# INFORME PRÁCTICA 4: PRUEBAS UNITARIAS DE SOFTWARE

## 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.

<Breve descripción del proceso de pruebas general que se ha aplicado: fases, técnicas de prueba aplicadas, criterios de cobertura elegidos, herramientas utilizadas para cada tipo de prueba, etc.>

<En las siguientes secciones (2, 3 y 4), explicar el detalle de la aplicación de las técnicas de prueba para cada una de las clases involucradas y los resultados obtenidos.

* En el caso de pruebas de caja negra, incluir una tabla con las clases de equivalencia identificadas, los valores interesantes seleccionados y los casos de prueba definidos a partir de ellos.
* Para pruebas de caja blanca, criterios de cobertura aplicados y cobertura conseguida. Si ha sido necesario ampliar el conjunto de casos de prueba definidos en el paso anterior, indicar cuáles son los casos añadidos.
* En la medida de lo posible, listar los errores que se han detectado en el proceso de pruebas.>

## Proceso de pruebas unitarias de la clase Seguro

Las pruebas que vamos a emplear 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 |
| 1. 90 2. 120 3. 50 4. -10 5. null |
| 1. 2022-10-13 2. 2021-10-13 3. 2020-10-13 4. 2023-10-13 5. 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.

## Proceso de pruebas de integración de la clase Cliente

totalSeguros():

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

VALORES:

|  |
| --- |
| 1. True 2. False 3. null |
| 1. [seguro0, seguro1, seguro2] 2. [] |

CASOS DE PRUEBA:

|  |
| --- |
| VALIDOS:  ([seguro0, seguro1, seguro2], true) : 1512€  ([seguro0, seguro1, seguro2], false) : 2016€  ([], false) : 0€ |
| NO VALIDOS:  ([], null) : No se puede implementar |

## Proceso de pruebas de integración de la interfaz

Interfaz:

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

VALORES:

|  |
| --- |
| 1. 12345678A 2. “” |

CASOS DE PRUEBA:

|  |
| --- |
| VALIDOS:  (12345678A)  (“”) |

## Proceso de pruebas unitarias de la clase ListaOrdenada

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

CASOS DE PRUEBA:

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

|  |  |  |
| --- | --- | --- |
| MÉTODO | CASOS VÁLIDOS | CASOS NO VÁLIDOS |
| get | (0, [1,2,3]) : 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]) : [2,3] | (-1, [1,2,3]) : IndexOutOfBoundsException  (0, []) : IndexOutOfBoundsException |
| size | [1,2,3] : 3  [] : 0 |  |
| clean | [1,2,3] : []  [] : [] |  |