P11- Análisis de pruebas: Cobertura

Cobertura

Un análisis de la cobertura de nuestras pruebas nos permite conocer la **extensión** de las mismas. En esta sesión utilizaremos la herramienta Cobertura para analizar tanto la cobertura de líneas como de condiciones de nuestros test.

Bitbucket

El trabajo de esta sesión también debes subirlo a *Bitbucket*. Todo el trabajo de esta práctica deberá estar en el directorio **P11-Cobertura** dentro de tu espacio de trabajo, es decir, dentro de tu carpeta: ppss-Gx-apellido1-apellido2-2019.

Recuerda que si trabajas desde los ordenadores del laboratorio primero deberás configurar git y clonar tu repositorio de Bitbucket.

Ejercicio 1: proyecto cobertura

Crea un proyecto Maven (en la carpeta *ppss-Gx-apellido1-apellido2-2019/P11-Cobertura/*) un proyecto maven, con groupId = **ppss**, y artifactID= **cobertura**. El nombre del proyecto IntelliJ será **cobertura**.
Añade la siguiente clase al paquete "grupo1".

```
package grupo1;
public class MultipathExample {
      public int multiPath(int a, int b, int c) {

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

A) ¿Cuál es la complejidad ciclomática (CC) del método multiPath()? Implementa un número mínimo de casos de prueba para conseguir una **cobertura del 100% de líneas y de condiciones**. ¿Cuál es ese número? Explica por qué es diferente del valor de CC.

B) Obtén un **informe de cobertura** para dicho proyecto de las **dos formas** que hemos visto en clase de teoría. Indica las **diferencias** entre ambas opciones observando las goals ejecutadas que se muestran en consola de salida de Maven. Lógicamente, debes obtener informes IDÉNTICOS en ambos casos. ¿Qué debes hacer para que, ejecutando la alternativa 1 y a continuación la 2 o al revés, obtengas exactamente el mismo informe?

Nota: Si observas detenidamente la "traza" de la ejecución de maven mostrada en el terminal, fíjate que, a pesar de que en la documentación del plugin se indica claramente que la goal **cobertura:cobertura** invoca a la fase *test* antes de ejecutarse a sí misma (http://www.mojohaus.org/cobertura-maven-plugin/cobertura-mojo.html), en realidad no es eso lo que ocurre.
Además, puedes comprobar que si algún test unitario falla, NO se detiene el proceso de construcción, es decir, se genera igualmente el informe de cobertura.
La ejecución de la fase **site** tiene activada por defecto la generación de informes para los tests unitarios y los de integración. En cualquier caso sólo se genera **un informe**, teniendo en cuenta la ejecución de los dos tipos de tests.

C) Ahora añade la siguiente clase junto con las pruebas para conseguir una cobertura del 100% de líneas y condiciones. Vuelve a generar el informe con mvn site, pero ten en cuenta que no queremos recompilar todo el código de nuevo.

```
package grupo2;
public class HandleStrings {
    public String extractMiddle(String s) {
        int idx1 = s.indexOf(':');
        int idx2 = s.indexOf(':', idx1 + 1);
        return s.substring(idx1 + 1, idx2);
    }
}
```

Observa el informe generado para la clase *HandleStrings* y explica el valor obtenido para la cobertura de condiciones (pulsando con el botón izquierdo del ratón sobre dicho valor te abrirá una ventana de ayuda). Explica qué diferencia hay entre dicho valor y cero.

Fíjate también en que ahora el valor de CC para el proyecto ha cambiado. Justifica el nuevo valor.

D) Realiza las modificaciones necesarias en el pom (tal como hemos explicado en clase) para que Cobertura NO instrumente la clase *HandleStrings*. Vuelve a generar el site (recuerda que debes poner a cero los contadores de cobertura).

➪ ➪ Ejercicio 2: proyecto multimódulo matriculacion

Para este ejercicio usaremos el proyecto multimódulo ***matriculacion*** (concretamente el proyecto de los módulos de proyecto IntelliJ P07B-dbunit).

Para ello copia la carpeta P07B-dbunit/matriculacion (y todo su contenido) en el directorio de esta práctica (*ppss-Gx-apellido1-apellido2-2019/P11-Cobertura/*).

Puedes borrar el fichero *matriculacion.iml* del proyecto, ya que no nos hará falta. A continuación 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 proyecto (utiliza la fase site). Recuerda que para generar el informe debes hacerlo desde el proyecto matriculacion-dao. Antes de generar el informe deberías asegurarte de que no existe el directorio target usando el comando maven correspondiente.

B) Observa los porcentajes obtenidos de cobertura de líneas y condiciones obtenidas a nivel de proyecto y paquete y justifícalos. Explica los valores de CC obtenidos a nivel de paquete y clase.

C) El proyecto matriculacion-dao también ejecuta las clases de matriculacion-comun. ¿por qué no aparecen en el informe?

D) Ejecuta "mvn clean" y genera el informe con usando la goal cobertura:cobertura. ¿Por qué no ves el mismo informe? Utiliza la goal correcta para obtener el mismo informe que en el apartado anterior.

E) Ahora vamos a exigir un mínimo de cobertura en nuestros tests. Comprueba que si exigimos una cobertura de condiciones del 85% y de líneas del 90% (a nivel de paquete) y ejecutamos "*mvn cobertura:check-integration-test*", se interrumpe el proceso de construcción del proyecto. Vamos a invocar directamente la goal *cobertura:check-integration-test*, en lugar de ejecutar la fase *verify*. Para ello sólo tienes que configurar en el pom la correspondiente etiqueta <check> (en el plugin de cobertura, dentro de la etiqueta <build>). NO incluyas la etiqueta <executions> en el plugin (ya que vamos a ejecutar "*mvn verify*"). Tampoco es necesario que incluyas la etiqueta <haltOnFailure> ya que por defecto tiene el valor true.

> **Nota**: Puedes comprobar que si configuramos el plugin tal y como hemos indicado en clase (que además es como se indica en la documentación oficial del plugin), si ejecutamos *mvn verify,* el informe de cobertura obtenido muestra siempre un 0% de cobertura para los tests de integración. Este comportamiento se debe a la implementación de la goal *failsafe:integration-test* que estamos usando, que no permite ejecutar dos veces los tests de integración. Para conseguir el funcionamiento correcto deberíamos usar la version 2.18 del plugin *failsafe*. Pero entonces no podríamos ejecutar los tests JUnit5 :(

F) Ahora vamos a cambiar los porcentajes exigidos. Queremos conseguir un porcentaje de cobertura de condiciones del 25% y de líneas del 40% (a nivel de paquete), y con un porcentaje de cobertura de condiciones y líneas del 20 y 40% respectivamente (a nivel de clase). Cambia la configuración del pom, y ejecuta de nuevo la *goal* anterior. Verás que se detiene la construcción del proyecto. Realiza los cambios necesarios en la configuración del pom para que la construcción tenga éxito cuando ejecutemos "*mvn cobertura:check-integration-test*" con los porcentajes indicados. **Nota**: la propiedad "*haltOnFailure*" tiene que tener el valor "true".