P10- Análisis de pruebas: Cobertura

Cobertura

Un análisis de la cobertura de nuestras pruebas nos permite conocer la **extensión** de las mismas. Si un código no está siendo probado, no podemos ser efectivos. En esta sesión utilizaremos la herramienta JaCoCo para analizar tanto la cobertura de nuestros tests.

Esta herramienta se integra con Maven, de forma que podemos incluir el análisis de cobertura en la construcción de nuestro proyecto.

GitHub

El trabajo de esta sesión también debes subirlo a *GitHub*. Todo el trabajo de esta práctica deberá estar en el directorio **P10-Cobertura** dentro de tu espacio de trabajo.

➪ ➪ Ejercicio 1: Proyecto cobertura

Crea un proyecto Maven en la carpeta *P10-Cobertura/ (*de tu directorio de trabajo), con groupId = **ppss**, y artifactID= **cobertura**. El nombre del proyecto IntelliJ será **cobertura**.

Añade la siguiente clase al paquete "***ejercicio1***".

```
package ejercicio1;
public class MultipathExample {
      public int multiPath1(int a, int b, int c) {

         if (a > 5) {
            c += a;
        }
        if (b > 5) {
            c += b;
        }
        return c;
    }
}
```

A) Sin usar JaCoCo: calcula la complejidad ciclomática (CC) para el método *multiPathExample()*. Implementa un número mínimo de casos de prueba para conseguir una **cobertura del 100% de líneas y de condiciones**. para el método ***multiPath1()***.

B) Obtén un **informe de cobertura** para el proyecto (con JaCoCo) y familiarízate con el informe obtenido (el valor "n/a" significa "Not applicable"). Puedes abrir directamente el fichero ***index.html*** del informe en el navegador desde el menú contextual, seleccionando "*Open in Browser*". Fíjate en el valor de cobertura proporcionado por JaCoCo (en clase hemos explicado cómo se calcula).

> **Nota:** Cuando abrimos un fichero html en el navegador desde InteliJ, en ocasiones no se visualiza correctamente. Si eso ocurre, ábrelo directamente desde el Gestor de Archivos.

C) Añade el siguiente **caso de prueba** al conjunto (a=7, b=7, c=7, resultado esperado = 7) y vuelve a generar el informe. Observa que el informe que ves no es correcto. Como habrás podido comprobar, JaCoCo no borra informes anteriores. Para solucionar el problema simplemente debes ejecutar la fase *clean* antes de ejecutar los tests y generar un nuevo informe.

D) Recuerda que siempre tienes que ejecutar *mvn clean*, antes de generar un nuevo informe. Cambia el caso de prueba del apartado anterior por ((a=3, b=6, c=2, resultado esperado = 8). Ahora añade el método ***multiPath2()*** a la clase *MultipathExample* y genera de nuevo el informe. Añade el mínimo

número de casos de prueba necesarios (utilizando un test parametrizado) para conseguir una cobertura del 100% de condiciones y decisiones para este método y vuelve a generar el informe. Observa las diferencias. Debes tener claro cómo obtiene JaCoCo el valor de CC para el nuevo método que has añadido

```
public int multiPath2(int a, int b, int c )
   if ((a > 5) && (b < 5)) {
      b += a;
   }
   if (c > 5) {
      c += b;
   }
   return c;
}
```

E) Añade el método ***multiPath3()*** a la clase *MultipathExample* y genera de nuevo el informe. Añade los casos de prueba necesarios (utilizando un test parametrizado) para conseguir una cobertura del 100% de condiciones y decisiones para dicho método y vuelve a generar el informe. Observa las diferencias. De nuevo debes tener claro cómo obtiene JaCoCo el valor de CC para el nuevo método que has añadido.

```
public int multiPath3(int a, int b, int c )
   if ((a > 5) & (b < 5)) {
      b += a;
   }
   if (c > 5) {
      c += b;
   }
   return c;
}
```

➪ ➪ Ejercicio 2: Informes de cobertura

En el proyecto anterior, vamos a crear un nuevo paquete "***ejercicio2***", al que deberás añadir las clases de la carpeta /plantillas-P10/ejercicio2. Cada fichero debes añadirlo donde corresponda.

A) Obtén los **informes de cobertura** de nuestros **tests unitarios** y de **integración**, teniendo en cuenta las clases que has añadido (tendrás que modificar convenientemente el pom del proyecto).

B) Queremos "**chequear**" que se alcanzan ciertos niveles de cobertura con nuestras **pruebas unitarias**. Modifica el pom para que se compruebe de forma automática las siguientes dos "reglas" (<rules>):

- ➡ A nivel de **proyecto**, queremos conseguir una CC con un valor mínimo del 90%, una cobertura de instrucciones mínima del 80%, y que no haya ninguna clase que no se haya probado.
- ➡ A nivel de **clase**, queremos conseguir una cobertura de líneas del 75%

C) Obtén de nuevo los informes de cobertura, y comprueba que el proceso de construcción falla. porque se incumplen las dos reglas (<rule>) que hemos impuesto. Con respecto al nivel de proyecto, modifica la condición (<limit>) que no permite completar la construcción, cambiando el valor del contador correspondiente. Con respecto a nivel de clase, no queremos cambiar la regla, de forma que para que la construcción se lleve a cabo con éxito tendrás que excluir una de las clases del paquete ejercicio2 anidando en la <rule> correspondiente el elemento <exclude>, como se muestra a continuación:

```
<rule>
   <element>...</element>
   <excludes>
      <exclude>ejercicio2.NombreDeLaClase</exclude>
   </excludes>
   ...
</rule>
```

➪ ➪ Ejercicio 3: Proyecto Matriculacion

Para este ejercicio usaremos el proyecto multimódulo ***matriculacion*** de la práctica P06B.

Para ello copia tu solución, es decir, la carpeta *matriculacion* (y todo su contenido) en el directorio de esta práctica *P10-Cobertura/*.

Una vez copiada la carpeta *matriculacion*, simplemente abre el proyecto ***matriculacion*** desde IntelliJ (seleccionando la carpeta que acabas de copiar).

Se pide:

A) Modifica convenientemente el pom del módulo ***matriculacion-dao*** para obtener un informe de cobertura para dicho módulo (tanto para los tests unitarios como para los tests de integración). Genera el informe a través del correspondiente comando maven (puedes obtener los informes directamente desde este módulo).

B) Observa los valores obtenidos a nivel de proyecto y paquete. Tienes que **tener claro cómo se obtienen** dichos valores. Fíjate también en los valores de CC obtenidos a nivel de paquete y clase.

C) El proyecto *matriculacion-dao* también ejecuta las clases de *matriculacion-comun*, sin embargo no aparecen en el informe (es obvio porque los ejecutables de *matriculacion.comun* no están instrumentados).

D) Dado que tenemos un proyecto multimódulo, vamos a usar la goal *jacoco:report-aggregate* para generar un informe para las dependencias de cada módulo, a partir del mecanismo reactor de maven. (ver https://www.jacoco.org/jacoco/trunk/doc/report-aggregate-mojo.html)

   Para ello tendrás que comentar el plugin jacoco del proyecto *matriculacion-dao* y configurar el pom del proyecto *matriculacion* para obtener los informes unitarios, de integración y además el informe agregado (goal *report-aggregate)*. Si observas la documentación del enlace del apartado anterior, verás que dicha goal no está asociada por defecto a ninguna fase de maven. Debes tener claro qué ocurrirá si no la asociamos a ninguna fase, así como a qué fase deberemos asociarla.

E) Ejecuta el comando maven correspondiente para obtener el informe agregado de cobertura para el proyecto multimódulo (así como los informes de cobertura para los test unitarios y de integración para todos los módulos del proyecto). Averigua dónde se genera el informe agregado de cobertura. Observa las diferencias con el informe que hemos obtenido anteriormente.

Resumen

¿Qué conceptos y cuestiones me deben quedar CLAROS después de hacer la práctica?

NIVELES DE COBERTURA

- La cobertura es una métrica que mide la extensión de nuestras pruebas. Existen diferentes variantes de esta métrica, que se pueden clasificar por niveles, de menos a más cobertura. Es importante entender cada uno de los niveles.
- El cálculo de esta métrica forma parte del análisis de pruebas, que se realiza después de su ejecución..

HERRAMIENTA JaCoCo

- JaCoCo es una herramienta que permite analizar la cobertura de nuestras pruebas, calculando los valores de varios "contadores" (JaCoCo counters), como son: líneas de código, instrucciones, complejidad ciclomática, módulos, clases,... También se pueden calcular los valores a nivel de proyecto,, paquete, clases y métodos.
- JaCoCo puede usarse integrado con maven a través del plugin correspondiente. Es posible realizar una instrumentación de las clases on-the-fly, o de forma off-line. En nuestro caso usaremos la primera de las opciones.
- JaCoCo genera informes de cobertura tanto para los tests unitarios como para los tests de integración. Y en cualquier caso, se pueden establecer diferentes "reglas" para establecer diferentes niveles de cobertura dependiendo de los valores de los contadores, de forma que si no se cumplen las restricciones especificadas, el proceso de construcción no terminará con éxito.
- De igual forma, para proyectos multimódulo, se pueden generar informes "agregados", de forma que el informe agregado de cada módulo "incluye" los informes de cobertura de los módulos de los que depende.

Observaciones sobre esta práctica

EJERCICIO 1:

- Cuando se indica que se quiere conseguir una cobertura del 100% de líneas y de condiciones, hay que considerar LAS DOS a la vez. Cuidado con esto. Siempre que se diga que se quiere una cobertura de nivelX y nivelY, nos estaremos refiriendo a que queremos conseguir ambos niveles a la vez.
- Para justificar el valor de CC, en el apartado A hay que usar las fórmulas que ya conocemos. A partir del apartado B), cuando se pide justificar los valores de cobertura del informe de JaCoCo, lógicamente tendréis que tener en cuenta la fórmula que usa JaCoCo.
- Los tests parametrizados no tienen por qué crearse en una nueva clase. No hay ningún problema en que la clase contenga tests parametrizados y sin parametrizar
- En este primer ejercicio tenéis que familiarizaros con los informes de cobertura de JaCoCo y tenéis que tener claros qué significan los valores de los contadores, teniendo en cuenta que dichos informes se proporcionan en varios niveles.
- También hay que tener claro cómo hay que configurar el pom para poder generar los informes de cobertura de los tests unitarios.

EJERCICIO 2:

- El ejercicio 2 se resuelve en el proyecto "cobertura" que hemos creado en el ejercicio1. En un paquete diferente.
- Tenéis que modificar correctamente el pom, para generar no sólo los informes de cobertura de los tests unitarios, sino también los informes de los tests de integración.
- También tenéis que familiarizaros con el uso de la goal del plugin JaCoCo para "chequear" diferentes niveles de cobertura. El apartado B) requiere configurar DOS "rules", una a nivel de proyecto, y otra a nivel de clase.

EJERCICIO 3:

- El ejercicio 3 usa el proyecto multimódulo matriculación, y debe ser un proyecto IntelliJ totalmente independiente del proyecto "cobertura" de los ejercicios anteriores. Es decir, para esta práctica tendremos dos proyectos IntelliJ: uno con un proyecto maven ("cobertura") y otro con un proyecto maven multimódulo ("matriculacion").
- En este ejercicio obtenemos 3 informes de cobertura diferentes: para los tests unitarios, para los tests de integración y los informes agregados. Hay que tener claro dónde (en qué carpeta) se genera cada uno de ellos. Así como dónde y en qué ficheros se almacena el resultado del análisis de cobertura cuando se ejecutan los correspondientes tests.
- Recuerda que el proyecto matriculación es un proyecto maven multimódulo. Usamos la relación de herencia para evitar en lo posible redundancias en las configuraciones del pom de los diferentes módulos.
- Observa los diferentes informes para tener claro como calcula JaCoCo las diferentes métricas en cada uno de los niveles.