Práctica 6. Pruebas

Grado en Ingeniería Informática - Ingeniería del Software
Dpto. de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

1. Introducción y objetivos

En esta práctica deberéis **diseñar, construir y ejecutar casos de pruebas que permitan detectar errores** en la aplicación de gestión de *Notas*. Para ello, realizaréis:

- pruebas unitarias de caja negra,
- pruebas de volumen
- y pruebas de sobrecarga.

Para realizar esta práctica es necesario disponer del código de la aplicación de gestión de *Notas* en su versión número 3, es válido tanto el código de la práctica 4 como el de la práctica 5.

2. Actividades a realizar en la práctica

2.1. Pruebas unitarias de caja negra

2.1.1. Diseño de las pruebas

Diseñad, utilizando la técnica de particiones de equivalencia, casos de pruebas para los siguientes métodos de la clase *NotesDbAdapter*, de acuerdo con las siguientes especificaciones:

```java
/**
 * Crea una nueva nota a partir del título y texto proporcionados. Si la
 * nota se crea correctamente, devuelve el nuevo rowId de la nota; en otro
 * caso, devuelve -1 para indicar el fallo.
 *
 * @param title
 *            el título de la nota;
 *            title != null y title.length() > 0
 * @param body
 *            el texto de la nota;
 *            body != null
 * @return rowId de la nueva nota o -1 si no se ha podido crear
 */
public long createNote(String title, String body){
            ....
}

/**
 * Borra la nota cuyo rowId se ha pasado como parámetro
 *
 * @param rowId
 *            el identificador de la nota que se desea borrar;
 *            rowId > 0
 * @return true si y solo si la nota se ha borrado
 */
public boolean deleteNote(long rowId){
            ....
}

/**
 * Actualiza una nota a partir de los valores de los parámetros. La nota que
 * se actualizará es aquella cuyo rowId coincida con el valor del parámetro.
 * Su título y texto se modificarán con los valores de title y body,
 * respectivamente.
 *
 * @param rowId
 *            el identificador de la nota que se desea borrar;
 *            rowId > 0
 * @param title
 *            el título de la nota;
 *            title != null y title.length() > 0
 * @param body
 *            el texto de la nota;
 *            body != null
 * @return true si y solo si la nota se actualizó correctamente
 */
public boolean updateNote(long rowId, String title, String body){
            ....
}
```

2.1.2. Construcción y ejecución de las pruebas

Escribid el código necesario para ejecutar de forma automática los casos de prueba diseñados en la sección anterior en la aplicación de gestión Notas. Para ello, se sugiere tener en cuenta lo siguiente:

- Escribid el código correspondiente a los casos de prueba en una clase distinta a *NotesDbAdapter* (por ejemplo, en una clase denominada *Test*).
- Tenéis que tener en cuenta que, en ocasiones, vais a realizar invocaciones a los métodos que se prueban con valores deliberadamente erróneos. **Si queréis que la ejecución automática de las pruebas no se interrumpa, deberéis capturar cualquier excepción o error que se produzca al ejecutar los métodos**. Una posible estrategia es capturar cualquier instancia de la clase *Throwable* que puedan lanzarse desde los métodos invocados en las pruebas.
- Los resultados de la ejecución de cada caso de prueba se pueden registrar utilizando el método `d()` de la clase *android.util.Log*. Para cada caso de prueba, se sugiere registrar una identificación del mismo y su resultado de ejecución (si es correcto, incorrecto o si por el contrario se ha producido una excepción).
- Para iniciar la ejecución automática de las pruebas, incluid una nueva opción en el menú de la aplicación.

2.1.3. Análisis de los resultados y corrección de los errores detectados

Ejecutad las pruebas y analizad los resultados. *¿Funcionan los casos de prueba correspondientes a particiones de equivalencia válidas? ¿Qué errores aparecen con los de particiones de equivalencia no válidas, en el caso de que se produzcan?*

Existe un conjunto de casos de prueba para los que la ejecución de las pruebas no dará ningún error explícito, pero las notas resultantes de la ejecución de los mismos no podrán ser borradas por un usuario desde la interfaz de la aplicación. ¿Es esto un error? ¿Por qué?

Realizad las modificaciones necesarias en el código para corregir los errores detectados.

2.2. Prueba de sistema: Prueba de volumen

Las pruebas de volumen son pruebas de sistema consistentes en examinar el funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos. El volumen de datos se simula a través de las cargas de trabajo esperadas del sistema.

2.2.1. Diseño de la prueba

Debéis diseñar una prueba de volumen utilizando la técnica de clases de equivalencia considerando un nuevo requisito:

> ***No se espera que haya más de 1000 notas almacenadas en la aplicación.***

2.2.2. Construcción y ejecución de la prueba

Escribid un método que cree notas de forma masiva. Haced que el título de cada nota tenga un prefijo común y que incluya un número distinto consecutivo. Puede ser buena idea desarrollar también otro método que, de forma automática, identifique y borre las notas creadas por el primero.

Igual que en el caso anterior, añadid una nueva opción de menú que permita ejecutar los casos de prueba diseñados de forma automática.

2.2.3. Análisis de los resultados y corrección de los errores detectados

Ejecutad las pruebas y analizad los resultados. *¿Funcionan los casos de prueba correspondientes a particiones de equivalencia válidas? ¿Y los de particiones de equivalencia no válidas?*

Como parte del análisis, ejecutad como usuarios los siguientes escenarios cuando se hayan creado un gran número de notas:

- Cread una nota nueva y, una vez creada, modificadla.
- Desplazaos hasta la nota número 75, editadla, confirmad los cambios y, a continuación, editad también la nota número 76.

¿Se ha producido algún error? ¿Ha aparecido algún problema de usabilidad?

Corregid este problema de usabilidad añadiendo un atributo a la clase *Notepadv3* que almacene la posición que ocupa en la lista la última nota creada, editada o visualizada, teniendo en cuenta además lo siguiente:

- El atributo privado *mList* de la clase *Notepadv3* es de tipo *ListView* y permite acceder a la lista de notas que se muestran en la actividad. El método `setSelection()` de la clase *ListView* permite actualizar la posición seleccionada de la lista, y desplazar la parte visible de la lista hacia esa posición.
- Con anterioridad al lanzamiento de la actividad *NoteEdit* para la creación de una nota, con la invocación `mList.getCount()` se puede averiguar cuántos elementos

hay en la lista de notas. Este número será la posición de la nota nueva, siempre y cuando los títulos de las notas creadas sigan un orden alfabético.

- Cuando se edita o se elimina una nota, se invoca al método `onContextItemSelected()`, cuyo parámetro *item* (de tipo *MenuItem*) permite acceder a la posición en la que se encuentra la nota seleccionada dentro de la lista de notas a través de la invocación a `((AdapterView.AdapterContextMenuInfo)item.getMenuInfo()).position`.

2.3. Prueba de sistema: Pruebas de sobrecarga

Las pruebas de sobrecarga o estrés son pruebas de sistema que consisten en comprobar la robustez del mismo, haciéndolo trabajar por encima del umbral límite de su capacidad a través de cargas masivas de trabajo o datos.

El objetivo es establecer los puntos extremos en los cuales el sistema empieza a operar por debajo de los requisitos establecidos o directamente no opera, así como determinar la forma en que deja de operar (tipos de errores que se producen, posibilidad de recuperación, etc.).

2.3.1. Diseño, construcción y ejecución de una prueba de sobrecarga de datos

Vais a diseñar una prueba en la que averiguar la longitud máxima que el texto de una nota es capaz de soportar, en cuanto a número de caracteres.

Para ello, cread un método que vaya insertando notas cuyo texto tenga longitudes crecientes y que registre a través de la clase *android.util.Log* el tamaño del texto insertado. No incrementéis el número de caracteres del texto de la nota que se inserta de uno en uno, pues sería demasiado lento. El método debe terminar cuando la inserción de una nota falle.

Observad el registro del `«LogCat»` y averiguad el tamaño de la última nota creada antes de fallar. Tratad de seguir usando la aplicación y averiguad si la causa del fallo en la inserción de última nota ha sido por el tamaño de la misma o ha sido por otra causa. El resultado de vuestras averiguaciones tendréis que plasmarlo en la documentación entregada en esta práctica.

Es posible que, tras la ejecución de esta prueba, tengáis que borrar manualmente todo el contenido de la aplicación de gestión *Notas* desde el gestor de aplicaciones de Android, desinstalarla completamente o incluso volver a crear el dispositivo virtual con el que estéis trabajando. En cualquier caso, **se recomienda encarecidamente no realizar esta prueba en un dispositivo real**.

2.3.2. Ejecución de una prueba de sobrecarga generando eventos de usuario

En esta prueba vais a utilizar `monkey` [1], una herramienta del SDK de Android que se ejecuta tanto en emulador como en terminales reales. Utilizaremos `monkey` cuando queramos realizar pruebas de sobrecarga simulando el comportamiento de un usuario de manera aleatoria pero reproducible, ya que `monkey` genera eventos de usuario, por ejemplo clicks de ratón. `Monkey` también genera eventos del sistema.

`Monkey` se utiliza en la terminal a través de la orden `adb`, con la siguiente sintaxis:

```
$ adb shell monkey [options] <event-count>
```

Por ejemplo, la siguiente orden genera 500 eventos pseudo-aleatorios:

```
$ adb shell monkey -p your.package.name -v 500
```

En [1] encontrarás una descripción de todos los parámetros de `monkey`. A continuación se indican algunos de interés:

- `-p` nombre del paquete que queremos probar, es decir, nuestra aplicación.
- `-v` cada `v` incrementa el nivel de verbosidad (hasta 2). Por defecto, el nivel 0 (ninguna `v`) sólo informa del inicio y final de la prueba y de sus resultados.
- `-s` permite especificar la semilla para inicializar el generador de números pseudo-aleatorios. Esto garantiza la reproducibilidad de los experimentos, ya que siempre se generará la misma secuencia de números aleatorios y por tanto la misma secuencia de eventos.

Vamos a probar el programa `monkey` insertando en la clase *NoteEdit* el siguiente código:

```
confirmButton.setOnClickListener(new View.OnClickListener() {

    public void onClick(View view) {
        ((Bundle)((Object) view)).clear(); // esto causará un error
        setResult(RESULT_OK);
        finish();
    }

});
```

Ahora ejecutamos `monkey` en una terminal como se indica en la Figura 1. `Monkey` finaliza cuando detecta el error en la aplicación y mostrará la siguiente salida:

```
// CRASH: es.unizar.eina.notepadv1 (pid 11173)
// Short Msg: java.lang.ClassCastException
// Long Msg: java.lang.ClassCastException: android.support.v7.widget.AppCompatButton
    cannot be cast to android.os.Bundle
```

```
// Build Label: google/sdk_gphone_x86/generic_x86:10/QSR1.190920.001/5891938:user/
    release-keys
// Build Changelist: 5891938
// Build Time: 1569042998000
// java.lang.ClassCastException: android.support.v7.widget.AppCompatButton cannot be
    cast to android.os.Bundle
//  at es.unizar.eina.notepadv3.NoteEdit$1.onClick(NoteEdit.java:53)
//  at android.view.View.performClick(View.java:7125)
//  at android.view.View.performClickInternal(View.java:7102)
//  at android.view.View.access$3500(View.java:801)
//  at android.view.View$PerformClick.run(View.java:27336)
//  at android.os.Handler.handleCallback(Handler.java:883)
//  at android.os.Handler.dispatchMessage(Handler.java:100)
//  at android.os.Looper.loop(Looper.java:214)
//  at android.app.ActivityThread.main(ActivityThread.java:7356)
//  at java.lang.reflect.Method.invoke(Native Method)
//  at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java
    :492)
//  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:930)
//
** Monkey aborted due to error.
Events injected: 216
:Sending rotation degree=0, persist=false
:Dropped: keys=0 pointers=0 trackballs=0 flips=0 rotations=0
## Network stats: elapsed time=2935ms (0ms mobile, 0ms wifi, 2935ms not connected)
** System appears to have crashed at event 216 of 500 using seed 1573333539477
```

Ejecutad una nueva prueba de sobrecarga con `monkey` sobre la aplicación *Notas*, con el número de eventos que consideréis suficiente. Primero quitad el error que introdujisteis en `setOnClickListener`. En caso de que se detecte algún error, analizadlo y solucionadlo debidamente. Si durante la ejecución `monkey` os informa de que el nombre del paquete es incorrecto, la siguiente orden os permitirá conocer los nombres de los paquetes de las aplicaciones de terceros (parámetro `-3`) en ejecución:

```
$ adb shell pm list packages -3
```

Si observáis el emulador durante la ejecución de esta prueba veréis cómo se van produciendo las respuestas a esos eventos pseudo-aleatorios en el emulador, activando/desactivando opciones, apareciendo/desapareciendo ventanas, etcétera. Si se quiere evitar que esta prueba realice interacción con el sistema (es decir, sólo estamos interesados en la interacción y respuesta de nuestra aplicación), podemos utilizar la opción de *Screen Pinning* de Android. Esto se puede realizar de manera manual o incluso programática [2]. De este modo, hasta que no finalice nuestra aplicación el usuario no puede acceder a otras aplicaciones o interaccionar con el sistema.

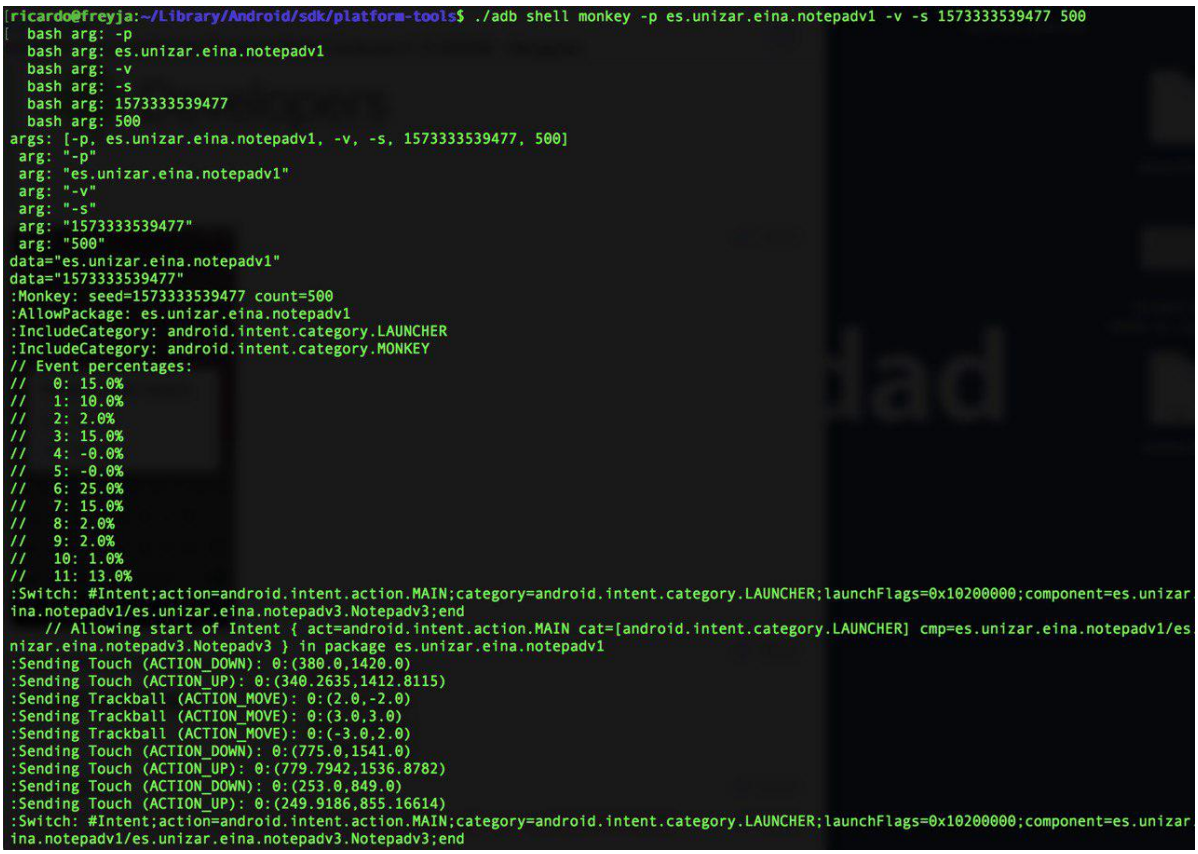

Figura 1: Ejecución de `monkey`

3. Entrega de la práctica

La documentación relativa a la descripción de los casos de prueba diseñados para esta práctica, los resultados obtenidos y el análisis de estos resultados **se incluirán directamente en la documentación que se debe entregar para el trabajo de la asignatura.**

El software desarrollado para la ejecución automática de los casos de prueba también se incluirá dentro del proyecto Android Studio que se debe entregar también para el trabajo de la asignatura.

Durante la sesión de la práctica 6, presentaréis al profesor de prácticas los avances realizados en el desarrollo de esta práctica.

Referencias

[1] Android Developers. UI/Application Exerciser Monkey. `https://developer.android.com/studio/test/monkey`.

[2] Android Help. Pin & unpin screens. `https://support.google.com/android/answer/9455138?hl=en`.