INFORME PRÁCTICA 4: PRUEBAS UNITARIAS DE SOFTWARE

Zdravko Dimitrov Aranudov
Sara Grela Carrera

1. Introducción

Este documento pretende informar de las pruebas unitarias realizadas para la clase Seguro, SegurosGUI y ListaOrdenada. En cada apartado se explicarán las pruebas de caja negra y blanca realizadas, así como la cobertura obtenida. Además, en la interfaz SegurosGUI se explicarán las pruebas de integración que se han hecho con la herramienta FEST.

A modo de aclaración para el docente, se incluyen algunos comentarios adicionales sobre comandos de Maven utilizados, dependencias añadidas al fichero POM.xml y/o errores que hayamos detectado.

Como se ha mencionado anteriormente, se han aplicado las técnicas de caja negra y blanca, escogiendo como criterios la cobertura de sentencias, la de decisiones y la de condiciones, ya que el código presentaba sentencias "if". El resto se han descartado por no considerarse necesarias en este caso, ya que, al ser clases simples, estos tres criterios son suficientes.

2. Proceso de pruebas de la clase Seguro

En la clase Seguro, comenzamos definiendo las siguientes pruebas de caja negra:

Método precio():

Parámetros	Clases válidas	Valores	Clases no válidas	Valores
Nivel de cobertura	1. Todo riesgo. 2. Terceros + lunas. 3. Terceros simple.	Todo riesgo Terceros + lunas Terceros simple	11. !={Todo riesgo, terceros + lunas, terceros simple}	Terceros ampliados
Potencia	4. [90, 110] 5. (110, ∞)	90 100 110 ------ 200	12. < 0 13. !Número	-1 AAA
Fecha último siniestro	6. [hoy – 1 año, hoy] 7. [hoy – 3 años, hoy – 1 año) 8. (-∞, hoy – 3 años)	Hoy – 1 año Hoy – 6 meses Hoy ------ Hoy – 3 años Hoy – 2 años ------ Hoy – 5 años	14. > hoy 15. !fecha	Cualquier fecha no válida (fecha con mes 13) Mañana
Grado minusvalía	9. True. 10. False.	True False	-	-

Casos de prueba válidos:

(Todo riesgo, 90, hoy – 1 año, True): 937,5

(Terceros + lunas, 100, hoy – 6 meses, False): 830

(Terceros simple, 110, hoy, True): 465

(Todo riesgo, 200, hoy – 3 años, False): 1250

(Terceros + lunas, 200, hoy – 2 años, False): 770

(Terceros simple, 90, hoy – 5 años, False): 420

Casos de prueba no válidos:

(Terceros ampliados, 110, hoy, True): DatoIncorrectoException

(Todo riesgo, -1, hoy – 1 año, False): DatoIncorrectoException

(Terceros + lunas, AAA, hoy – 6 meses, True): NumberFormatException

(Terceros simple, 200, '03/13/2021', False): DatoIncorrectoException

(Todo riesgo, 110, mañana, False): DatoIncorrectoException

Método Seguro():

Parámetros	**Clases válidas**	**Valores**	**Clases no válidas**	**Valores**
Potencia	1. [90, 110] 2. (110, ∞)	90 100 110 ------ 200	7. < 0 8. !Número	-1 AAA
Cliente	3. No null	-	9. Null	-
Cobertura	4. Todo riesgo. 5. Terceros + lunas. 6. Terceros simple.	Todo riesgo Terceros + lunas Terceros simple	10. !={Todo riesgo, terceros + lunas, terceros simple}	Terceros ampliados

Casos de prueba válidos:

(90, No null, Todo riesgo)

(100, No null, Terceros + lunas)

(110, No null, Terceros simple)

(200, No null, Todo riesgo)

Casos de prueba no válidos:

(-1, No null, Todo riesgo): DatoIncorrectoException

(AAA, No null, Todo riesgo): NumberFormatException

(90, Null, Terceros + lunas): NullPointerException

(100,No null, Terceros ampliados): DatoIncorrectoException

A continuación, pasamos con las pruebas de caja blanca, en las que los criterios de cobertura aplicados han sido: de sentencias, decisiones y condiciones. Con ellos, hemos comprobado que no se había conseguido cobertura de sentencias completas. Algunas instrucciones no se ejecutaban nunca cuando sí debían hacerlo. Por ello, hemos añadido el caso de prueba válido:

(Terceros, 10, hoy, False): 600

En cuanto a los errores detectados, en las pruebas de cobertura hemos detectado métodos que habíamos definido, pero nunca utilizábamos porque no eran necesarios. También estábamos implementando incorrectamente una comparación de un objeto con nulo. Además, aunque no es un error como tal, también vimos que podíamos simplificar nuestro código debido a los "ifs" que teníamos, por lo que lo modificamos cambiándolos por un "switch".

3. Proceso de pruebas de la clase SegurosGUI

A continuación, realizaremos pruebas de integridad para comprobar el funcionamiento conjunto entre la clase SegurosGUI y Seguros.

Después de haber verificado que ambas clases funcionan según lo esperado, es suficiente añadir un caso de prueba válido que devuelva el precio del seguro en la interfaz. Además, harán falta dos casos de prueba no válidos para demostrar que se notifica al usuario cuando alguno de los campos sea incorrecto. Para estos casos de prueba, sirven las clases de equivalencia realizadas para el método precio() de la clase Seguro.

Caso de prueba válido:

(TODORIESGO, 90, hoy - 1 año, True): 937,5

Casos de prueba no válidos:

(TERCEROSLUNAS, -1, hoy - 2 años, False): DatoIncorrectoException

(TERCEROS, 40, mañana, True): DateTimeParseException

Para probar estos casos, hemos empleado la herramienta FEST, para lo que hemos tenido que incluir en el POM la siguiente dependencia:

<dependency>

```
        <groupId>org.easytesting</groupId>
         <artifactId>fest-swing</artifactId>
          <version>1.2.1</version>
</dependency>
```

4. Proceso de pruebas de la clase ListaOrdenada

Para esta parte se nos proporcionó primero un archivo jar para probar los métodos de esta clase sin conocer su código. Para utilizarlo, hemos añadido la correspondiente dependencia en el POM junto con el comando de Maven "mvn install:install-file -Dfile=C:\Users\Sara\Desktop\Uni\ListaOrdenada-0.0.1.jar -DgroupId="es.unican.is2" -DartifactId=ListaOrdenada -Dversion="0.0.1" -Dpackaging=jar".

En cuanto a las pruebas de caja negra, Para todos los métodos que se prueban, también hay que tener en cuenta el estado de la lista. Por comodidad lo pondré en este comentario, ya que es para todos igual: lista vacía, lista con un elemento y lista con elementos.

Método get(int índice):

Parámetros	Clases válidas	Valores	Clases no válidas	Valores
Índice	1. [0, lista.size() -1]	0 4 Lista.size() - 1	2. < 0 3. > lista.size()	-1 Lista.size() + 1

Casos válidos	Casos no válidos
(1, []): 1 (3, [1,2,3,4]): 4 (0, [1]): 1 (2, [1,2,3,4]): 3	(2, []): IndexOutOfBoundsException (-1, [1,2,3,4]): IndexOutOfBoundsException (4, [1,2,3,4]): IndexOutOfBoundsException

Método add(E elemento):

Parámetros	Clases válidas	Valores	Clases no válidas	Valores
elemento	1. !null		2. null	

Casos válidos	Casos no válidos
(1, []): [1] (2, [1]): [1,2] (3, [1,2,3,4]): [1,2,3,3,4]	(null, [1]): NullPointerException (null, [1,2,3,4]): NullPointerException

(5, [1,2,3,3,4]): [1,2,3,3,4,5]	

Método remove(int índice):

Parámetros	**Clases válidas**	**Valores**	**Clases no válidas**	**Valores**
Índice	1. [0, lista.size() -1]	0 4 Lista.size() - 1	2. < 0 3. > lista.size()	-1 Lista.size() + 1

Casos válidos	**Casos no válidos**
(0, [1]): (1, []) (3, [1,2,3,4]): (4, [1,2,3]) (1, [1,2,3]): (2, [1,3])	(2, []): IndexOutOfBoundsException (-1, [1,2,3,4]): IndexOutOfBoundsException (4, [1,2,3,4]): IndexOutOfBoundsException

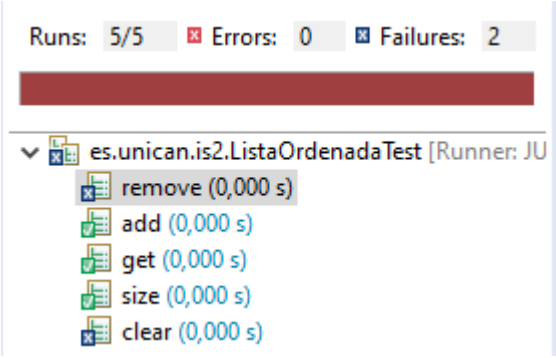
Método size():

Casos válidos	**Casos no válidos**
([]): 0 ([1]): 1 ([1,2,3,4]): 4	

Método clear():

Casos válidos	**Casos no válidos**
([]): ([]) ([1]): ([]) ([1,2,3,4]): ([])	

Con estos casos de prueba hemos detectado que los métodos erróneos son el remove(int index) y el clear(). En el método de eliminar un elemento, nos daba error en el caso de prueba válido al tener la lista con un elemento. Tras añadir un elemento, este método no lo está eliminando correctamente (también lo hemos comprobado imprimiéndolo por pantalla y, efectivamente, no lo elimina). Algo similar ocurre con el método de limpiar la lista, en el que hemos comprobado que no elimina correctamente los elementos.



Al añadir la clase ListaOrdenada y ver la implementación de los métodos, comprobamos que, efectivamente, estos dos no son correctos. El método remove(int index) elimina el elemento de la posición anterior y el clear() lo que hace es clonar la lista, no vaciarla.

Al ejecutar las pruebas de caja blanca, obtenemos tanto cobertura de sentencias como de condición y decisión completas, por lo que podríamos dar por finalizada la práctica.