INFORME PRÁCTICA 4: PRUEBAS UNITARIAS DE SOFTWARE

Jaime López-Agudo Higuera

1. Introducción

Este documento contiene la memoria de la práctica 4, esta contiene la información sobre el desarrollo de las diferentes pruebas realizadas (Caja negra, blanca y pruebas de integración).

Pruebas de caja negra

El proceso seguido para el desarrollo de las pruebas de caja negra es el siguiente:

1.-Desarrollo de las clases de equivalencia (válidas y no válidas).
2.-Creación de los casos de prueba válidos y no válidos.
3.-Implementación de los casos de prueba en código (JUnit).
4.-Comprobación de resultados de casos de prueba.
5.-Corrección de errores.

En la práctica se pide usar las técnicas de clases de equivalencia y AVL para obtener valores interesantes, de esta forma, de cada valor que queramos probar deberemos obtener sus valores límites y un valor medio.

Pruebas de caja blanca

En el caso de las pruebas de caja blanca, la forma de crearlas es la siguiente:

1.-Análisis de las pruebas de caja negra ya probadas para comprobar la cobertura actual (Usando Eclemma).
2.-Definir las pruebas de necesarias para completar la cobertura deseada.

En la práctica se pide usar una cobertura de decisión/condición, por lo que en cada una de las sentencias condicionales del código se deberá probar cada una de las decisiones dentro de las sentencias a verdadero y falso y probar la sentencia completa a verdadero y falso también.

Pruebas de integración

El proceso seguido para la realización de las pruebas de integración de la interfaz gráfica es la siguiente:

1.-Revisión de la interfaz y cambios en el código para poder usar la clase implementada Empleado.
2.-Comprobación manual de 2 o 3 casos de prueba para comprobar las funcionalidades básicas de la interfaz (botones, desplegables, cuadros de texto, radioButtons…).
3.-Automatización mediante FEST de todas las pruebas de caja negra realizadas en la anterior práctica.
4.-Ejecución de pruebas y corrección de errores.

2. Proceso de pruebas de la clase Empleado

1.-Pruebas de caja negra

Siguiendo la estructura de desarrollo de casos de prueba de caja negra expuesta en la introducción del documento, se han obtenido los siguientes apartados:

1.1.-Clases de equivalencia y valores interesantes identificados:

Parámetro	Clases Equivalencia Válidas	Clases Equivalencia No Válidas
Fecha	**1.[Hoy-5 años, Hoy]** **2.[Hoy-10 años, Hoy-5 años)** **3.[Hoy-20 años, Hoy-10 años)** **4.[Hoy-∞, Hoy-20 años)**	**7.{Hoy+1día, null}**
Categoría	**5.{DIRECTIVO/EJECUTIVO/OBRERO}**	**8.null, !{DIRECTIVO/EJECUTIVO/OBRERO}**
Baja	**6.{TRUE/FALSE}**	**9.null**

1.2.-Casos de prueba definidos (testCalculaSueldoBruto()) [Categoría, Fecha, Baja]:

CASOS DE PRUEBA VÁLIDOS

1.- [DIRECTIVO, Hoy, true]=1125€
2.- [GESTOR, Hoy-3 años, false]=1200€
3.- [OBRERO, Hoy-5 años+1día, true]=750€
4.- [DIRECTIVO, Hoy-5 años, false]=1550€
5.- [GESTOR, Hoy-7 años, true]=937.5€
6.- [OBRERO, Hoy-10 años+1dia, false]=1050€
7.- [GESTOR, Hoy-10años, true]=975€
8.- [DIRECTIVO, Hoy-15 años, false]=1600€
9.- [GESTOR, Hoy-20 años+1día, true]=975€
10.- [GESTOR, Hoy-20 años, false]=1400€
11.- [GESTOR, Hoy-50 años, true]=1050€

CASOS DE PRUEBA NO VÁLIDOS

12.- [GESTOR, Hoy+1día, true]
13.- [GESTOR, null, true]
14.- [null, Hoy, true]
15.- [PEPE, Hoy, true]
16.- [GESTOR, Hoy, null]
(15 y 16 no probables en código, comentadas dentro de la clase EmpleadoTest)

1.3.-Casos de prueba definidos (testDarDeAlta()) [Categoría, Fecha, Baja]:

CASOS DE PRUEBA VÁLIDOS

1.- [Directivo, Hoy, true]=se puede cambiar la baja a false.

CASOS DE PRUEBA NO VÁLIDOS

2.- [Directivo, Hoy, false]=no se puede cambiar la baja a false

1.4.-Casos de prueba definidos (testDarDeBaja()) [Categoría, Fecha, Baja]:

CASOS DE PRUEBA VÁLIDOS

1.- [Directivo, Hoy, false]=se puede cambiar la baja a true

CASOSO DE PRUEBA NO VÁLIDOS

2.- [Directivo, Hoy, true]=no se puede cambiar la baja a true

2.-Pruebas de caja blanca

Siguiendo los criterios de aplicación de las pruebas de caja blanca de la introducción, se ha obtenido lo siguiente:

2.1.-Criterios de cobertura

El criterio de cobertura usado será decisión/condición.

2.2.-Casos de prueba añadidos

En este caso no fue necesario añadir ningún caso nuevo para cumplir la cobertura, ya que cada una de las sentencias condicionales es evaluada en su totalidad:

1.-fechaContratacion==null→True: **Caso de prueba 13**
2.-fechaContratacion==null→False: **Casos de prueba 1-12 y 14-16**
3.-fechaContratación.isAfter(LocalDate.now())→True: **Caso de prueba 12**
4.-fechaContratación.isAfter(LocalDate.now())→False: **Casos de prueba 1-11 y 13-16**
5.-Decisión 1→True: **Casos de prueba 12,13,14,15,16**
6.-Decisión 1→False: **Casos de prueba 1-11**
7.-Directivo→True: **Casos de prueba 1,4,8**
8.-Directivo→False: **Casos de prueba 1,3,5,6,7,9,10,11,12,13,14,15,16**
9.-Gestor→True: **Casos de prueba 2,5,7,9,10,11**
10.-Gestor→False: **Casos de prueba 1,3,4,6,8**
11.-Obrero→True: **Casos de prueba 3,6**
12.-Obrero→False: **Casos de prueba 1,2,4,5,7,8,9,10,11,12,13,14,15,16**
13.-fechaContratación.isBefore(LocalDate.now().minusYears(5).plusDays(1))→True:
Casos de prueba 1,2,3
14.-fechaContratación.isBefore(LocalDate.now().minusYears(5).plusDays(1))→False:
Casos de prueba 4,5,6,7,8,9,10,11,13,14,15,16
15.-fechaContratación.isAfter(LocalDate.now().minusYears(10))→True:
Casos de prueba 1,2,3
16.-fechaContratación.isAfter(LocalDate.now().minusYears(10))→False:
Casos de prueba 4,5,6,7,8,9,10,11

17.- fechaContratación.isBefore(LocalDate.now().minusYears(10).plusDays(1))→True:
Casos de prueba 4,5,6
18.- fechaContratación.isBefore(LocalDate.now().minusYears(10).plusDays(1))→False:
Casos de prueba 1,2,3,7,8,9,10,11
19.-fechaContratación.isAfter(LocalDate.now().minusYears(20))→True:
Casos de prueba 4,5,6
20.-fechaContratación.isAfter(LocalDate.now().minusYears(20))→False
Casos de prueba 1,2,3,7,8,9,10,11
21.-fechaContratación.isBefore(LocalDate.now().minusYears(20).plusDays(1))→True:
Casos de prueba 10,11
22.-fechaContratación.isBefore(LocalDate.now().minusYears(20).plusDays(1))→False:
Casos de prueba 1,2,3,4,5,6,7,8,9
23.-baja→True: **Casos de prueba 1,3,5,7,9,11,12,13,14,15**
24.-baja→False: **Casos de prueba 2,4,6,8,10**

2.3.-Aclaraciones

En el caso de la cobertura de condición, no se puede cumplir al 100% con el código, porque `if(fechaContratación.isAfter(LocalDate.now()) ||fechaContratación==null)` no se puede completar de las 4 formas distintas: (True,True), (False,True), (False,False), (True,True)

3. Proceso de pruebas de la clase EmpleadosGUI

1.-Pruebas de integración

Las pruebas de integración de la interfaz gráfica serán las mismas realizadas durante las pruebas válidas de caja negra, más unas pruebas de comprobación de aspecto de la interfaz:

```
demo.textBox("txtFechaContratacion").requireText("dd-mm-yyyy");
demo.button("btnCalcular").requireText("CALCULAR");
```

2.-Aclaraciones

En el código se ha implementado todas las pruebas de caja negra de la parte anterior (EmpleadoTest), pero en este caso solo sería necesario hacer una prueba por cada caso en que al aplicación tuviese que recalcular el sueldo, es decir, tenemos que comprobar todas categorías (Directivo/ejecutivo/obrero), así como las fechas en las que el sueldo tiene modificaciones, (fecha>5 años, fecha>10 años y fecha>20 años) y las bajas con true o false, con lo que las únicas pruebas que serían necesarias son las siguientes:
-Categoría=Obrero, fecha Contratación=hoy-10 años + 1 día, baja=false (Caso de prueba 6).
-Categoría=Directivo, fechaContratación=hoy-15 años, baja=false (Caso de prueba 8).
-Categoría=Gestor, fechaContratación=hoy-50 años, baja=true (Caso de prueba 11).
De esta forma tendremos todos los casos que pueden ser problemáticos, y no harán falta todos los casos implementados en código, solo los numero 6, 8 y 11.

4. Proceso de pruebas de la clase ListaOrdenadaAcotada

1.-Pruebas de caja negra

Las pruebas de caja negra realizadas para la clase ListaOrdenadaAcotada so las siguientes (pruebas aplicadas después de corregir el código de la clase).

1.1.-Clases de equivalencia y valores interesantes identificados:

	Clases de equivalencia válida	Clases de equivalencia no válida
Add(E e)	**1.-Añadirlo con lista vacía** **2.-Añadirlo 1 elemento mayor** **3.-añade 1 elemento menor o igual ERROR**	**14.-e=null ERROR** **15.-Lista llena**
Get(pos)	**4.-Lista vacía** **5.-Lista con 1 elemento** **6.-Lista completa**	**16.-pos<0**
Size()	**7.-Lista vacia** **8.-Lista con 1 elemento** **9.-Lista llena**	
Remove(pos)	**10.-Lista con 1 elemento** **11.-Lista con varios elementos**	**17.-Lista vacía**
Clear(E e)	**12.-Lista vacía** **13.-Lista con varios elementos ERROR**	

1.2.-Casos de prueba definidos (estado de lista, método ejecutado→comprobación)

CASOS DE PRUEBA VÁLIDOS

1.- {}, LISTA.ADD(1)→LISTA.GET(0)==1

2.- {} , LISTA.ADD(1);LISTA.ADD(2)→LISTA.GET(0)==1 &&LISTA.GET(1)==2

3.- {1}, LISTA.ADD(1);LISTA.ADD(0)→LISTA.GET(0)==0 &&LISTA.GET(1)==1 ERROR ADD()

4.- {}, LISTA.GET(0) →LISTA.GET(0)==null

5.- {0} → LISTA.GET(0)==0

6.- {0,1,2} →LISTA.GET(0)==0, LISTA.GET(1)==1, LISTA.GET(2)==2

7.- {} →LISTA.SIZE()==0

8.- {0} → LISTA.SIZE()==1

9.- {0,1,2} →LISTA.SIZE()==3

10.- {0}, LISTA.REMOVE(0)→LISTA.SIZE()==0

11.- {1,2}, LISTA.REMOVE(0), LISTA.REMOVE(0)→LISTA.SIZE==0

12.- {}, LISTA.CLEAR()→LISTA.SIZE()==0

13.- {0,1,2}, LISTA.CLEAR()→LISTA.SIZE()==0 &&lista.get(0)==null&& lista.get(1)==null&& lista.get(2)==null ERROR CLEAR (AssertTrue(lista.get(0)==null), antes e corrección de ListadOrdenadaAcotada

CASOS DE PRUEBA NO VÁLIDOS

14.- LISTA.ADD(NULL) →NullPointerException() ERROR con implementación original dejaba hacerlo, ahora tira NullPointerException

15.- {1,2,3}, LISTA.ADD(4) →IllegalStateException()

16.- {0,1,2}, LISTA.GET(-1)→IndexOutOfBoundsException()

17.- {}, LISTA.REMOVE(0) →IndexOutOfBoundsException()

2.-Pruebas de caja blanca

2.1.-Criterios de cobertura

El criterio de cobertura usado será decisión/condición.

2.2.-Casos de prueba añadidos

En este caso no se alcanza la cobertura deseada, porque no se ha creado la lista con el valor:

```
public final static int MAX_POR_OMISION = 10;
```

Por lo que se ha de añadir una prueba en la que se cree la lista con la longitud predeterminada, de forma que la prueba quede así:

18.- {}, LISTA.ADD(0,…,9) →LISTA.SIZE()==10

19.- {0,1,2,3,4,5,6,7,8,9}, LISTA.ADD(10) → IllegalStateException()