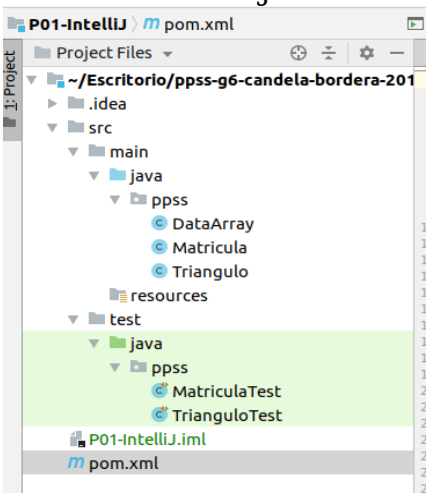
P01

Ejercicio 1

A) Anota la estructura de directorios del proyecto. La información del proyecto puede mostrarse desde diferentes "perspectivas". Si quieres ver exactamente las carpetas físicas del disco duro debes mostrar la vista "Project Files", en lugar de “Project", que es la que tendrás por defecto. La estructura de directorios es la misma en CUALQUIER proyecto Maven. Es importante conocerla, ya que el proceso de construcción que realiza Maven asume que determinados artefactos están situados en determinados directorios. Por ejemplo, si el código fuente de las pruebas lo implementásemos en el directorio /src/main/java, no se ejecutarían dichos tests aunque lanzásemos la fase “test” de Maven:

Con la vista Project Files:

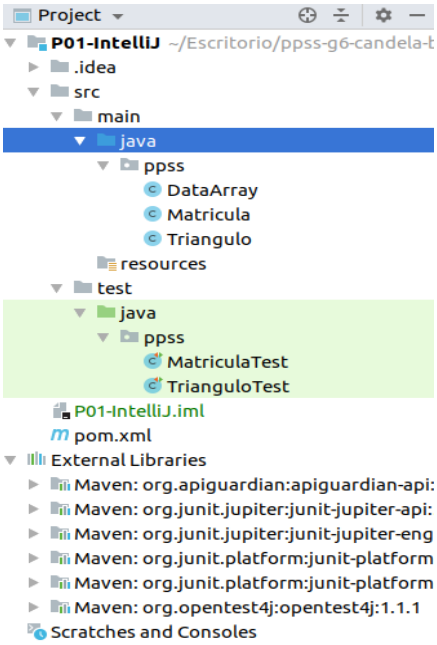


Con la vi

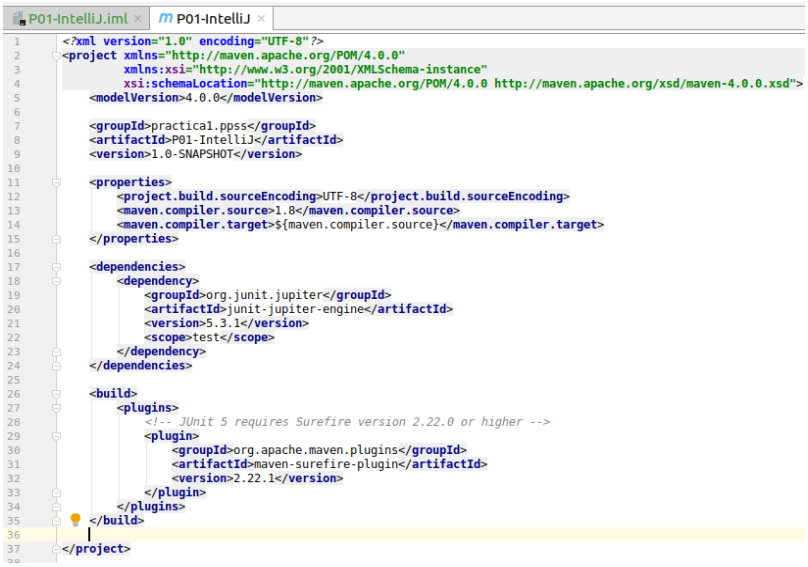
```
package ppss;

public class Matricula {
    public float calculaTasaMatricula(int edad, boolean familiaNumerosa,
                                      boolean repetidor) {
        float tasa = 500.00f;

        if ((edad < 25) && (!familiaNumerosa) && (repetidor)) {
            tasa = tasa + 1500.00f;
        } else {

            if ((edad > 50) && (edad < 65)) {
                tasa = tasa - 100.00f;
            } else if((familiaNumerosa) || (edad >= 65)) {
                tasa = tasa / 2;
            }
        }
        return tasa;
    }
}
```

sta

Project



B) Muestra en el editor la configuración de nuestro proceso de construcción (fichero pom.xml):



Verás que el fichero xml contiene información sobre: las coordenadas, propiedades, dependencias
y sobre la construcción del proyecto (etiqueta <build>)
Maven . Nuestro pom define las coordenadas de nuestro proyecto, y además hace referencia a dos artefactos: uno en la sección de dependencias y otro en la sección <build>, indica cuáles son sus coordenadas: tanto las del proyecto, como la de los dos artefactos referenciados en él.

Proyecto:

practica1.ppss:P01-IntelliJ:1.0-Snapshot

Artefacto1, dependencia jUnit:

org.junit.jupiter:junit-jupiter-engine:5.3.1
(La etiqueta <scope> no se incluye en las coordenadas)

Artefacto2:
org.apache.maven.plugins:maven-surefire-plugin:2.22.1

¿Por qué nuestro proyecto tiene definidas unas coordenadas?
Las coordenadas se usan para identificar exactamente la ruta de cada fichero en el repositorio maven

¿Por qué los artefactos de nuestro pom están en secciones (etiquetas) diferentes y qué tipo de ficheros son?
Para diferenciar para qué se usan. Por ejemplo plugin es un conjunto de goals, en cambio la sección <dependencies> contiene librerías externas
Son ficheros .jar principalmente, aunque también pueden ser de tipo .war

La etiqueta <properties> se utiliza para definir y/o asignar/modificar valores a determinadas “variables” de nuestra configuración de nuestro proceso de construcción. Podemos usar propiedades ya predefinidas (por ejemplo la propiedad “project.build.sourceEncoding”), o podemos definir cualquier propiedad que nos interese. A partir de Maven 3 es OBLIGATORIO especificar en el pom.xml un valor para la propiedad “project.build.sourceEncoding”, por lo que esta línea aparecerá en todos los Sicheros pom.xml de nuestros proyectos.
Observa que hemos especificado el valor "test" para la etiqueta <scope> en uno de los artefactos.
Dicho valor indica que el artefacto en cuestión sólo se necesita durante la compilación de los tests).
Si esta etiqueta se omite, su valor por defecto es "compile" y significa que el artefacto es necesario para compilar los fuentes del proyecto.
El pom.xml de nuestra construcción incluye el plugin maven-sure8ire-plugin. Este plugin ya está incluido por defecto cuando usamos el empaquetado jar. Lo que ocurre es que la versión que se incluye por defecto es anterior a la 2.20.0 (que es la versión mínima requerida para poder compilar nuestros tests con junit5). Averigua qué versión viene incluida por defecto comentando el plugin en el pom, y consultando la información del plugin desde la ventana Maven Projects.

D)La clase TrianguloTest contiene la implementación de cuatro casos de prueba (los cuatro métodos anotados con @Test) asociados a la especificación del apartado anterior. ¿Cuáles son exactamente? Identifícalos como C1, C2, C3, y C4, y muéstralos en una tabla con cuatro filas con la siguiente información:

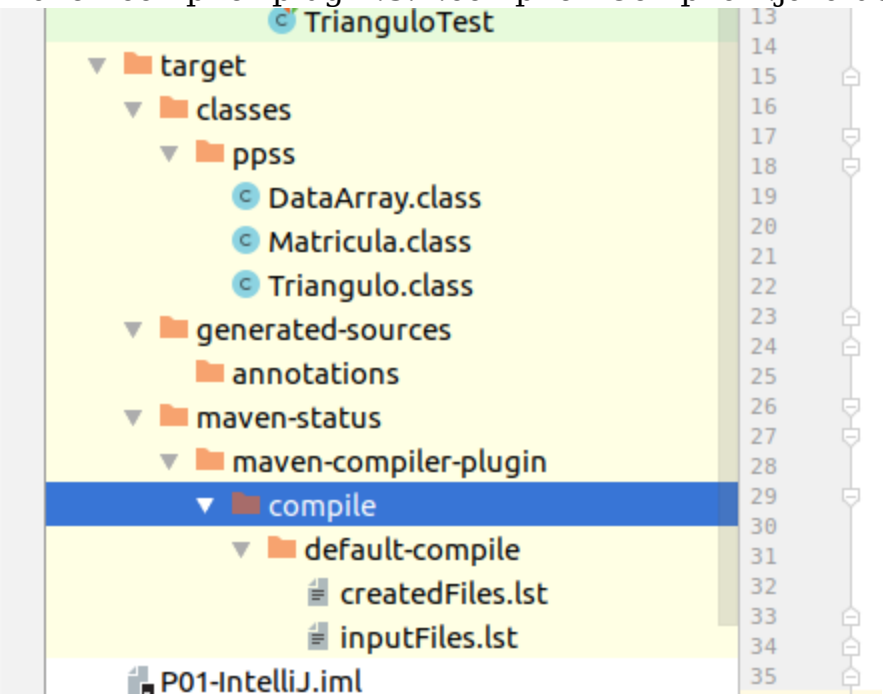
Identificador del caso de prueba	Dato de entrada 1	Dato de entrada 2	Dato de entrada 3	Resultado Esperado	Resultado Real
C1	80	80	80	Equilatero	
C2	100	100	50	Isósceles	
C3	130	90	140	Escaleno	

C4	250	250	50	Valor a fuera del rango permitido	

E)Observa la implementación de cada test y verás que todos ellos siguen la misma lógica de programa. Anota el algoritmo que refleja dicha lógica. Recuérdalo porque lo utilizaremos también en sesiones posteriores.

1. Asignar valores de entrada
2. Asignar valore esperado
3. Asignar valor real la salida del método testeado con la entrada
4. Comprobar que valor esperado y valor real son iguales

Ejercicio 2
A)Después de realizar el mvn compile, se genera el siguiente directorio, llamado Target:
Las goals ejecutadas son:
(plugin:versión:goal)
maven-resources-plugin:2.6:resources - Copia *.* de src/main/resources en target
maven-compiler-plugin:3.1:compile - Compila *.java de /src/main/java



Al ejecutar la fase clean, se borra el directorio Target junto con su contendo, las goals ejecutadas son:
maven-clean-plugin:2.5:clean

B)He observado lo que propone el ejercicio

C)Se ha modificado esta parte de código ya que provocaban que el resultado esperado en C3, no fuese el correcto

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

El if tiene que tener un OR en vez de un AND, y el texto mostrado por pantalla no era el que se esperaba en los tests
D)Un test adicional C5 con datos de entrada a=7,b=7 y c=7 sería redundante ya que se comportaría de la misma forma que C1, generando el mismo resultado, por lo que no es aconsejable añadirlo.
En cambio los tests C2 y C3 si que son necesarios ya que se encargan de obtener un valor totalmente diferente al de C1.Propongo los siguientes tests, ya que el primero se encarga de
comprobar si un valor se pasa del valor máximo permitido, y el segundo se comprueba si un triangulo es Escaleno, que anteriormente no se había probado. Ambos casos aportan valor, y no se han probado anteriormente.

```
@Test
public void testTipo_trianguloC5() {
    a = 250;
    b = 10;
    c = 10;
    resultadoEsperado = "Valor a fuera del rango permitido";
    resultadoReal = tri.tipo_triangulo(a,b,c);
    assertEquals(resultadoEsperado, resultadoReal);
}

@Test
public void testTipo_trianguloC6() {
    a = 3;
    b = 5;
    c = 7;
    resultadoEsperado = "Escaleno";
    resultadoReal = tri.tipo_triangulo(a,b,c);
    assertEquals(resultadoEsperado, resultadoReal);
}
```

Ejercicio 3

A) Tabla

ID	EDAD	FAM NUM	REPETIDOR	VAL.ESPERADO	VAL. REAL
C1	23	SI	SI	250	
C2	23	NO	SI	2000	
C3	23	NO	NO	500	
C4	65	SI	SI	250	
C5	51	SI	SI	400	
C6	35	NO	NO	500	

B) Tests implementados

ID	EDAD	FAM NUM	REPETIDOR	VAL.ESPERADO	VAL. REAL
C1	23	SI	SI	250	250
C2	23	NO	SI	2000	2000
C3	23	NO	NO	500	500
C4	65	SI	SI	250	250
C5	51	SI	SI	400	400**
C6	35	NO	NO	500	500

**Fallo encontrado y depurado (devolvía 150), al tener familia numerosa en los dos if, se dividía por 2 y se restaba 100. Se ha modificado el orden de las condiciones para que no entrara dos veces

```
package ppss;

public class Matricula {
    public float calculaTasaMatricula(int edad, boolean familiaNumerosa,
                                      boolean repetidor) {
        float tasa = 500.00f;

        if ((edad < 25) && (!familiaNumerosa) && (repetidor)) {
            tasa = tasa + 1500.00f;
        } else {

            if ((edad > 50) && (edad < 65)) {
                tasa = tasa - 100.00f;
            } else if((familiaNumerosa) || (edad >= 65)) {
                tasa = tasa / 2;
            }
        }
        return tasa;
    }
}
```

C) Me he basado en la tabla que se ha especificado en el enunciado, la mayoría se han basado en esto, aunque también hay el que ha añadido más casos de prueba pero acaban siendo redundantes porque el resultado es el mismo, aunque se varíen los datos de entrada, ejemplo:

probar un caso con edad < 25 con familia numerosa y otro con edad 25..50 con familia numerosa, la tasa será igual para ambos casos

Ejercicio 4

A) mvn package realiza las siguientes acciones:

1º Goal resources
2º Goal compile
3º Goal testResources
4º Goal testCompile
5º Goal test
6º Goal jar

Si modifco un test para que falla, se bloquea la ejecución en test, y no llega a ejecutarse el empaquetado jar

Si modifico una clase java, eliminando un punto y coma, se detiene en la goal compile, ya que no es capaz de compilar el código

En ambos casos se detiene la construcción de maven ya que la fase anterior no se ha finalizado correctamente.
(Esto lo he intuido, preguntar al profesor)

B) La fase install, copia el artefacto generado en la fase anterior en el repositorio local.
Para entrar en la carpeta m2 hay que teclear en la consola: cd ~/.m2
La ruta del artefacto es la siguiente:
~/.m2/repository/practica1
Se encuentra en esta ruta ya que es la carpeta que usa Maven para guardar los artefactos.
jUnit se encuentra en la siguiente carpeta:
~/.m2/repository/junit

Ejercicio 5

A)

B) Implementado
C)
D)