Laureano Ateca González

Informe de pruebas: historia de usuario Listar gasolineras cercanas desde punto conocido - 399313

Estructura del informe de pruebas:

- Pruebas de aceptación: Es el Product Owner quien detectará si se cumplen los casos de prueba para estas pruebas en el Sprint Review.

- Pruebas de interfaz de usuario (UI Test). Ejecución de pruebas y corrección de errores para el escenario Listar gasolineras cercanas desde punto conocido.

- Pruebas unitarias. Ejecución de pruebas y corrección de errores para las clases OrdenarGasolinerasPorDistancia y PresenterGasolineras.

Pruebas de aceptación:

Debe ser el Product Owner quien compruebe en un dispositivo móvil si se cumplen los casos de prueba de las pruebas de aceptación, teniendo instalado el .apk en dicho dispositivo y abriendo la aplicación.

Pruebas de interfaz de usuario:

Son pruebas que verifican el correcto comportamiento de la funcionalidad haciendo una integración completa de sus elementos. Las dependencias entre clases se realizan con objetos reales y no con objetos Mock.

Pruebas unitarias:

Casos de prueba de unitarias:

Clase OrdenarGasolinerasPorDistancia → int compare(Gasolinera o1, Gasolinera o2)

Identificador	Valores de entrada	Valor esperado
UOGPD.1a	Gasolinera gasolinera1, Gasolinera gasolinera2 → distancia a un punto conocido igual entre las dos gasolineras que se comparan.	0
UOGPD.1b	Gasolinera gasolinera1, Gasolinera gasolinera2 → distancia a un punto conocido menor para la gasolinera1 que la de la gasolinera2 al compararlas.	-1
UOGPD.1c	Gasolinera gasolinera1, Gasolinera gasolinera2 → distancia a un punto conocido mayor para la gasolinera1 que la de la gasolinera2 al compararlas.	1
UOGPD.1d	null, Gasolinera gasolinera2 .	NullPointerException
UOGPD.1e	Gasolinera gasolinera1, null .	NullPointerException

Resultado de la ejecución:

Al ejecutar la clase de pruebas OrdenarGasolinerasPorDistanciaTest.java, que contiene la implementación de los casos de prueba a, b, c, d y e, no se ha detectado ningún error. No hay fallos y las pruebas han pasado satisfactoriamente, por lo que no hay que realizar ninguna corrección.

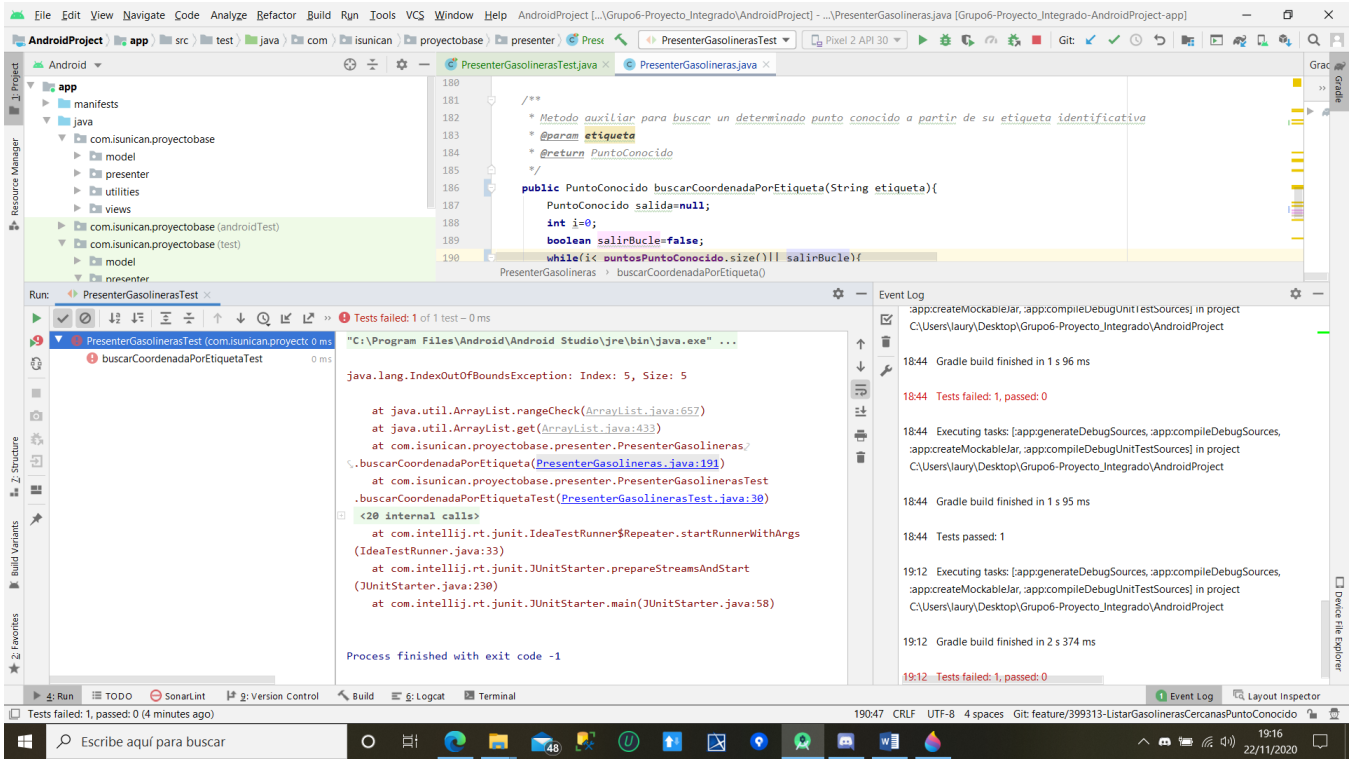
Esto se verifica mediante la ejecución de la clase en Android Studio (la cual se ha apoyado en la librería de JUnit), y también mediante el proceso de integración continua, en el que Travis ejecuta las pruebas e indica si pasan o no, siendo este el primer caso.

Clase PresenterGasolineras → PuntoConocido buscarCoordenadaPorEtiqueta(String etiqueta)

Identificador	Valores de entrada	Valor esperado
UPG.1a	String etiqueta → etiqueta que se encuentra en algún PuntoConocido del listado.	PuntoConocido buscado
UPG.1b	String etiqueta → etiqueta que no se encuentra en ningún PuntoConocido del listado.	null
UPG.1c	null	null

Resultado de la ejecución:

Al ejecutar la clase de pruebas PresenterGasolinerasTest.java, que contiene la implementación de los casos de prueba a, b y c, se han detectado un par de errores. Se observan en la siguiente imagen:



En un primer lugar saltó la excepción IndexOutOfBoundsException, lo que nos indica que hemos intentado acceder a una posición de la lista inexistente, entonces el lugar donde mirar el error es en la condición del bucle while. Se detecta un doble error, en primer lugar, si tenemos dos subcondiciones unidas por un OR (||), si una de las dos es true permanentemente el bucle va a seguir hasta que salte una excepción, como es el caso, o indefinidamente. En este caso la primera subcondición es true hasta recorrer la lista y la segunda es false; por la primera condición se entra en el bucle, pero si se encuentra el elemento de la lista y la variable salirBucle (segunda subcondición) pasa a valer true nunca se sale del bucle, ya que nunca deja de ser true y salta la excepción.

El error salta en el caso de prueba UPG.1a.

La condición debería ser while(i<puntosPuntoConocido.size() && !salirBucle)

Ya que si no se cumple una de las dos subcondiciones como true salimos del bucle, de manera que si recorremos la lista completa salimos del bucle y si encontramos el elemento cambiamos el valor de salirBucle de false a true y con la negación salimos del bucle. Lo que se busca es recorrer la lista hasta encontrar el elemento y no recorrer la lista entera si no es necesario; de esta manera se consigue.

Esto se verifica mediante la ejecución de la clase en Android Studio (la cual se ha apoyado en la librería de JUnit), y también mediante el proceso de integración continua, en el que Travis ejecuta las pruebas e indica si pasan o no, siendo este el primer caso.