Práctica 3
Pruebas dinámicas de código con JUnit

Javier Nogueras Iso y Miguel Ángel Latre
Escuela de Ingeniería y Arquitectura
Departamento de Informática e Ingeniería de Sistemas

1. Introducción

Los objetivos de esta práctica son los siguientes:

1. Profundizar en la utilización de la técnica de particiones de equivalencia, aplicándola al contenido de ficheros de texto.
2. Realizar pruebas de unidad de los métodos y clases de un pequeño programa.
3. Realizar pruebas de sistema sobre el mismo programa.
4. Utilizar la herramienta JUnit para la escritura y ejecución de pruebas en Java.
5. Utilizar mecanismos básicos de suplantación de clases para realizar pruebas de sistema con JUnit.

JUnit es un entorno para la escritura y ejecución de pruebas en Java. Esencialmente, está pensado para escribir conjuntos de pruebas de unidad que puedan ejecutarse de forma reiterada. En esta práctica, donde trabajaremos con un programa pequeño cuya interfaz con el usuario está basada en la utilización de la terminal, también lo utilizaremos para realizar pruebas de sistema.

El guion de esta práctica está escrito para utilizar JUnit 5 en el entorno de desarrollo IntelliJ (o Android Studio).

2. Descripción del sistema a probar durante esta práctica

Se desea disponer de un programa Java que, al iniciar su ejecución, solicite al usuario el nombre de un fichero de texto. Una vez que el usuario lo ha introducido, se muestra en la pantalla una lista de las letras del alfabeto español y el número de veces que dicha letra aparece en el fichero introducido.

Se muestra a continuación un ejemplo de ejecución del programa solicitado:

```
Nombre de un fichero de texto: src/main/res/quijote.txt

A:  200495
B:   24147
C:   59436
D:   87237
E:  229189
F:    7581
G:   17225
H:   19920
I:   90075
J:   10530
K:       0
L:   89141
M:   44658
N:  108441
Ñ:    4241
O:  162514
P:   35465
Q:   32483
R:  100954
S:  125727
T:   61749
U:   79559
V:   17855
W:       2
X:     377
Y:   25115
Z:    6491
```

En el ejemplo anterior, el texto introducido por el usuario a través del teclado aparece subrayado y en negrita, mientras que lo escrito en la pantalla por el programa aparece en redonda.

2.1. Detalles adicionales

- No debe distinguirse entre versiones mayúsculas y minúsculas de la misma letra.
- Nótese la presencia de la letra Ñ en los resultados mostrados.
- El resto de apariciones de letras voladas y letras con diacríticos (acentos agudos, graves, circunflejos, diéresis y cedillas) deben considerarse como ocurrencias de la letra sin modificaciones.

2.2. Material disponible para esta práctica

El enunciado de esta práctica se complementa en GitHub con el repositorio `https://github.com/miguel-latre/unizar-vv-practica3`.

En el proyecto IntelliJ de dicho repositorio (que ha sido configurado para que también pueda abrirse, compilarse y ejecutarse con Android Studio), el directorio del proyecto «`src/main/java`» es el directorio base de código fuente del programa que se solicita, y se han dejado en él los esqueletos de dos clases para realizar la práctica. En el directorio «`src/main/res`» se ubican tres ficheros de texto («`quijote.txt`», «`regenta.txt`» y «`hamlet.txt`»), codificados en UTF-8. El directorio «`src/test/java`» es el directorio base del código de pruebas y el directorio «`src/test/res`» contiene ficheros cuyo contenido es el texto que el programa debería mostrar en pantalla para los ficheros contenidos en «`src/main/res`», mencionados anteriormente, también codificados en UTF-8.

Los ficheros «`quijote.txt`», «`regenta.txt`» y «`hamlet.txt`» se han obtenido de la página web Project Gutenberg[1] y los ficheros originales descargados, que incluyen metadatos e información de licencia, se encuentran en el directorio «`originalesProjectGutenberg`».

3. Actividades a realizar en la práctica

3.1. Descarga o clonación del repositorio

Descargad el repositorio o clonadlo a un repositorio nuevo. Abridlo con IntelliJ o Android Studio y familiarizaos con su estructura, así como con el código de partida y los recursos proporcionados.

Observad como en el esqueleto del código, se proporcionan dos clases (`Main` y `ContadorDeLetras`) con el objetivo de descomponer el programa solicitado de forma tal que los métodos de la clase `Main` sean los únicos que se encarguen de la interacción con el usuario a través del terminal. Para ello, deberán hacer uso de los métodos de la clase

[1] `http://www.gutenberg.org/`

`ContadorDeLetras`: un constructor que asocie nuevos objetos de la clase `ContadorDeLetras` a ficheros de texto concretos, y el método `frecuencias` que devuelve un vector con el número de apariciones de cada letra del alfabeto español.

3.2. Diseño de pruebas de unidad

Vais a realizar pruebas de unidad sobre el método `frecuencias` de la clase `ContadorDeLetras`, cuya especificación se muestra a continuación:

```
/**
 * Si no ha sido analizado ya, analiza el contenido del fichero de texto
 * asociado a este objeto en el constructor. Devuelve un vector de 27
 * componentes con las frecuencias absolutas de aparición de cada letra del
 * alfabeto español en el fichero.
 *
 * @return vector de 27 componentes de tipo entero. Las primeras 26
 *         componentes almacenan el número de apariciones de las 26 letras del
 *         alfabeto inglés: la componente indexada por 0 almacena el número de
 *         apariciones de la letra A, la componente indexada por 1, el de la
 *         letra B y así sucesivamente. La última componente, almacena el
 *         número de apariciones de la letra Ñ.
 * @throws FileNotFoundException
 *             si el fichero de texto que se especificó al construir este
 *             objeto no existe o no puede abrirse.
 */
public int[] frecuencias() throws FileNotFoundException;
```

Para ello, y **antes** de escribir su código, **diseñad los casos de prueba** para dicho método, a partir de su especificación, la especificación del constructor de la clase en la que se define y de las consideraciones que se realizan en el enunciado del problema, en particular en la sección 2. Utilizad para el diseño de estas pruebas las técnicas de particiones de equivalencia y de análisis de valores límite. Aplicadlas tanto a los ficheros como continente como a su contenido:

- **Ficheros como continente:** referencias a objetos de la clase `File` que representan ficheros existentes, no existentes, o que son nulas (**`null`**).
- **Contenido de los ficheros:** Número de caracteres de los ficheros, caracteres que deben ser contados y que no, etc.

  La sección 2.1 te puede ser útil también para definir clases de equivalencia del contenido de los ficheros.

Tened en cuenta que el material suministrado con esta práctica (los ficheros «`quijote.txt`», «`regenta.txt`» y «`hamlet.txt`») será útil para definir tres pruebas concretas, pero que únicamente ejercitarán una misma clase de equivalencia. Existen otras que tenéis que identificar y para las que tenéis que preparar datos de prueba.

Se ha dejado disponible un documento Google[2] que os puede servir como plantilla. Podéis descargarlo en un formato editable o hacer una copia en vuestra unidad Drive.

Cuando tengais el diseño de las pruebas hecho, solicitad realimentación al profesor en la sesión de prácticas. Cuando sea necesario, los resultados esperados por cada caso de prueba pueden establecerse de forma indirecta haciendo referencia a los ficheros facilitados junto con el guion de esta práctica.

3.3. Escritura del programa

Escribe el programa solicitado, respetando la descomposición propuesta en la que la clase `Main` es la única que se encarga de la interacción con el usuario a través del terminal y los métodos de la clase `ContadorDeLetras` de asociarse a ficheros de texto concretos y calcular el número de apariciones de cada letra del alfabeto español.

Se recomienda realizar esta tarea intercalándola con la siguiente (la escritura de los tests de unidad).

3.4. Escritura y ejecución de pruebas de unidad

Escribe una clase de pruebas en JUnit para el método `frecuencias`. Para realizar las pruebas, puedes utilizar los documentos suministrados con este guion, pero como se ha comentado anteriormente, no serán los únicos que tengas que utilizar.

JUnit 5 proporciona un amplio conjunto de *aserciones* (métodos de JUnit 5 para comprobar el cumplimiento o no de determinadas condiciones al ejecutar las pruebas) a través de la clase `org.junit.jupiter.api.Assertions`. Veremos muchas de ellas a lo largo del curso, aunque para esta práctica vamos a necesitar únicamente las siguientes[3]:

- `public static void assertEquals(Object expected, Object actual)`

  Comprueba que los objetos `expected` y `actual` son iguales. Si no lo son, lanza un error de tipo `AssertionFailedError`. Si `expected` y `actual` son ambos nulos, considera que son iguales.

  El API de JUnit también define métodos `assertEquals` sobrecargados para comparar datos de todos los tipos primitivos de Java.

- `public static void assertArrayEquals(int[] expected, int[] actual)`

  Comprueba que los vectores `expected` y `actual` son iguales, tanto en longitud como en contendio. Si no lo son, lanza un error de tipo `AssertionFailedError`.

[2] `https://docs.google.com/document/d/1Zyf366tQ4tAUhT9JL1dk1oT-NOUWeoboiFe6qBDPneU/`

[3] `https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html`

Si `expected` y `actual` son ambos nulos, considera que son iguales.

El API de JUnit también define métodos `assertArrayEquals` sobrecargados para comparar vectores de datos de todos los tipos primitivos de Java y vectores de objetos.

- `public static <T extends Throwable> T assertThrows(Class<T> expectedType, Executable executable)`

  Comprueba que la ejeución de `executable` lanza una excepción de la clase `expectedType` y devuelve el objeto correspondiente a la excepción lanzada. Habitualmente, el argumento para el parámetro `executable` se especifica como una expresión lambda. Si `executable` no lanza ninguna excepción, o si lanza una excepción de una clase distinta a la esperada, el propio método `assertThrows` lanza un error de tipo `AssertionFailedError`.

  Si no se desea realizar ninguna comprobación adicional con el objeto de la excepción lanzada, el valor devuelto puede ser ignorado.

  Por ejemplo, la siguiente invocación al método `assertThrows` comprobaría que el código que se especifica lanza una excepción de la clase `NullPointerException`:

  ```
  assertThrows(NullPointerException.class, () -> {
      String str = null;
      str.length();
  });
  ```

3.4.1. Notas sobre la escritura de las pruebas

Si las pruebas encuentran defectos en vuestro código y no sois capaces de localizarlos en el código fuente, una buena estrategia consiste en escribir más pruebas para el propio método `frecuencias` o para otros métodos que puedan ser invocados desde él.

Para implementar el método `frecuencias`, puede ser útil saber que, si `linea` es un objeto de la clase `String`, la siguiente expresión devuelve otra cadena en la que los diacríticos de los caracteres que la componen han sido eliminados:

```
Normalizer.normalize(linea, Form.NFD)
      .replaceAll("\\p{InCombiningDiacriticalMarks}+", "")
```

Desafortunadamente, la expresión anterior también elimina la virgulilla de la letra Ñ, convirtiéndola en la letra N.

3.5. Realización de pruebas de sistema

Una vez realizadas pruebas de unidad sobre el método `frecuencias`, ahora podéis realizar pruebas sobre el método `main` de la clase `Main` de vuestro programa. Los casos de prueba de

sistema debéis implementarlos en una clase diferente a la que contenía las pruebas de unidad.

En cualquier caso, los casos de prueba van a ser muy parecidos a los de la sección anterior, aunque la diferencia principal reside en que ahora se está probando el funcionamiento de un método que interactúa con el usuario a través de la consola. Por tanto, para poder hacer pruebas automáticas, va a ser necesario redefinir la entrada estándar de forma que se simule la escritura desde teclado con los datos adecuados para cada caso de prueba, y redirigir la salida estándar para poder analizar posteriormente el contenido de la misma. Las redirecciones de la entrada y la salida las podemos hacer utilizando los métodos `System.setIn` y `System.setOut`.

En el caso que nos ocupa, para simular la entrada desde teclado se pueden utilizar objetos de la clase `ByteArrayInputStream`, construidos a partir del vector de *bytes* asociado a la cadenas de caracteres que queramos establecer como entrada proporcionada por el usuario (ved el método `String.getBytes`).

En cuanto a la salida, en este caso nos va a interesar redirigirla a un fichero temporal que, tras la ejecución del método `main`, compararemos con el fichero correspondiente de la carpeta «`resultado`» con el objeto de averiguar si son idénticos. El proceso de redirección va a consistir siempre en crear un nuevo objeto de la clase `PrintStream`, almacenar su referencia y utilizarla como argumento para el método `System.setOut`. En JUnit 5, este tipo de preparación común a la ejecución de cualquier método anotado con `@Test` puede escribirse en métodos encabezados con la etiqueta `@BeforeEach`. Análogamente, las acciones comunes que sea necesario realizar tras la ejecución de cada prueba pueden ir en métodos encabezados con la anotación `@AfterEach`.

La semántica de estas anotaciones en JUnit indican que, **antes** de la ejecución de **cada uno** de los métodos anotados con `@Test`, se ejecutan todos los métodos anotados con `@BeforeEach` y, **después** de **cada uno** de los métodos anotados con `@Test`, se ejecuten todos los métodos anotados con `@AfterEach`.

3.5.1. Preparación del entorno de pruebas

En JUnit 5, existen dos anotaciones más, `@BeforeAll` y `@AfterAll`, para encabezar métodos estáticos que, respectivamente, deban ejecutarse **una sola vez antes** del inicio de la ejecución del conjunto de pruebas definidas por la clase y **una sola vez después** de la ejecución del conjunto de pruebas definidas por la clase.

En esta práctica, se pide que creéis dos de estos métodos:

- Un método `@BeforeAll` para guardar las referencias originales a los objetos `System.in` y `System.out`, que se perderían si no se salvaguardan conforme se van realizando las pruebas.
- Un método `@AfterAll` para restaurar las referencias a la entrada y la salida originales.

3.6. Ejecución de baterías de pruebas

Llegados a este punto, disponéis de dos clases Java diferentes que definen pruebas en JUnit. Desde IntelliJ o Android Studio, podéis ejecutar de forma separada las pruebas de unidad definidas por una de las clases o las de sistema definidas por la otra.

En estos entornos, también es muy sencillo ejecutar automáticamente ambos conjuntos de pruebas. Basta con seleccionar el directorio o el paquete en el que se encuentran y en el menú contextual, elegir la opción *«Run 'All Tests'»* o *«Run 'Tests in 'es.unizar.eina.vv6f.practica3"»*, dependiendo del elemento que se haya seleccionado.

4. Entrega de resultados de la práctica

Esta práctica no está directamente vinculada al trabajo de la asignatura, pero va a ser realizada igualmente en grupo, en los mismos equipos que para el trabajo de la asignatura.

Como resultado de esta práctica se entregarán los siguientes documentos:

- La versión definitiva del diseño de las pruebas al que se refiere la sección 3.2, una vez recibida la realimentación del profesor. Debe proporcionarse un documento que incluya la identificación de particiones de equivalencia y el diseño de los casos de prueba. Cuando sea necesario, los resultados esperados por cada caso de prueba pueden establecerse de forma indirecta haciendo referencia a los ficheros facilitados junto con el guion de esta práctica. En el mismo documento se hará constar los nombres de los autores de la práctica. El formato del fichero puede ser de texto, PDF, o los correspondientes a Microsoft Office o LibreOffice. También puede proporcionarse un enlace a un documento de Google Docs al que se le haya dado acceso a los profesores de la asignatura.
- Un fichero comprimido en formato ZIP (o un fichero de texto con un enlace a un repositorio GitHub) que contenga el proyecto correspondiente a esta práctica (código fuente del programa solicitado, código fuente de las pruebas en JUnit y ficheros de configuración del proyecto) solicitado en las tareas 3.1, 3.3 y 3.4 y 3.5.

La fecha límite de entrega será el día anterior a la siguiente sesión de prácticas.