Laboratorio L11 – Parte 1 (guía)

JUnit 5.0

Objetivos

- Comprender la necesidad de verificar el software
- Sistematización de pruebas unitarias de Java utilizando JUnit5

Herramientas que vamos a utilizar

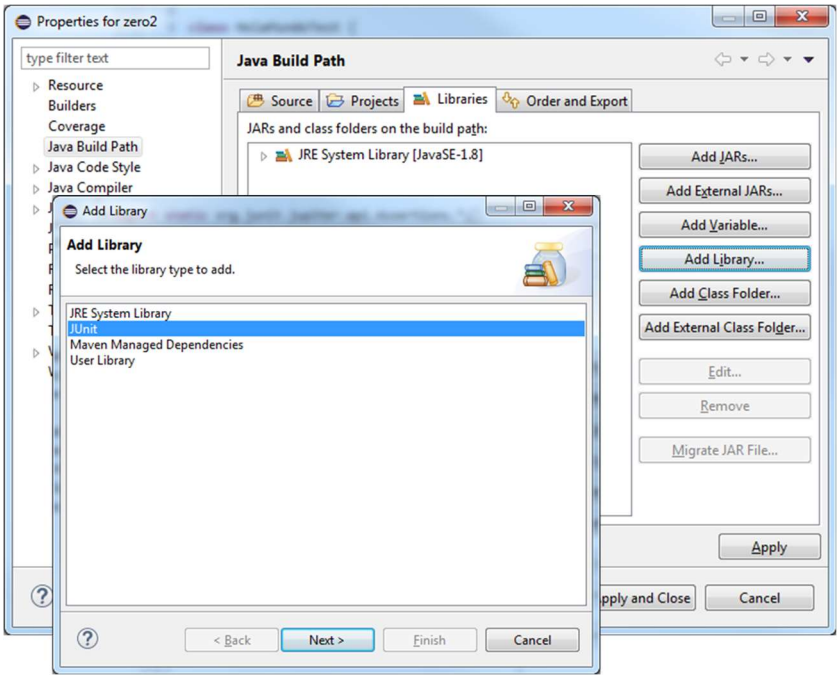
- Entorno de desarrollo *Eclipse*
- Librería *jUnit*

Entregas

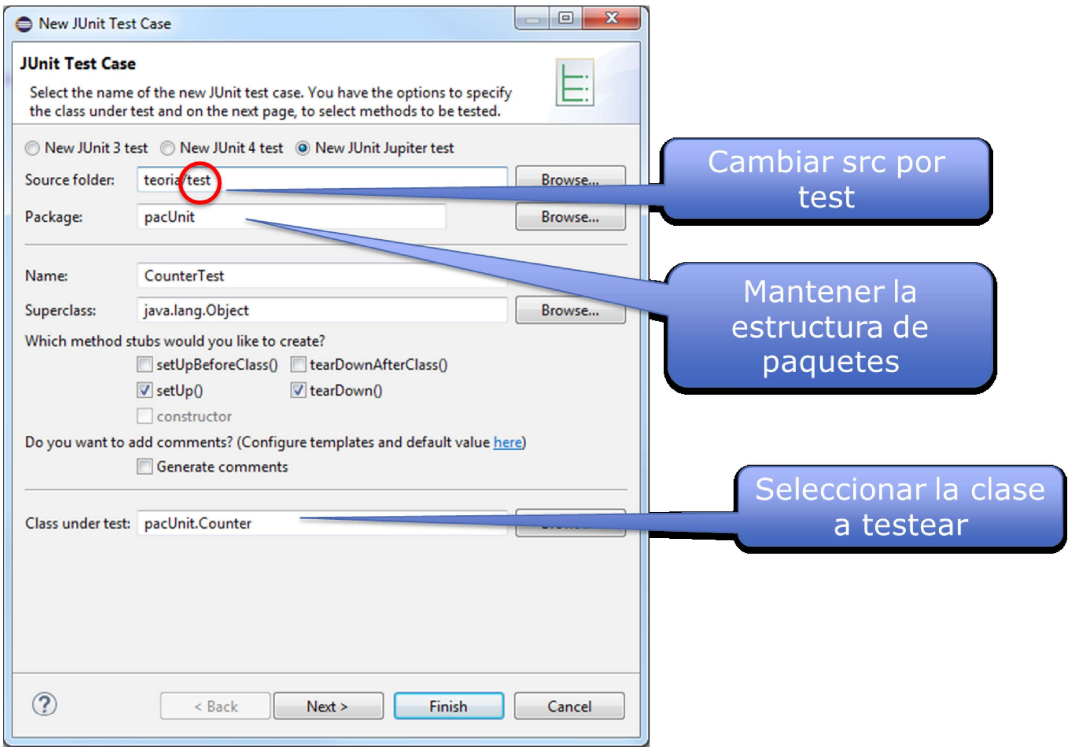
Fichero ZIP con la exportación del laboratorio L11. (***apellidoNombre_L11.zip)***. La entrega debe ser ***individual*** y realizarse a través de **eGela**.

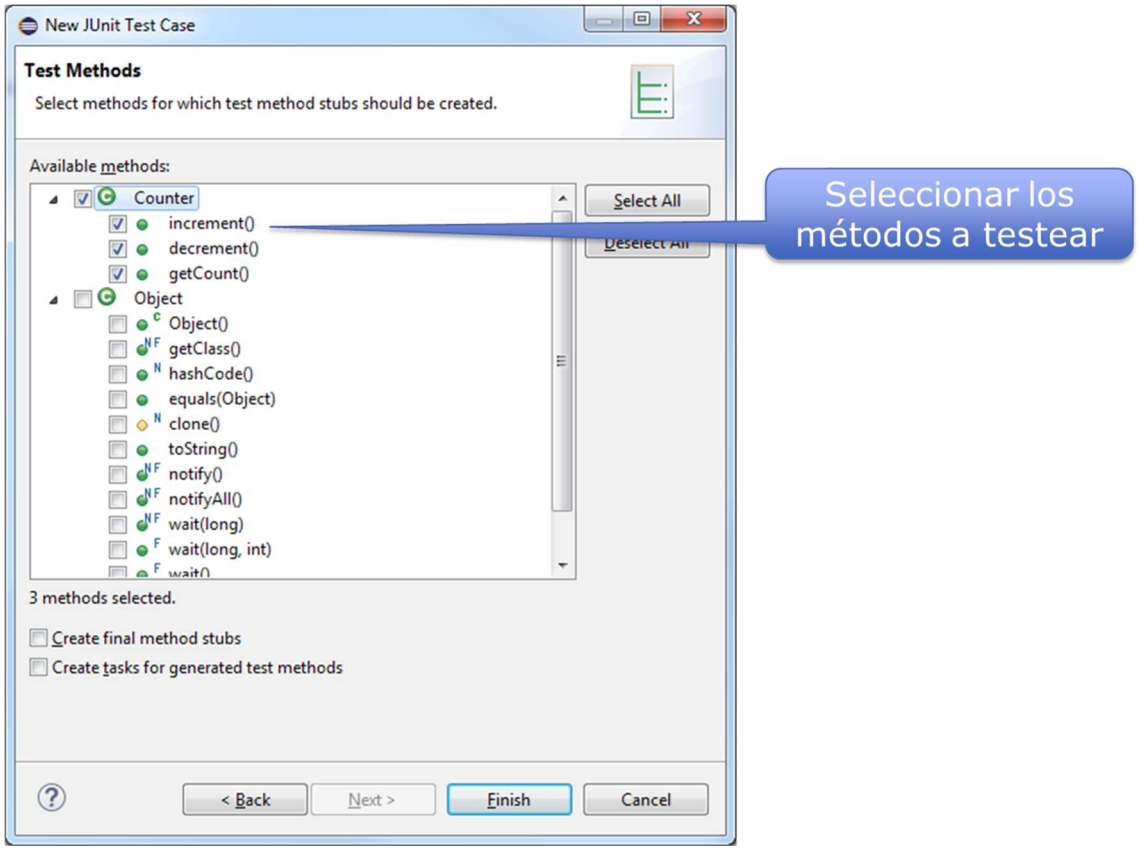
Tareas a realizar

1. Importa el proyecto **PROYCounter** que está en eGela. Este proyecto tiene el paquete `packUnit` donde está la clase `Counter` que se va a verificar con `JUnit`.
2. Crear la clase `Test` dentro de la carpeta *test* seleccionando las siguientes opciones a partir del menú principal: *File* → *New* → *Source Folder*
3. Añade el path de la librería de JUnit al path: ir a *Project* →*Properties* y en la opción *Java Build Path*, seleccionar la pestaña *Libraries*. Pinchar sobre el botón *AddLibrary*. Finalmente seleccionar que se quiere añadir la librería JUnit (y la versión 5).



4. Para generar la clase de test, selecciona la clase a verificar y ejecuta *File-→New-→JUnitTestCase* (si no aparece seleccionar *File-→New-→Other* y buscar *JUnitTestCase*). Sigue los pasos marcados a continuación





5. En la clase de test CounterTest que acabas de crear copia lo siguiente:

   a) Declara el atributo counter1

```
private Counter counter1;
```

   b) En el método setUp se generará el objeto que se utilizará por todos los métodos test

```
...
@BeforeEach
void setUp() {// Generate the objects to be used in all the test methods
        counter1 = new Counter();
}
```

   c) En el método testIncrement hay que comprobar que si incrementamos una vez, el valor devuelto es 1 y la segunda vez 2

```
@Test
void testIncrement() {
        assertEquals(1, counter1.increment());
        assertEquals(2, counter1.increment());
    }
```

   d) En el método testDecrement, comprobar que si se decrementa una vez, devuelve el valor -1

```
@Test
void testDecrement() {
        assertTrue(counter1.decrement() == -1);

}
```

6. Ejecuta los Test. Para ello selecciona *Run→Run as→JUnitTest*

   Analiza qué ocurre.

7. Cambia la constructora de la clase Counter e inicializa el atributo count a 10. Vuelve a ejecutar y mira lo que sucede
8. Usa en el método testDecrement, la versión sobrecargada de *assertTrue* que admite añadir un String al final. Cambia el mensaje de error que se mostrará.
9. Añade al principio de la clase y antes del método testDecrement la anotación @DisplayName. Vuelve a ejecutar y mira lo que sucede

Parte 2.– Lotería primitiva

JUnit 5.0 + Ficheros

10. Importa el proyecto **LoteriaPrimitiva** que está disponible en eGela.
11. Verifica la implementación del método `lowestNumber`. Casos de prueba a considerar:

    a) Cuando no hay números en la combinación
    b) Cuando hay números en la combinación y el mínimo está en la primera posición
    c) Cuando hay números en la combinación y el mínimo está en la última posición
    d) Cuando hay números en la combinación y el mínimo está en una posición intermedia

12. Añade el método `storeInFile` en la clase `Ticket`. Este método escribe en el fichero que recibe como parámetro, la información de un `Ticket`, formateada según está definida en su método `toString`. El fichero debe estar en la carpeta *data*. Si no está o no se puede crear en dicha carpeta, se elevará la excepción correspondiente.
13. Cambia el método `main` de clase `SimulatorScanner` para probar el método creado en el punto anterior. Comprueba que, tras la llamada, se ha creado correctamente el fichero.
14. Verifica la implementación del método `getPosNumber`. Casos de prueba a tener en cuenta:

    a) Cuando no hay números en la combinación
    b) Cuando en la combinación hay números y se intenta recuperar el valor de una posición válida
    c) Cuando hay números en la combinación y se intenta recuperar el valor de una posición menor que 0
    d) Cuando hay números en la combinación y se intenta recuperar el valor de una posición en la que aún no hay número
    e) Cuando hay números en la combinación y se intenta recuperar el valor de una posición inexistente ya que supera la cantidad máxima de números que puede tener una combinación

15. Añade el método de `readFromFile` en la clase `Ticket`: lee desde el archivo que este método recibe como parámetro, la combinación que contiene. Si el fichero no existe, se elevará la excepción producida. El formato del contenido del fichero será igual al generado por el método `storeInFile` y suponemos que siempre va a estar correctamente creado.
16. Cambiar el método `main` de la clase `SimulatorScanner` para incluir la llamada al nuevo método creado. Comprueba que dicho método lee correctamente la combinación de números contenida en fichero.

Ejercicios complementarios

17. Verificar la implementación del método `addNumber`. Casos de prueba a tener en cuenta:

    a) Cuando no se eleva ninguna excepción

    b) Cuando se eleva la excepción NumberOutOfRange porque el número es demasiado grande

    c) Cuando se eleva la excepción NumberOutOfRange porque el número es demasiado pequeño

    d) Cuando al repetir un número se eleva la excepción IllegalArgumentException

    e) Cuando se eleva la excepción IndexOutOfBoundsException al intentar añadir un nuevo número en una combinación que está llena

18. Mejora el tratamiento de las excepciones que pueden elevarse al llamar desde el main al método storeInFile de la clase SimulatorScanner, haciendo que en caso de que haya algún problema para crear el fichero con el nombre introducido por el usuario, se le pida que introduzca otro nombre, repitiendo esta petición hasta que todo vaya bien.

19. Mejora el tratamiento de las excepciones que pueden elevarse al llamar desde el main al método readFromFile de la clase SimulatorScanner, haciendo que en caso de que no encuente el fichero con el nombre introducido por el usuario, se le pida que introduzca otro nombre, repitiendo esta petición hasta que teclee un nombre o path correcto.