Introducción

El proyecto Bets que se realizó en la asignatura de Ingeniería del Software (2º curso), vamos a verificarlo considerando técnicas estáticas (visto en el Lab 1.0) y técnicas dinámicas. En este apartado del proyecto se solicita el diseño de casos de prueba utilizando la técnica de caja blanca y caja negra. Se realizará pruebas unitarias y pruebas de integración, haciendo uso de JUnit y Mock. El proyecto inicial lo puedes encontrar en el siguiente enlace https://github.com/jononekin/Bets2021.

Objetivos

1. Verificación estática y obtención de informe Sonar (Lab 1.0).
2. Diseñar casos pruebas según técnica de caja blanca y caja negra.
3. Implementar los casos de prueba haciendo uso de JUnit y Mocks.

El proyecto está adaptado para que sea compatible con los Mocks y se proporciona algunos ejemplos de prueba en la carpeta docs/castellano/EjemploArquitecturasPruebas.pdf así como su implementación en la carpeta src/test

El proyecto puede realizarse en grupo o individualmente, en cualquier caso, cada alumno deberá verificar un método (distinto del ejemplo presentado) e implementar sus pruebas (de manera individual). Si se trabaja en equipo, se elegirán diferentes métodos a verificar. El proyecto desarrollado debe estar en Github y el informe de análisis en sonarcloud.

Se debe conseguir una cobertura del 100% (sonarcloud.io) del método elegido. El fin último de la verificación es identificar errores, no su corrección.

1. Tareas a realizar (cada alumno individualmente):

a. Corrige los errores estáticos que se propusieron en el primer laboratorio de Sonar y documéntalos en el fichero sonar.pdf.
b. Elije un método (distinto del que está en los ejemplos de prueba) en la clase `DataAccess` cuya implementación tenga una complejidad ciclomática mayor que 4.
c. **Pruebas unitarias**: Realiza el diseño de **caja blanca** (grafo de diagrama de flujo, complejidad ciclomática y tablas de prueba) y caja negra (clases de equivalencia con valores límite) del método escogido. En el estudio diseñarás los casos de prueba necesarios para conseguir la cobertura al 100%. A continuación implementa las pruebas utilizando JUnit en dos clases diferentes. El nombre de la clase de test estará compuesta por el nombre del método más los sufijos "DAWTest" y "DABTest" respectivamente (por ejemplo, si el método es "createQuestion.java" los nombres de las clase de test serán createQuestionDAWTest.java y createQuestionDABTest). Finalmente describe los errores encontrados en un documento (p.ej. createQuestionDA.pdf). En la carpeta /src/test/java/ puedes encontrar pruebas unitarias del método `createQuestion`.
d. **Pruebas de Integración con Junit+Mockito**. Trabajarás con el método de `BLFacadeImplementation` que llama al método anterior (el de `DataAccess`). Realiza un estudio de caja negra (clases de equivalencia y valores límite si procede). Es posible que te sirvan los casos de prueba del estudio realizado anteriormente. Implementa las pruebas diseñadas utilizando un objeto Mock de la clase `DataAccess`. Es decir, no se trabajará directamente con la clase DataAccess, sino con su doble. El nombre de la clase seguirá la misma estructura que en el ejemplo `NombreDelMetodoMockIntTest`, que se interpreta como Test de Integración mediante Mock del método nombreDelMétodo. Finalmente describe los errores encontrados en un documento

(p.ej. createQuestionBLTest.pdf).En la carpeta /src/test/java/ puedes encontrar algunas pruebas del método `createQuestion`

2. Documentación a entregar al final (un documento pdf)

Presenta un documento PDF con la siguiente información:

a. El nombre de cada miembro del equipo o del autor del trabajo, junta con las horas de dedicación individual.
b. Para el método elegido:
   i. Autor de las pruebas y Código (totalmente documentado) de los métodos seleccionados en `DataAccess` y en `BLFacadeImplementation`.
   ii. Pruebas Unitarias:
      - Diseño: Las tablas de las técnicas de caja blanca y caja negra de `DataAccess`
      - Implementación: JUnit de la implementación de los casos
      - Defectos encontrados, caso que lo descubrió y descripción del error (valor esperado y valor obtenido).
   iii. Pruebas de Integración
      - Diseño: Las tablas de las técnicas de caja negra (Clases de Equivalencia, Casos derivados y Valores Límite) de `BLFacadeImplementation` o simplemente indicar si difiere en algo o no de la presentada en el apartado anterior.
      - Implementación: JUnit+Mock de los casos derivados
      - Defectos encontrados, caso que lo descubrió y descripción del error (valor esperado y valor obtenido).
c. Enlaces de Github y sonarcloud.io

3. Valoración. 1,25 puntos (diseño e implementación) + 0,25 análisis estático de código. Misma valoración para todos los miembros del grupo.