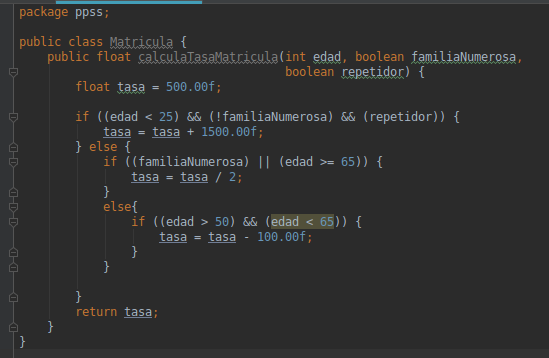
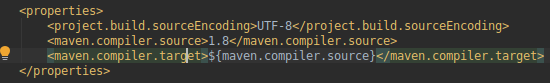
1. **P03-Automatización de pruebas: Drivers**
   1. **Ejercicio 1: drivers para calculaTasaMatricula()**

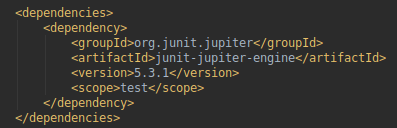
**A) Necesitas el código fuente de la unidad a probar. Puedes copiarlo de las plantillas de la primera práctica. No olvides crear el paquete ppss, tal y como se indica arriba en el nombre del método.**

****

**Asegúrate de que el código de tu SUT esté en tu disco duro en el directorio requerido por Maven. Añade en el pom las propiedades <project.build.sourceEcoding>, <maven.compiler.source> y <maven.compiler.target> (cópialas del pom de las plantillas de la primera práctica)**

****

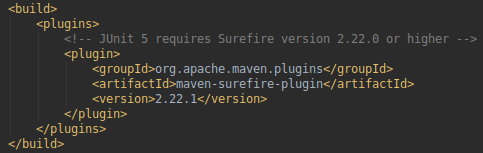
**B) A continuación vamos a implementar los drivers asociados a tu tabla de diseño de casos de prueba (o usa la tabla que os hemos proporcionado). Primero incluye en el pom las dependencias necesarias. Y a continuación, muestra en el editor el código de la clase Matricula, selecciona el nombre de la clase, y desde el menú contextual pulsa sobre "Generate...→Test...". Asegúrate de que el test que se va a generar es JUnit5. Selecciona finalmente la casilla con la unidad a probar.**

****

**El nombre de cada driver será "C1\_calculaTasaMatricula",... y así sucesivamente. Puedes usar las anotaciones que hemos visto en clase exceptuando @Tag y @ParameterizedTest.. Una vez implementados los tests, compílalos usando la fase test-compile de Maven. Para que IntelliJ nos muestre esa fase debes deseleccionar "Show Basic Phases Only" desde la rueda dentada de la ventana Maven ¿Tienes claro qué diferencia hay entre ejecutar mvn compiler:testCompile y mvn test-compile? Puedes comprobarlo si ejecutas previamente la fase clean antes de cada uno de dichos comandos.**

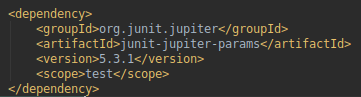
Todo perfecto.

**C) Recuerda que en el pom todavía no hemos añadido la sección <build>. Ejecuta los tests desde la fase Maven. Comprueba que NO se ejecuta ningún test, a pesar de que, por defecto, el plugin surefire está incluido en el pom. Como ya vimos en la primera práctica, JUnit5 necesita otra versión del plugin. Modifica el pom para incluir el plugin surefire con la versión correcta. Si ahora ejecutas de nuevo la fase test, deberías ver en el informe Maven que se han ejecutado todos los tests. Si quieres ver el informe de forma gráfica selecciona el icono con la "M" de color rojo a la derecha de la rueda dentada de la ventana Maven.**

****

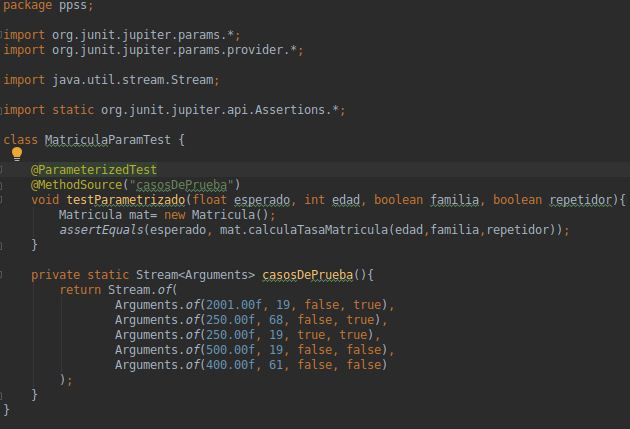
Los test funcionan correctamente.

**D) Implementa en una nueva clase con nombre "MatriculaParamTest" un test parametrizado con los datos de la tabla de casos de prueba proporcionada. Deberás modificar el pom para añadir la dependencia correspondiente. Nota: todas las anotaciones que hemos visto en clase, que pueden usarse conjuntamente con la anotación @Test, se pueden usar igualmente con la anotación @ParameterizedTest. Es decir, que puedes usar @BeforeEach, @BeforeAll, ... o cualquier otra independientemente de que el driver esté anotado con @Test o con @ParemeterizedTest.**

****

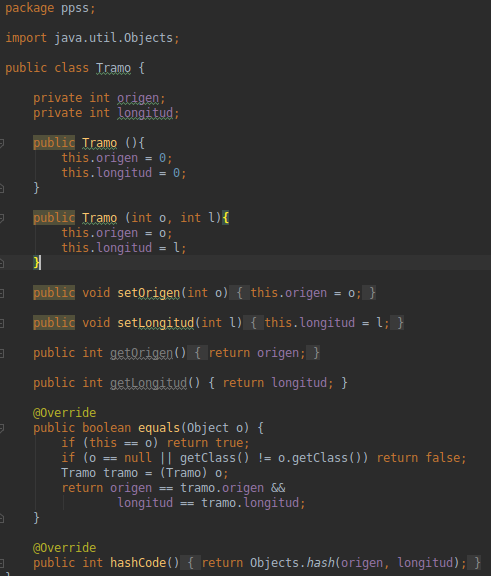
**Ejecuta de nuevo la fase test de Maven, deberías obtener informes idénticos tanto para los drivers de MatriculaTest, como para los de MatriculaParamTest. Fíjate en que la goal surefire:test ejecuta TODOS los métodos anotados con @Test o @ParameterizedTest.**

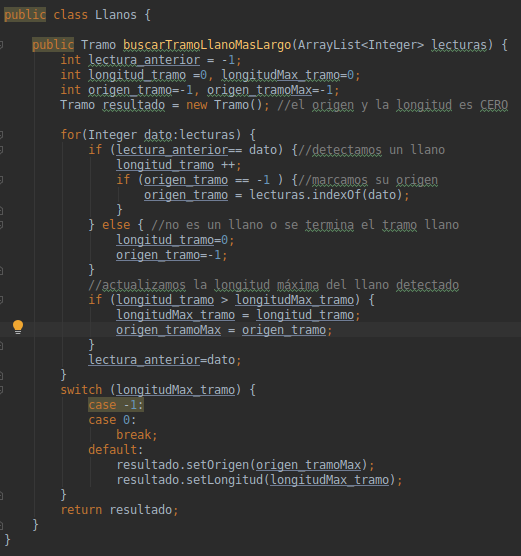
Todo bien.

****

* 1. **Ejercicio 2: drivers para buscarTramoLlanoMasLargo()**

**A) Añade el código fuente de la unidad en el paquete ppss, y la clase Llanos El código es el proporcionado en el enunciado de la práctica anterior. Añade también una nueva clase Tramo, con dos atributos privados: origen y longitud (de tipo entero), así como sus correspondientes getters y setters, más dos constructores para la clase: uno sin parámetros (en el que se inicializan a cero los dos atributos), y otro en el que los valores de los atributos se pasan como parámetros en el constructor. Nota: En el código de la práctica anterior se usa el método setDuracion(), debería ser setLongitud().**

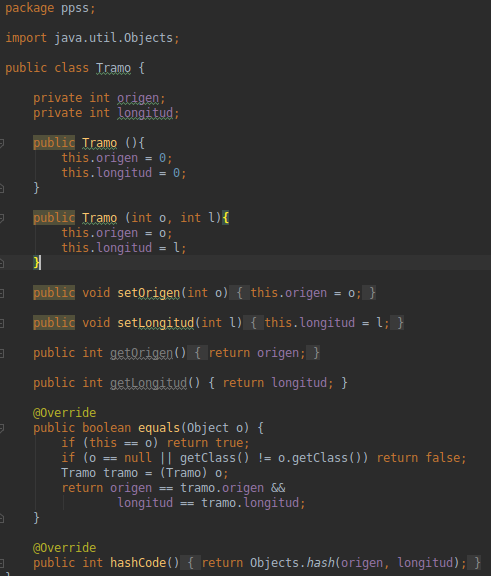
****

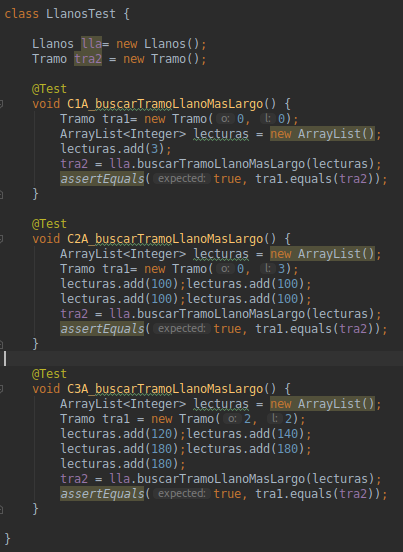
****

**B) A continuación implementa los drivers asociados a tu tabla de diseño de casos de prueba (o usa la**

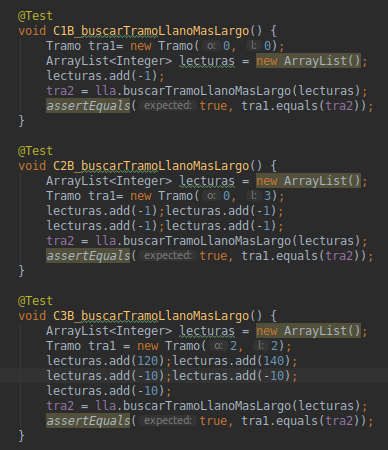
**tabla que os hemos proporcionado). La clase que contiene los drivers debe llamarse LlanosTest.**

**El nombre de cada driver será "C1A\_buscarTramoLlano",... y así sucesivamente. No uses tests parametrizados**

****

****

**C) Añade a la clase LlanosTest, tres nuevos drivers asociados a la siguiente tabla adicional, obtenida también aplicando el método del camino básico. Los nombres de los drivers deben ser C1B\_buscarTramoLlano,... y así sucesivamente.**

****

**D) Ejecuta todos los tests y depura el código si detectas algún error. Recuerda que, aunque realices modificaciones en el código para depurarlo, TODOS los tests deben seguir en verde. También debes**

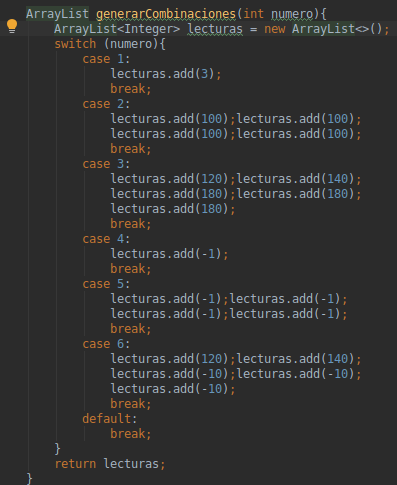
**intentar depurar el código SIN cambiar la estructura del CFG del método que queremos probar, si no lo haces así, tendrás que revisar la tabla, puesto que es probable que deje de ser efectiva y eficiente..**

Solo hay que cambiar esta variable: **int lectura\_anterior = 0;**

**E) Implementa en una nueva clase con nombre "LlanosParamTest" un test parametrizado con los datos de las tablas de casos de prueba A y B. Ejecuta de nuevo la fase test de Maven, deberías obtener informes idénticos tanto para los drivers de LlanosTest, como para los de LlanosParamTest.**

****

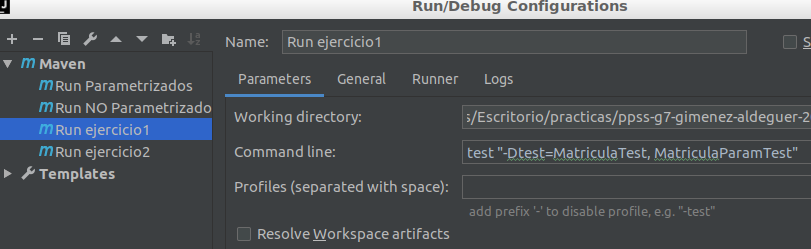
Generar las diferentes combinaciones para el arraylist:

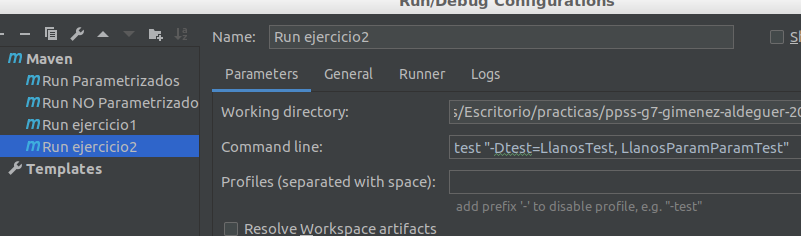
****

* 1. **Ejercicio 3: selección y filtrado de las ejecuciones de los tests**

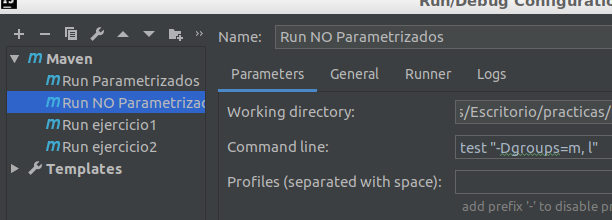
**A) Crea "Configurations" para ejecutar las clases con los drivers del Ejercicio1, y las del Ejercicio2, con los nombres "Run ejercicio1" y "Run ejercicio2", respectivamente (usando el parámetro "test" del plugin surefire). Nota: Si el valor que asignamos al parámetro "test" contiene espacios en blanco, deberás usar dobles comillas para que IntelliJ no lo interprete como el nombre de una fase y/o goal, es decir usaremos: -Dtest=“eEqueta1, etiqueta2”, en vez de. -Dtest=etiqueta1, etiqueta2.**

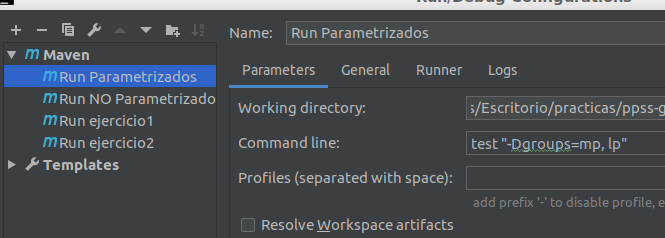
****

****

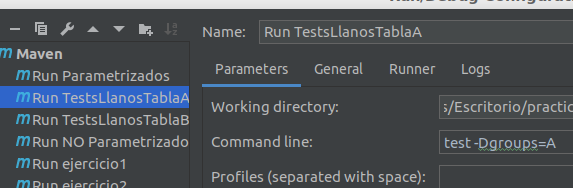
****

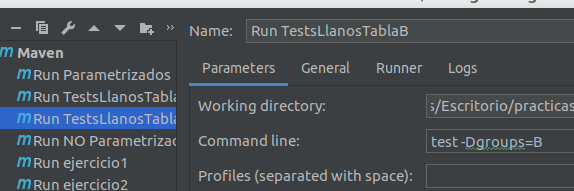
**B) Etiqueta (anotación @Tag) el código de pruebas, de forma que podamos ejecutar todos los tests parametrizados, y crea una "configuration" con nombre "Run Parametrizados", o todos los tests sin parametrizar (y añade la "configuration" con nombre "Run NO Parametrizados".**

****

****

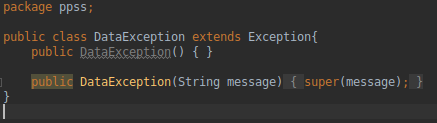
**C) Etiqueta (anotación @Tag) el código de pruebas, de forma que podamos ejecutar únicamente los tests de la Tabla A del ejercicio anterior, y crea una "configuration" con nombre "Run TestsLlanosTablaA", o sólo los tests de la Tabla B (y añade la "configuration" con nombre "Run TestsLlanosTablaB".**

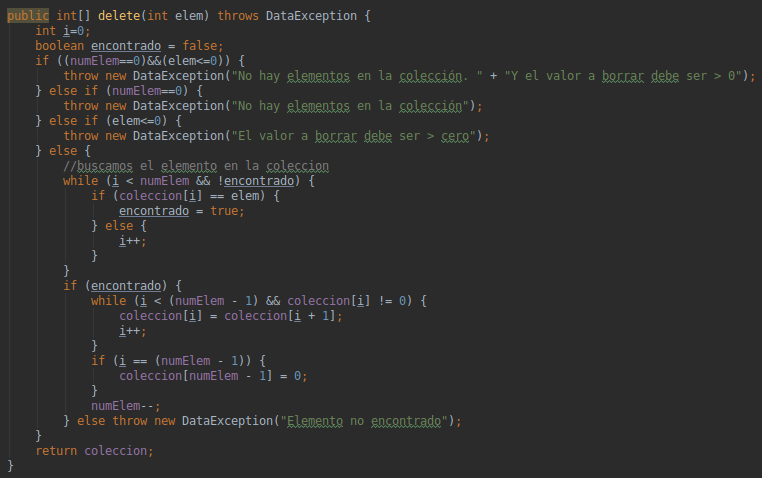
****

****

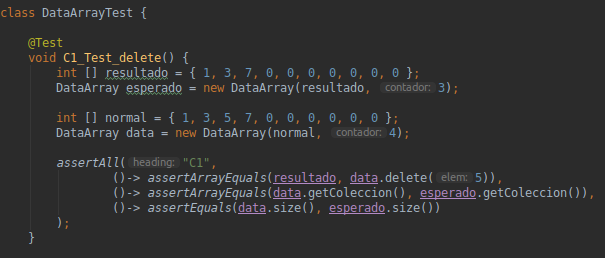
* 1. **Ejercicio 4: pruebas de excepciones y agrupaciones de aserciones**

**Implementa en una clase DataArrayTest, los drivers correspondientes a la tabla de casos de prueba proporcionada. Deberás agrupar las aserciones y comprobar que el mensaje de las excepciones generadas es el correcto.**

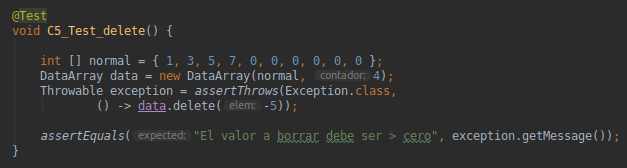
****

****

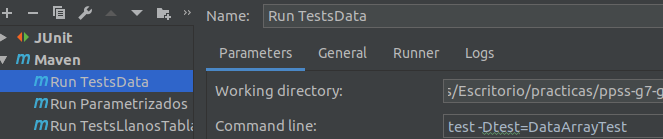
Las aserciones las he hecho así:

****

Y comprobar el mensaje de las excepciones generadas así:

****

**Añade una nueva “configuration” para poder ejecutar únicamente la clase DataArrayTest, con el nombre “Run DataArrayTest” (no es necesario que uses la anotación @Tag).**

****