ojo podremos ver los tests que se han ejecutado con éxito:



- C) El error se encuentra en la línea 40, en ningún caso podremos obtener el resultado esperado “Valor c fuera del rango permitido”, el valor correcto es “Valor c sobrepasa el rango permitido”, la cual tampoco hemos obtenido (el valor real obtenido es “No es un triángulo”), por lo que hay que arreglar las condiciones del if:

```

35     @Test
36     public void testTipo_trianguloC3() {
37         a = 1;
38         b = 2;
39         c = 0;
40         resultadoEsperado = "Valor c fuera del rango permitido";
41         tri = new Triangulo();
42         resultadoReal = tri.tipo_triangulo(a,b,c);
43         assertEquals(resultadoEsperado, resultadoReal);
44     }

```

```

public String tipo_triangulo(int a, int b, int c) {
    String result=null;

    if ((a < 1) && (a > 200)) {
        return "Valor a sobrepasa el rango permitido";
    }
    if ((b < 1) && (b > 200)) {
        return "Valor b sobrepasa el rango permitido";
    }
    if ((c < 1) && (c > 200)) {
        return "Valor c sobrepasa el rango permitido";
    }
}

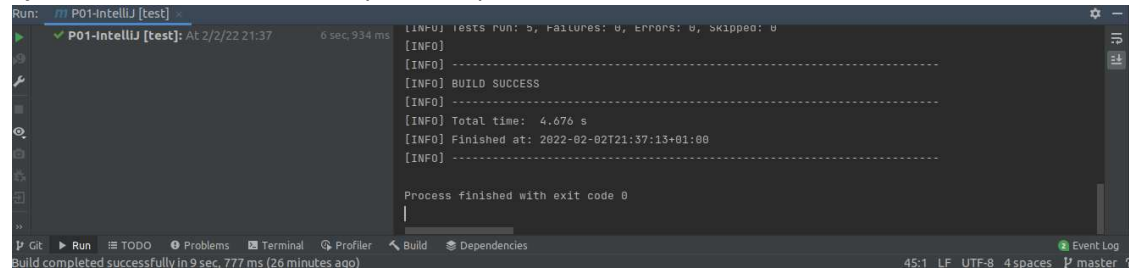
```


En la captura de arriba no es correcto poner &&, debería ser ||:

```
public String tipo_triangulo(int a, int b, int c) {
    String result=null;

    if ((a <1) || (a > 200)) {
        return "Valor a sobrepasa el rango permitido";
    }
    if ((b <1) || (b > 200)) {
        return "Valor b sobrepasa el rango permitido";
    }
    if ((c <1) || (c > 200)) {
        return "Valor c sobrepasa el rango permitido";
    }
}
```

Ejecutamos los tests otra vez y vemos que todo esta correcto:



- D) El caso C5 no aportaría ningún valor, ya que se estaría haciendo el mismo recorrido que la ejecución C1

```
57
58     @Test
59     public void testTipo_trianguloC5() {
60         a = 7;
61         b = 7;
62         c = 7;
63         resultadoEsperado = "Equilatero";
64         tri= new Triangulo();
65         resultadoReal = tri.tipo_triangulo(a,b,c);
66         assertEquals(resultadoEsperado, resultadoReal);
67     }
68 }
69
```

Los siguientes son posibles tests que si aportarían valor, al realizar recorridos diferentes que no se han probado en tests anteriores:

```
78 ▶ public void testTipo_trianguloC6() {
79
80     a = 2;
81     b = 4;
82     c = 5;
83     resultadoEsperado = "Escaleno";
84     tri = new Triangulo();
85     resultadoReal = tri.tipo_triangulo(a,b,c);
86     assertEquals(resultadoEsperado, resultadoReal);
87 }
88
89 @Test
90 public void testTipo_trianguloC7() {
91
92     a = 1;
93     b = 0;
94     c = 2;
95     resultadoEsperado = "Valor b sobrepasa el rango permitido";
96     tri = new Triangulo();
97     resultadoReal = tri.tipo_triangulo(a, b, c);
98     assertEquals(resultadoEsperado, resultadoReal);
99 }
100
101 @Test
102 public void testTipo_trianguloC8() {
103
104     a = 0;
105     b = 2;
106     c = 1;
107     resultadoEsperado = "Valor a sobrepasa el rango permitido";
108     tri = new Triangulo();
109     resultadoReal = tri.tipo_triangulo(a, b, c);
110     assertEquals(resultadoEsperado, resultadoReal);
111 }
112 }
```

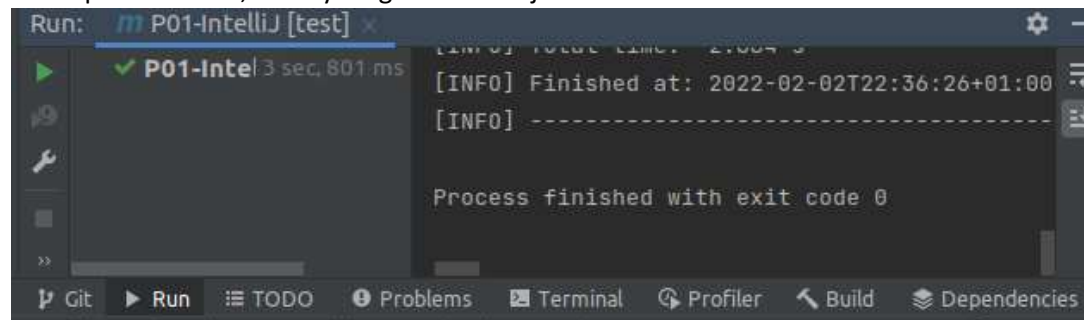
Run: P01-IntelliJ [test] - 6 sec, 318 ms [INFO] Finished at: 2022-02-02T21:46:31+01:00 [INFO] ----- Process finished with exit code 0

Ejercicio 3:

A) Son necesarios 7 test para comprobar todos los posibles recorridos del programa.

Id	edad	familiaNumerosa	repetidor	Resultado esperado
C1	23	True	True	250.00
C2	23	false	true	2000.00
C3	23	false	false	500.00
C4	28	true	false	250.00
C5	28	false	false	500.00
C6	55	false	false	400.00
C7	70	false	false	250.00

B) La implementación de los tests es la misma que para el test C1, únicamente cambiando los datos de entrada y el resultado esperado, y como podemos ver, no hay ningún fallo al ejecutar nuestros tests:



Ejercicio 4:

A) Contenido de target al ejecutar solamente hasta la fase “test”:



Contenido de target al ejecutar la fase package:

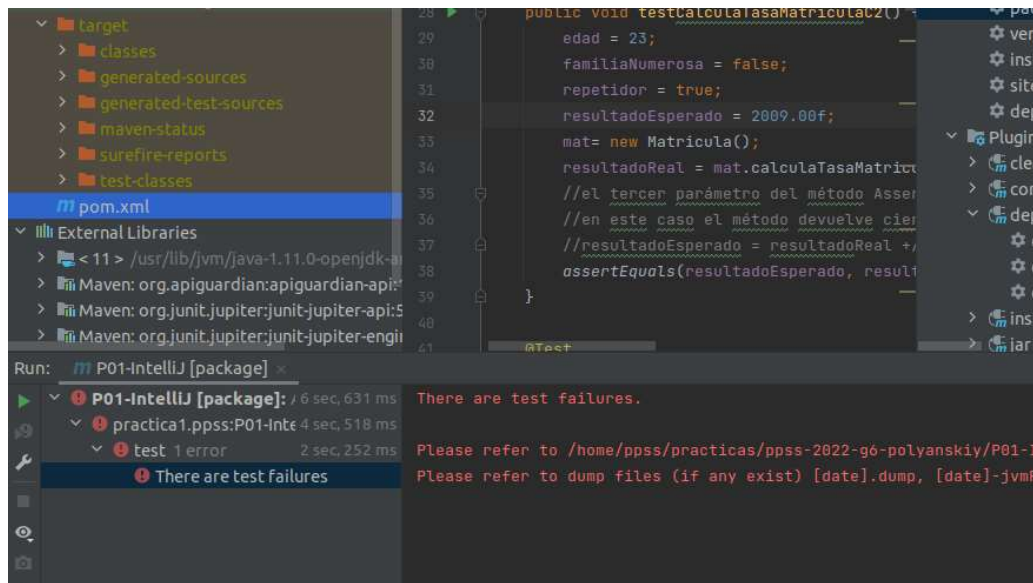


Se ha generado el archivo P01-IntelliJ-1.0-SNAPSHOT.jar

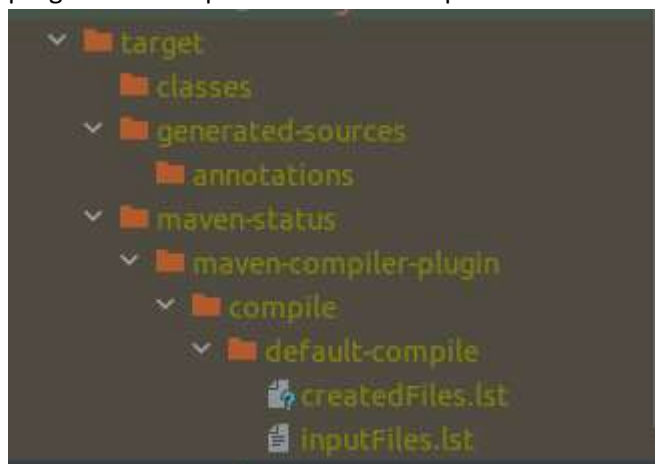
```
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ P01-IntelliJ ---  
[INFO] Building jar: /home/ppss/practicas/ppss-2022-g6-polyanskiy/P01-IntelliJ/target/P01-IntelliJ-1.0-SNAPSHOT.jar
```

Al ejecutar la fase package se ha ejecutado el goal Maven-jar-plugin:2.4:jar

Si hay un fallo en los tests, el fichero .jar no se genera, es decir el programa no continua después de la fase de test:



Al quitar un ; del fichero Matricula.java, se ha causado un error de compilación, por lo que no se ha compilado el proyecto, es decir el programa no ha pasado la fase “compile”



- B) La ubicación de los archivos generados por el proyecto se ubican para este caso en `.m2/repository/practica1/ppss/P01-IntelliJ/`. Además de los archivos para las demás librerías y dependencias estarán directamente en `.m2/repository/`

