Cada uno de los módulos, puede construirse por separado, vamos a comprobarlo:

- ejecuta la fase ***compile*** desde el proyecto *matriculacion-comun*: verás que en la vista *Project* aparece el directorio *target* sólo en ese proyecto.
- ahora ejecuta la misma fase desde el proyecto *matriculacion-dao*. Verás que no compila!!! Al estar ejecutando los proyectos por separado, fíjate que *matriculacion-dao* depende de *matriculacion-comun*. Lo deberías tener indicado en el pom en forma de dependencia. Por lo tanto, maven, busca el artefacto de *matriculacion-comun* en nuestro repositorio local. Como no lo encuentra, intentará buscar en la nube, y tampoco lo va a encontrar, y finalmente al compilar mostrar un error al compilar, ya que en el código fuente de *matriculacion-dao* se usan las clases de *matriculacion-comun*.

  Tendrás que ejecutar la fase ***install*** desde *matriculacion* para que se generen todos los artefactos de cada módulo y se guarden en nuestro repositorio local. Comprueba que están ahí.

  Una vez hecho esto, puedes, por ejemplo, ejecutar de nuevo ***clean*** sobre el proyecto *matriculacion*. Y acontinuación vuelve a ejecutar la fase ***compile*** desde el proyecto *matriculacion-dao*. Ahora debe compilar sin problema.

Observa la salida por pantalla de maven y entiende bien por qué se realizan las acciones de construcción en ese orden. Recuerda que debes tener claro qué ficheros y/o artefactos se generan en cada una de las fases ejecutadas.

Necesitarás este proyecto multimódulo para la siguientes sesión de prácticas en la que añadiremos y ejecutaremos tests unitarios y de integración.

Resumen

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

PROYECTOS INTELLIJ

- Un proyecto IntelliJ puede contener diferentes tipos de proyectos: proyectos maven, java, android, servicios rest, servicios web..., cada uno de ellos se denomina MÓDULO.
- Un proyecto IntelliJ no se construye ya que no contiene código, el proceso de construcción se realiza para cada uno de sus módulos.

PROYECTOS MAVEN MULTIMÓDULO

- Un proyecto MAVEN puede ser "single" (single-maven proyect) o puede contener, a su vez, varios proyectos maven. (multimodule maven proyect),cada uno de los cuales se denomina MODULO.
- El proyecto multimódulo tiene un empaquetado pom, en el que se pueden AGREGAR tantos módulos (proyectos maven) como se necesite. El mecanismo de agregación permite ejecutar un comando maven (una vez), y se ejecutará automáticamente en todos los módulos agregados, en el orden determinado por las dependencias entre ellos, e identificado por el mecanismo REACTOR de maven.
- Se pueden establecer relaciones de HERENCIA entre los módulos de un proyecto multimódulo, de esta forma pueden compartir propiedades, plugins, y dependencias.
- En un proyecto multimódulo se pueden usar ambas relaciones (herencia y agregación) o sólo una de ellas.
- Los módulos de un proyecto multimódulo pueden construirse de forma individual: un módulo es un proyecto maven con un empaquetado diferente de pom: puede ser jar, war, ear..

Resumen

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

PRUEBAS DE INTEGRACIÓN

- Nuestro SUT estará formado por un conjunto de unidades. El objetivo principal es detectar defectos en las INTERFACES de dicho conjunto de unidades. Recuerda que dichas unidades ya habrán sido probadas individualmente
- Son pruebas dinámicas (requieren la ejecución de código), y también son pruebas de verificación (buscamos encontrar defectos en el código). Se realizan de forma INCREMENTAL siguiendo una ESTRATEGIA de integración, la cual determinará EL ORDEN en el que se deben seleccionar las unidades a integrar
- Las pruebas de integración se realizan de forma incremental, añadiendo cada vez un determinado número de unidades al conjunto, hasta que al final tengamos probadas TODAS las unidades integradas. En cada una de las "fases" tendremos que REPETIR TODAS las pruebas anteriores. A este proceso de "repetición" se le denomina PRUEBAS DE REGRESIÓN. Las pruebas de regresión provocan que las últimas unidades que se añaden al conjunto sean las MENOS probadas. Este hecho debemos tenerlo en cuenta a la hora de tomar una decisión por una estrategia de integración u otra, además del tipo de proyecto que estemos desarrollando (interfaz compleja o no, lógica de negocio compleja o no, tamaño del proyecto, riesgos,...)
- Recuerda que no podemos integrar dos unidades si no hay ninguna relación entre ellas, ya que el objetivo es encontrar errores en las interfaces de las unidades (en la interconexión entre ellas). Por eso es indispensable tener en cuenta el diseño de nuestra aplicación a la hora de determinar la estrategia de integración (qué componentes van a integrarse cada vez.

DISEÑO DE PRUEBAS DE INTEGRACIÓN

- Se usan técnicas de caja negra. Básicamente se seleccionan los comportamientos a probar (en cada una de las fases de integración) siguiendo unas guías generales en función del tipo de interfaces que usemos en nuestra aplicación.
- Recuerda que si nuestro test de integración da como resultado "failure" buscaremos la causa del error en las interfaces, aunque es posible que en este nivel de pruebas, salgan a la luz errores no detectados durante las pruebas unitarias (ya que NO podemos hacer pruebas exhaustivas, por lo tanto, nunca podremos demostrar que el código probado no tiene más defectos de los que hemos sido capaces de encontrar).

AUTOMATIZACIÓN DE LAS PRUEBAS DE INTEGRACIÓN CON UNA BD

- Usaremos la librería dbunit para automatizar las pruebas de integración con una BD. Esta librería es necesaria para controlar el estado de la BD antes de las pruebas, y comprobar el estado resultante después de ellas..
- Los tests de integración deben ejecutarse después de realizar todas la pruebas unitarias. Necesitamos disponer de todos los .class de nuestras unidades (antes de decidir un ORDEN de integración), por lo que los tests requerirán del empaquetado en un .jar de todos los .class de nuestro código.
- Los tests de integración son ejecutados por el plugin failsafe. Debemos añadirlo al pom, asociando la goal integration-test a la fase con el mismo nombre.
- El plugin sql permite ejecutar scripts sql. Es interesante para incluir acciones sobre la BD durante el proceso de construcción del proyecto. Por ejemplo, "recrear" las tablas de nuestra BD en cada ejecución, e inicializar dichas tablas con ciertos datos. La goal "execute" es la que se encarga de ejecutar el script, que situaremos físicamente en la carpeta "resources" de nuestro proyecto maven. Necesitaremos configurar el driver con los dato de acceso a la BD, y también asociar la goal a alguna fase previa (por ejemplo pre-integration-test) a la fase en la que se ejecutan los tests de integración.
- Es habitual que el código a integrar esté distribuido en varios proyectos. Los proyectos maven multimódulo nos permiten agrupar y trabajar con múltiples proyectos maven. En este caso nuestro proyecto maven multimódulo contendrá varios subproyectos maven.
- El proyecto maven multimódulo (que agrupa al resto de subproyectos), únicamente contiene el fichero de configuración pom.xml, es decir, NO tiene código (carpeta src), simplemente sirve para agrupar todo el código de nuestra aplicación en una única carpeta. El pom del proyecto multimódulo permite reducir duplicaciones, ya que se puede aplicar el mismo comando a todos los submódulos (misma configuración a todos los submódulos (relación de agregación, y/o aplicar la misma configuración a los submódulos (relación de herencia).
- Si usamos un proyecto maven multimódulo,, el mecanismo reactor ejecuta el comando maven en todos los submódulos ordenándolos previamente atendiendo a las dependencias entre ellos. Por lo tanto, si ejecutamos las pruebas de integración desde el proyecto multimódulo, estaremos integrando en orden ascendente

Ejecuta el driver antes de crear el usuario ppss y repite el test para comprobar que funciona en ambos casos.

Resumen

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

PRUEBAS DEL SISTEMA Y ACEPTACIÓN

- Las pruebas del sistema son pruebas de VERIFICACIÓN, mientras que las pruebas de aceptación son pruebas de VALIDACIÓN. En ambos casos nuestro SUT estará formado por todo el código de nuestra aplicación.
- Las pruebas del sistema se diseñan desde el punto de vista del desarollador mientras que las pruebas de aceptación se diseñan desde el punto de vista del usuario final. En ambos casos se deben tener en cuenta varios entradas y resultados esperados "intermedios".
- Tanto las pruebas del sistema como las de aceptación se basan en las propiedades emergentes.. Dadas dichas propiedades emergentes, el objetivo de las pruebas del sistema es encontrar defectos en las funcionalidades de dicho sistema, mientras que el objetivo de las pruebas de aceptación es comprobar que se satisfacen los criterios de aceptación.

DISEÑO DE PRUEBAS DE ACEPTACIÓN

- En ambos casos se usan técnicas de caja negra.
- En una prueba del sistema la selección de comportamientos a probar se hace desde un punto de vista técnico (se tienen en cuenta cómo se ha implementado los componentes implicados en las funcionalidades probadas).
- En una prueba de aceptación el diseño se realiza pensando siempre en el uso "real" de nuestra aplicación por el usuario o usuarios finales. (no tenemos en cuenta qué componentes están implicados).

AUTOMATIZACIÓN DE LAS PRUEBAS DE ACEPTACIÓN CON SELENIUM IDE

- Seleccionamos diferentes escenarios de uso de nuestra aplicación. Selenium IDE NO es un lenguaje de programación, es una herramienta que nos permite generar scripts (de forma automática o manual), que podemos ejecutar desde un navegador, por lo tanto podremos usarla si nuestra aplicación tiene una interfaz web.
- Aunque Selenium IDE no es un lenguaje de programación, en las últimas versiones se han incorporado comandos para controlar el flujo de ejecución de nuestros scripts (incluyendo por ejemplo acciones condicionales y bucles).
- No es posible integrar Selenium IDE en una herramienta de construcción de proyectos, lo que constituye una limitación frente a otras aproximaciones.
- En los ejercicios propuestos se han trabajado una serie de comandos, que es necesario conocer y saber aplicar.
- Recuerda que esta práctica, aunque es muy guiada y se os proporciona la solución, requiere un trabajo personal para poder asimilar todos los conceptos trabajados en clase.

Resumen

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

AUTOMATIZACIÓN DE PRUEBAS DE ACEPTACIÓN

- Consideraremos el caso de que nuestro SUT sea una aplicación web, por lo que usaremos webdriver para implementar y ejecutar los casos de prueba. La aplicación estará desplegada en un servidor web o un servidor de aplicaciones, y nuestros tests accederán a nuestro SUT a través de webdriver, que será nuestro intermediario con el navegador (en este caso usaremos Chrome, junt con el driver correspondiente)
- Si usamos webdriver directamente en nuestros tests, éstos dependerán del código html de las páginas web de la aplicación a probar, y por lo tanto serán muy "sensibles" a cualquier cambio en el código html. Una forma de independizarlos de la interfaz web es usar el patrón PAGE OBJECT„ de forma que nuestros tests NO contendrán código webdriver, independizándolos del código html. El código webdriver estará en las page objects que son las clases que dependen directamente del código html, a su vez nuestros tests dependerán de las page objects.
- Junto con el patrón Page Object, podemos usar la clase PageFactory para crear e inicializar los atributos de una Page Object. Los valores de atributos se inyectan en el test mediante la anotación @Findby, y a través del localizador correspondiente. Esta inyección se realiza de forma "lazy", es decir, los valores se inyectan justo antes de ser usados.
- Con webdriver podemos manejar las alertas generadas por la aplicación a probar, introducir esperas (implícitas y explícitas), realizar scroll en la pantalla del navegador, agrupar elementos, capturar la pantalla del navegador y manejar cookies, entre otras cosas.
- El manejo de las cookies del navegador nos será útil para acortar la duración de los tests, ya que podremos evitar loguearnos en la aplicación para probar determinados escenarios.
- También podremos acortar los tiempos de ejecución de nuestros tests si los ejecutamos en modo headless. Podemos decidir si vamos a ejecutar o no nuestros tests en modo headless aprovechando la capacidad del plugin surefire y/o failsafe, de hacer llegar a nuestros tests ciertas propiedades definidas por el usuario. De esta forma podremos, cuando lancemos el proceso de construcción, ejecutar en ambos modos sin modificar el código.

Resumen

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

DISEÑO DE PRUEBAS DE ACEPTACIÓN DE PROPIEDADES EMERGENTES NO FUNCIONALES

- Los comportamientos a probar se seleccionan teniendo en cuenta la especificación (caja negra).
- En general, las propiedades emergentes no funcionales contribuyen a determinar el rendimiento (performance) de nuestra aplicación, en cuyo caso tendremos que tener en cuenta el "perfil operacional" de la misma, que refleja la frecuencia con la que un usuario usa normalmente los servicios del sistema.
- Es muy importante que las propiedades emergentes puedan cuantificarse, por lo que deberemos usar las métricas adecuadas que nos permitan validar dichas propiedades

AUTOMATIZACIÓN DE PRUEBAS DE ACEPTACIÓN

- Usaremos JMeter, una aplicación de escritorio que nos permitirá implementar nuestros drivers sin usar código java..
- Las "sentencias" de nuestros drivers NO son líneas de código escritas de forma secuencial, sino que usaremos una estructura jerárquica (árbol) en donde los nodos representan elementos de diferentes tipos (grupos de hilos, listeners, samplers, controllers...), que podremos configurar.. El orden de ejecución de dichas "sentencias" dependerá de dónde estén situados en la jerarquía los diferentes tipos de elementos
- Los "resultados" de la ejecución de nuestros test JMeter, consisten en una serie de datos calculados a partir de ciertas métricas (número de muestras tiempos de ejecución,...), que necesariamente estarán contenidas en los listeners que hayamos usado para la implementación de cada driver.
- Los resultado obtenidos por la herramienta JMeter no son suficientes para determinar la validez o no de nuestras pruebas. Será necesario un análisis posterior, que dependerá de la propiedad emergente que queramos validar, para poder cuantificarla.

➪ ➪ 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 (*ppss-2021-Gx-apellido1-apellido2/P10-Cobertura/*).

Puedes borrar el fichero *matriculacion.iml* del proyecto (carpeta "matriculacion"), 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. Genera el informe a través del correspondiente comando maven.

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. Deberías tener claro el por qué no aparecen.

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, aprovechándonos 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*, e incluirlo en el pom del proyecto *matriculacion*. En lugar de usar las goals *report* y *report-integration*, debes usar la goal report-aggregate. Si observas la documentación del enlace, 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. Averigua dónde se genera dicho informe 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 se tengan en cuenta todos y cada uno de los módulos del proyecto y sus dependencias entre ellos.