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 tests.

Bitbucket

El trabajo de esta sesión también debes subirlo a Bitbucket. Todo el trabajo de esta práctica, tanto los proyectos maven que vamos a crear, como cualquier otro documento con vuestras notas de trabajo, deberán estar en la carpeta **P11** de vuestro repositorio**.**

Ejercicios

1. **Ejercicio 1**. Crea un módulo proyecto Maven con nombre "practica11", con valores para ArtifactId y GroupId **practica11** y **ppss**, respectivamente. En el campo de texto Package que nos muestra Netbeans pondremos como valor **ejercicio1** y añade la siguiente clase:

```
package ppss.ejercicio1;
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 los casos de prueba mínimos para conseguir una cobertura del 100% de líneas y de condiciones. ¿Cuál es ese número? Explica por qué es diferente del valor de CC.

   Modifica convenientemente el pom (si es necesario), tal y como se explicó en clase de teoría, y obtén un informe de cobertura para dicho proyecto de las dos formas que hemos visto en clase de teoría. (**Nota**: en las transparencias de teoría, el plugin de cobertura debería estar anidado dentro de <reporting><plugins>, en lugar de <reporting>). Indica claramente las diferencias entre ambas opciones observando la consola de salida de Maven (en una de ellas verás que aparece un hiperenlace que puedes abrir directamente desde dicha consola). Lógicamente, debes obtener informes IDÉNTICOS en ambos casos. ¿Qué debes hacer para que esto ocurra (es decir, que ejecutando la alternativa 1 y luego la 2 o viceversa, obtengas exactamente el mismo informe)?

   B) 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 ppss.ejercicio1;
public class HandleStrings
{
    public String extractMiddle(String s)
    {
        int idx1 = s.indexOf(':');
        int idx2 = s.indexOf(':', idx1 + 1);
        return s.substring(idx1 + 1, idx2);
    }

}
```

C) Realiza las modificaciones necesarias en el pom (tal como se explicó en clase de teoría) para que Cobertura no instrumente la clase HandleStrings. Vuelve a generar el site (utiliza el comando adecuado para obtener un resultado correcto). Observa el contenido del directorio target/site/cobertura y explica lo que ha ocurrido.

D) ¿Para instrumentar el código y ver el informe de cobertura necesitamos obtener el "jar" del proyecto? Si tu respuesta es afirmativa explica por qué, y si es negativa indica como podríamos, además de generar el informe de cobertura, obtener dicho jar?

2. **Ejercicio 2**. Para este ejercicio utilizaremos los proyectos matriculacion-comun y matriculacion-dao, que obtuvimos como solución de la práctica P7 (Integración con DbUnit). Modifica convenientemente el pom del módulo matriculacion-dao para obtener un informe de cobertura para dicho proyecto (utiliza la segunda de las alternativas vistas en clase). A la vista del informe, se pide:

A) Observa los porcentajes obtenidos de cobertura de líneas obtenidas en el informe de cobertura y explica por qué algunas clases tienen un 0% de cobertura de líneas y ramas. Explica los valores de CC obtenidos a nivel de proyecto, paquete y clase.

B) Fíjate que algunas clases clases presentan un nivel de cobertura con el valor N/A. N/A son las abreviaturas de “Not Applicable”. Según esto, explica la razón por la que no es aplicable para cada una de las clases en las que figura dicho valor.

C) Ahora vamos a exigir un mínimo de cobertura en nuestros tests. Fíjate que, en este caso, estamos trabajando con tests de integración. La goal que debemos utilizar es "cobertura:check-integration-test", en lugar de "cobertura:check".

Exigimos un mínimo de cobertura

Hemos visto en clase que para configurar el plugin de cobertura para forzar un nivel mínimo de cobertura, utilizamos la etiqueta <executions> y añadimos una ejecución con las goals *cobertura:clean* y *cobertura:check*. Dado que vamos a utilizar la goal *cobertura:check-integration-test*, y que dicha goal "llama" a la fase "verify" antes de ejecutarse a sí misma, no será necesario que incluyamos el elemento <executions> en la configuración del plugin, y lanzaremos la construcción del proyecto directamente invocando la goal correspondiente. Es decir: *mvn cobertura:check-integration-test*

Ahora no estamos interesados en visualizar ningún informe, sino en exigir un nivel mínimo de cobertura en nuestros tests. Comprueba que si exigimos una cobertura de ramas del 85% y de líneas del 90% (a nivel de paquete) no nos deja continuar con el proceso de construcción del proyecto. Indica qué repercusiones tiene el detener la construcción del proyecto.

Comprueba que si configuramos el pom tal y como está indicado en las transparencias de clase (y ejecutamos con las dos formas vistas en clase) no obtenemos un resultado correcto. Explica por qué ocurre esto.

Indica qué cambios tendrías que hacer en la configuración del pom para que no se detuviese la construcción del proyecto exigiendo un porcentaje de cobertura de ramas del 30% y de líneas del 40% (a nivel de paquete), y con un porcentaje de cobertura de ramas y líneas del 28 y 40% respectivamente (a nivel de clase). **Nota**: la propiedad "haltOnFailure" tiene que tener el valor "true".

D) Observa los nuevos informes obtenidos (tendrás que generarlos previamente) y explica por qué en los informes de cobertura no aparece el paquete ppss. matriculacion.to