INFORME PRÁCTICA 4: PRUEBAS UNITARIAS DE SOFTWARE

1. Introducción

En esta práctica vamos a realizar tanto pruebas unitarias como pruebas de integración para una aplicación que lleva la gestión de las nóminas de los trabajadores de una empresa. Para ello realizaremos pruebas unitarias tanto pruebas de caja blanca como de caja negra para una clase parcialmente dada.

Además, realizaremos pruebas de integración para la GUI de la aplicación mediante FEST, así como pruebas unitarias adicionales para la clase ListaOrdenadaAcotada.

2. Proceso de pruebas de la clase Empleado

Caja negra

Primero hemos diseñado una tabla utilizando la técnica de partición equivalente para el método sueldoBruto() de un empleado. En ella, distinguimos 3 parámetros para determinar el sueldo del empleado: *Categoría*, *FechaContratación* y *Baja*. Dadas las tablas 1 y 2, podemos distinguir distintas clases de equivalencia para cada parámetro, estas son aquellos datos de entrada que generan el mismo comportamiento lógico en el programa. Los rangos están expresados en años.

Parámetros	Clases válidas	Clases no válidas
Categoría	1. DIRECTIVO 2. GESTOR 3. EMPRESARIO	10. null
FechaContratación	4. [hoy, hoy-5] 5. (hoy-5, hoy-10] 6. (hoy-10, hoy-20) 7. (hoy-20, -∞)	11. null 12. (hoy, ∞)

Baja	8. true 9. false	

Después hemos realizado AVL (Análisis de Valor Límite) con las clases anteriores para determinar que los rangos de valores programados en la función retornan los resultados esperados. Hemos aplicado la técnica 1-wise, la cuál consiste en que el valor interesante de cada parámetro se incluye en al menos un caso de prueba. Los casos de prueba no abordados aquí los completaremos posteriormente con las pruebas unitarias de caja blanca.

Encontramos los siguientes valores diferentes para cada parámetro, incluyendo tanto los límites como un valor intermedio en los casos de rango:

Válidos

1. DIRECTIVO 2. GESTOR 3. OBRERO

4. hoy, hoy-3, hoy-5 5. hoy-6, hoy-7, hoy-10

6. hoy-11, hoy-15, hoy-20 7. hoy-21, hoy-50

8. true 9. false

No válidos

10. null

11. null

12. hoy + 10

Probaremos las clases válidas de forma que haya al menos un valor de cada en los distintos casos de prueba, y comprobaremos que el resultado obtenido de la función sea el correcto según las tablas 1 y 2. El criterio de cobertura a utilizar es el *1-wise,* esto es establecer casos de prueba en los que el dominio de las variables aparezcan al menos en un caso de prueba.

Casos de prueba válidos

Cargo	FechaContratación	Baja	Valor esperado
DIRECTIVO	hoy	no	1500 €
GESTOR	hoy-3	si	900 €
OBRERO	hoy-5	no	100 €
OBRERO	hoy-6	no	150 €
GESTOR	hoy-7	no	1250 €
DIRECTIVO	hoy-10	no	1550 €
OBRERO	hoy-11	no	200 €
GESTOR	hoy-15	no	1300 €
DIRECTIVO	hoy-20	no	1600 €
OBRERO	hoy-21	no	300 €
GESTOR	hoy-50	no	1400 €

Casos de prueba no válidos

Cargo	FechaContratación	Baja
null	hoy	no
GESTOR	null	no
OBRERO	hoy+5	no

Una vez definidos los casos de prueba, programaremos los tests de caja negra en Junit sin escribir aún el código de la clase Empleado, sólo la estructura. Estos test se encuentran definidos en EmpleadoTest. Además hemos definido los casos no válido para el constructor de Empleado, el cuál no debe contener ningún parámetro null, o una fecha de contratación posterior al día de hoy.

Una vez realizados los test, ejecutamos un test de cobertura para ver cuánto código de la clase Empleado se ha ejecutado. Comprobamos que la cobertura es del 82.5% con los tests unitarios de caja negra.

Una vez realizados los tests, prodeceremos a realizar la clase Empleado con el funcionamiento deseado. Además, hemos añadido unos *getters* y *setters* para el manejo de los atributos, así como un par de constructores adicionales para inicializar un Empleado con atributos por defecto, es decir, fecha de contratación igual al dia de hoy y baja igual a *false*.

Después procederemos con los tests de caja blanca.

Caja blanca

Tras comprobar la cobertura de los tests unitarios de caja negra, necesitaremos completar la cobertura no alcanzada. Para ello, tenemos que aplicar la técnica de cobertura de decisión/condición.

Basados en el test cobertura previamente realizado de la caja negra, nos fijamos en qué líneas de código no se ejecutan, por lo que haremos test que lleguen a esas sentencias, estos son, los tests para añadir un empleado de cada tipo correcto con los constructores auxiliares (2 y 4 parámetros), y los tests para los métodos adicionales *getters* y *setters*, a parte del *compareTo()* .

3. Proceso de pruebas de la clase EmpleadosGUI

Para desarrollar las pruebas de la clase EmpleadosGUI, hemos hecho uso de FEST, que es un conjunto de librerías que permiten la automatización de pruebas de interfaces gráficas de tipo Java Swing.

Hemos empezado comprobando el aspecto inicial de la interfaz, esto es, los botones y el texto de las etiquetas. Después, hemos comprobado los valores por defecto de la interfaz (“dd/mm/yyyy”, boton baja no seleccionado, caja de texto de sueldo vacía y el selector de categoría en DIRECTIVO).

Después hemos usado FEST para introducir un usuario de cada tipo de contrato, estos son, Directivo, Gestor y Obrero con las fechas entre el rango correspondiente para hallar los valores esperados.

Finalmente hemos añadido que no permita introducir una fecha vacía o con formato incorrecto.

4. Proceso de pruebas de la clase ListaOrdenadaAcotada

Caja negra

Empezamos creando una lista correcta con 3 elementos, y comprobamos que la primera posición de la lista sea el último elemento añadido y el resto de elementos tenían valor nulo.

Después, implementamos la clase Empleado como Comparable, generando el método compareTo(), de forma que la lista siempre añadiese en la primera posición. Comprobamos los atributos de los elementos de la lista extraídos con get() con sus valores correspondientes.

Los siguientes métodos (get para posiciones negativas, get para posiciones >= size, remove para posiciones negativas y remove para posiciones >=size) funcionaban correctamente.

También comprobamos añadía hasta la última posicion correctamente, y llenar una lista con 100 elementos también funcionaba.

La lista permitía añadir elementos nulos por lo que no era un comportamiento deseado.

La cobertura con la caja negra es del 85.9%.

Caja blanca

Fuimos a la clase corregida, y corregimos que no se pueda añadir elemento nulo. También añadimos los casos de igualdad en el while (para guardar índice) y for de la inserción (para desplazar elementos hacia delante) del método add().

Después desarrollamos en la clase Empleado la función compareTo(), en la que ordenamos por Categorías, y dentro de estas por orden alfabético. Después cambiamos los parámetros del test de caja negra para que también cumplan el orden establecido.

Finalmente creamos el tests de caja blanca en el cuál comprobamos el orden de la lista con distintos casos de empleado. También añadimos varios casos para el constructor de la lista (crear una lista sin parámetro de 10 elementos, añadir un elemento a la lista llena).

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

La cobertura conseguida final es del 100%.