INFORME PRÁCTICA 4: PRUEBAS UNITARIAS DE SOFTWARE

1. Introducción

El objetivo de este documento es plasmar el proceso de pruebas que hemos empleado para cada clase pedida. Además de esto, comentaremos los fallos que hemos conseguido detectar gracias a este proceso.

2. Proceso de pruebas unitarias de la clase Seguro

Las pruebas que vamos a emplear para la clase Seguro son pruebas de caja negra, ya que hemos realizado las pruebas basándonos sólo en la especificación del método, es decir, en sus entradas y en sus salida. No hemos tenido en cuenta la implementación que realizamos previamente.

Para las pruebas unitarias de la clase Seguro hemos realizado únicamente las del método precio ya que de las demás no se requería.

A continuación se muestra una tabla con las variables que intervienen en el método precio, así como sus clases válidas y no válidas, englobando todos sus posibles valores.

precio():

VARIABLES	CLASES VÁLIDAS	CLASES NO VÁLIDAS
cobertura	TERCEROS, TODORIESGO, TERCEROSLUNAS	!={TERCEROS,TODORIESGO,TERCEROSLUNAS}
potencia	>= Inicio_Tramo_1 && <= Fin_Tramo_1, > Fin_Tramo_1, < Inicio_Tramo_1	null, potencia < 0
fechaContratacion	> fechaActual – 1año && <= fechaActual, <= fechaActual – 1año && > fechaActual – 2años, <= fechaActual – 2años	null, fechaContratacion > fechaActual

En la siguiente tabla hemos listado los valores que vamos a emplear en las pruebas, dichos valores abarcan todo el abanico de posibilidades, tanto válidas como no válidas para las variables anteriormente mencionadas.

VALORES:

1. TERCEROS 2. TODORIESGO 3. TERCEROSLUNAS 4. NOVEL
5. 90 6. 120 7. 50 8. -10 9. null
10. 2022-10-13 11. 2021-10-13 12. 2020-10-13 13. 2023-10-13 14. null

Los casos de prueba siguientes son una combinación de valores válidos y no válidos, de los cuales hemos indicado la salida esperada. Esto quiere decir que, si nuestra implementación fuese correcta, para cada caso de prueba deberíamos obtener la salida que lo acompaña.

CASOS DE PRUEBA:

VALIDOS: (TERCEROS, 90, 2022-10-13) : 336 € (TODORIESGO, 120, 2021-10-13) : 1080 € (TERCEROSLUNAS, 50, 2020-10-13): 600 €
NO VALIDOS: (NOVEL, 90, 2022-10-13): No se puede implementar (TERCEROS, -10, 2022-10-13): PotenciaNegativaEx (TERCEROS, null, 2022-10-13): No se puede implementar (TODORIESGO, 90, 2023-10-13): FechaContratacionFuturaEx (TERCEROS, 90, null): FechaNulaEx

Este proceso nos ha ayudado a detectar que en la clase Seguro no habíamos implementado la gestión de errores para los atributos, ni habíamos lanzado excepciones en los casos debidos.

Al finalizar estas observaciones hemos añadido al método las partes que le faltaban para estar correcto.

3. Proceso de pruebas de integración de la clase Cliente

Las pruebas que vamos a emplear para la clase Cliente son pruebas de caja negra, al igual que en el apartado anterior. No hemos tenido en cuenta la implementación que realizamos previamente.

Para las pruebas unitarias de la clase Cliente hemos realizado únicamente las del método totalSeguros ya que de los demás no se requería.

A continuación se muestra una tabla con las variables que intervienen en el método totalSeguros, así como sus clases válidas y no válidas, englobando todos sus posibles valores.

totalSeguros():

VARIABLES	CLASES VÁLIDAS	CLASES NO VÁLIDAS
minusvalia	true, false	null
seguros	Con elementos, vacío	

En la siguiente tabla hemos listado los valores que vamos a emplear en las pruebas, dichos valores abarcan todo el abanico de posibilidades, tanto válidas como no válidas para las variables anteriormente mencionadas. En el caso de los valores 4-5 se refieren a la variable seguro, que es una lista y cada uno de sus elementos es de la clase Seguro.

VALORES:

1. True 2. False 3. null
4. [seguro0, seguro1, seguro2] 5. []

Los casos de prueba siguientes son una combinación de valores válidos y no válidos, de los cuales hemos indicado la salida esperada. Esto quiere decir que, si nuestra implementación fuese correcta, para cada caso de prueba deberíamos obtener la salida que lo acompaña.

CASOS DE PRUEBA:

<u>VALIDOS:</u> ([seguro0, seguro1, seguro2], true) : 1512€ ([seguro0, seguro1, seguro2], false) : 2016€ ([], false) : 0€
<u>NO VALIDOS:</u> ([], null) : No se puede implementar

Al finalizar la fase de prueba no detectamos ningún error en los casos de prueba por tanto consideramos que el código implementado era correcto.

4. Proceso de pruebas de integración de la interfaz

Las pruebas que vamos a emplear para la interfaz son pruebas de caja negra, al igual que en los apartados anteriores.

A continuación se muestra una tabla con la única variable que interviene en la interfaz, así como sus clases válidas y no válidas, englobando todos sus posibles valores.

Interfaz:

VARIABLES	CLASES VÁLIDAS	CLASES NO VÁLIDAS
DNI	String de longitud > 0, “”	

En la siguiente tabla hemos listado los dos valores que vamos a emplear en las pruebas, dichos valores abarcan todo el abanico de posibilidades, tanto válidas como no válidas para el DNI.

VALORES:

1. 12345678A
2. “”

Los casos de prueba siguientes abarcan todos los casos posibles, de los cuales hemos indicado la salida esperada. Esto quiere decir que, si nuestra implementación fuese correcta, para cada caso de prueba deberíamos obtener la salida que lo acompaña.

CASOS DE PRUEBA:

VALIDOS:
(12345678A) : Muestra nombre, matricula, tipo de cada seguro, total a pagar
(“”) : DNI No Válido

Al realizar las pruebas detectamos que había varios fallos en el código que nos habían proporcionado y realizamos los cambios necesarios para su correcto funcionamiento.

5. Proceso de pruebas unitarias de la clase ListaOrdenada

Las pruebas que vamos a emplear para la clase ListaOrdenada son pruebas de caja negra, al igual que en apartados anteriores.

MÉTODO	PARAMETROS	CLASES VÁLIDAS	CLASES NO VÁLIDAS
get	Índice	>= 0	< 0
	lista	Lista ordenada con elementos	Lista no ordenada, lista vacía
add	elemento	!= null	Null
	Lista	Lista ordenada con elementos, lista vacía	Lista no ordenada

remove	Índice	>= 0	< 0
	Lista	Lista ordenada con elementos	Lista no ordenada, lista vacía
size	Lista	Lista ordenada con elementos, lista ordenada vacía	Lista no ordenada
clear	Lista	Lista ordenada con elementos, lista ordenada vacía	Lista no ordenada

Los casos de prueba siguientes abarcan todos los casos posibles, de los cuales hemos indicado la salida esperada. Esto quiere decir que, si nuestra implementación fuese correcta, para cada caso de prueba deberíamos obtener la salida que lo acompaña.

Cabe destacar la "sintaxis" que hemos empleado, para los casos en los que aparezca "(num, []) : salida", el primer elemento indica la entrada, por ejemplo en el get (0,[1,2,3]) corresponde a que el 0 es el índice, [1,2,3] la lista y 1 la salida del método get. En los casos en los que aparece "→" significa que no es la salida, si no que hay elementos modificados.

CASOS DE PRUEBA:

Nota: en ninguno de los métodos es implementable la lista no ordenada.

MÉTODO	CASOS VÁLIDOS	CASOS NO VÁLIDOS
get	(0, [1]) : 1	(-1, [1,2,3]): IndexOutOfBoundsException (0, []) : IndexOutOfBoundsException
add	(3, [1,2,4]) → [1,2,3,4] (1, []) → [1]	(null, [1,2,3]) : NullPointerException (null, []) : NullPointerException
remove	(0, [1,2,3,4]) : 1 → [2,3,4]	(-1, [1,2,3]) : IndexOutOfBoundsException (0, []) : IndexOutOfBoundsException
size	[2,3,4] : 3 [] : 0	
clear	[1,2,3] → [] [] → []	

Al realizar las pruebas hemos encontrado un error en el método add, ya que añade los elementos pero no de manera ordenada. Y además, encontramos errores también el método clear.

Hemos corregido dichos errores y han pasado correctamente todos los test implementados para las pruebas de caja negra. Hemos observado el código y no vemos necesario hacer test de caja blanca ya que consideramos que los test de caja negra cubren todas las posibilidades.