Java Intermedio 2019 - Módulo IV

November 27, 2019

1 Java Intermedio - Módulo IV

1.0.1 Programa (modificado)

Introducción a Maven, Hibernate y Java Persistence API (JPA). Introducción al uso de test unitarios. Junit.

1.1 Java Persistence API (JPA)

JPA es la propuesta estándar que ofrece Java para implementar un Framework Object Relational Mapping (ORM), que permite interactuar con la base de datos por medio de objetos, de esta forma, JPA es el encargado de convertir los objetos Java en instrucciones para el Manejador de Base de Datos (MDB).

Cuando empezamos a trabajar con bases de datos en Java lo primero que nos enseñan es a utilizar el API de JDBC el cual nos permite realizar consultas directas a la base de datos a través de consultas SQL nativas. JDBC por mucho tiempo fue la única forma de interactuar con las bases de datos, pero representaba un gran problema ya que Java es un lenguaje orientado a objetos y se tenía que convertir los atributos de las clases en una consulta SQL como SELECT, INSERT, UPDATE, DELETE, etc. lo que ocasionaba un gran esfuerzo de trabajo y provocaba muchos errores en tiempo de ejecución.

Una de las cosas más importantes para comprender qué es JPA es entender que JPA es una especificación y no un Framework como tal, ¿pero qué quiere decir esto exactamente?, pues bien, una especificación no es más que un documento en el cual se plasman las reglas que debe de cumplir cualquier proveedor que desee desarrollar una implementación de JPA, de tal forma que cualquier persona puede tomar la especificación y desarrollar su propia implementación de JPA, ¿Esto quiere decir que pueden existir muchas implementaciones de JPA? la respuesta es sí, de echo en la actualidad existen varios proveedores como lo son los siguientes:

- Hibernate
- ObjectDB
- TopLink
- EclipseLink
- OpenJPA

1.2 Hibernate

Hibernate es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

En otras palabras, Hibernate es un Framework que agiliza la relación entre la aplicación y la base de datos.

1.2.1 ¿Por qué usar un framework ORM?

Cuando desarrollamos aplicaciones en muchos casos todo termina siendo un conjunto de ABM (alta, baja y modificaciones de datos) que luego consultamos. Para ello se utiliza una base de datos donde hay muchas tareas repetidas: por cada objeto que quiero persistir debo crear una clase que me permita insertarlo, eliminarlo, modificarlo y consultarlo. Con excepción de consultas especiales, el resto es siempre lo mismo. Este es el momento dónde un ORM tiene una importancia fundamental. Con solo configurarlo todas estas tareas se ejecutan automáticamente y solo hay que preocuparse por las consultas especiales.

1.2.2 ¿Cómo funciona?

El desarollador deberá configurar en un archivo XML o mediante annotations donde corresponde un atributo de una clase, con una columna de una tabla. Es una tarea simple donde existen herramientas que lo hacen por nosotros

1.2.3 Características

- Simplicidad y flexibilidad: necesita un único fichero de configuración en tiempo de ejecución y un documento de mapeo para cada aplicación. Este fichero puede ser el estándar de Java (extensión properties) o un fichero XML. También se tiene la alternativa de realizar la configuración de forma programática. El uso de frameworks de persistencia, tales como EJBs hace que la aplicación dependa del framework. Hibernate no crea esa dependencia adicional. Los objetos persistentes en la aplicación no tienen que heredar de una clase de Hibernate u obedecer a una semántica específica. Tampoco necesita un contenedor para funcionar.
- Completo: ofrece todas las características de orientación a objetos, incluyendo la herencia, tipos de usuario y las colecciones. Además, también proporciona una capa de abstracción SQL llamada HQL. Las sentencias HQL son compiladas por el framework de Hibernate y cacheadas para su posible reutilización.
- Prestaciones: uno de las grandes confusiones que aparecen al utilizar este tipo de frameworks es creer que las prestaciones de la aplicación se ven muy mermadas. Este no es el caso de Hibernate. La clave en este tipo de situaciones es si se realizan el número mínimo de consultas a la base de datos. Muchos frameworks de persistencia actualizan los datos de los objetos incluso cuando no ha cambiado su estado. Hibernate sólo lo hace si el estado de los objetos ha cambiado. El cacheado de objetos juega un papel importante en la mejora de las prestaciones

de la aplicación. Hibernate acepta distintos productos de cacheado, tanto de código libre como comercial.

Ejemplo

1.2.4 Ejericio

Desarrollar el ejemplo visto en clase. Investigar acerca de los métodos de la Api y sus características. Implementar capas faltantes como por ejemplo la DAO.

1.3 Introducción al test unitario: Junit 4

Las pruebas unitarias o Unit testing, forman parte de los diferentes procedimientos que se pueden llevar a cabo dentro de la metodología ágil. Son principalmente trozos de código diseñados para comprobar que el código principal está funcionando como esperábamos. Pequeños test creados específicamente para cubrir todos los requisitos del código y verificar sus resultados.

El proceso que se lleva a cabo, consta de tres partes. El Arrange, donde se definen los requisitos que debe cumplir el código principal. El Act, el proceso de creación, donde vamos acumulando los resultados que analizaremos. Y el Asert, que se considera el momento en que comprobamos si los resultados agrupados son correctos o incorrectos. Dependiendo del resultado, se valida y continúa, o se repara, de forma que el error desaparezca.

1.3.1 Características de las pruebas unitarias

- Automatizable; Aunque los resultados deben ser específicos de cada test unitario desarrollado, los resultados se pueden automatizar, de forma que podemos hacer las pruebas de forma individual o en grupos.
- Completas; El proceso consta de pequeños test sobre parte del codigo, pero al final, se debe comprobar su totalidad.
- Repetibles; En el caso de repetir las pruebas de forma individual o grupal, el resultado debe ser siempre el mismo dando igual el orden en que se realicen los test, los tests se almacenan para poder realizar estas repeticiones o poder usarlos en otras ocasiones.
- Independientes; Es un código aislado que se ha creado con la misión de comprobar otro código muy concreto, no interfiere en el trabajo de otros desarrolladores.
- Rápidos de crear; a pesar de lo que muchos desarrolladores opinen, el código de los tests unitarios no debe llevar más de 5 minutos en ser creado, están diseñados para hacer que el trabajo sea más rápido..

1.3.2 Ventajas de pruebas unitarias

1. Proporciona un trabajo ágil; Como procedimiento ágil que es, te permite poder detectar los errores a tiempo, de forma que puedas reescribir el código o corregir errores sin necesidad de tener que volver al principio y rehacer el trabajo. Puesto que las pequeñas se van haciendo periódicamente y en pequeños packs. Disminuyendo el tiempo y el coste.

2. Calidad del código; Al realizar pruebas continuamente y detectar los errores, cuando el código está terminado, es un código limpio y de calidad.
3. Detectar errores rápido; A diferencias de otros procesos, los tests unitarios nos permiten detectar los errores rápidamente, analizamos el código por partes, haciendo pequeñas pruebas y de manera periódica, además, las pruebas se pueden realizar las veces que hagan falta hasta obtener el resultado óptimo.
4. Facilita los cambios y favorece la integración; Los tests unitarios, nos permiten modificar partes del código sin afectar al conjunto, simplemente para poder solucionar bugs que nos encontramos por el camino. Los tests unitarios, al estar desglosados en bloques individuales permiten la integración de nuevas aportaciones para hacer un código más complejo o actualizarlo en función de lo que el cliente demande.
5. Proporciona información; Gracias al continuo flujo de información y la superación de errores, se puede recopilar gran cantidad de información para evitar bugs venideros.
6. Proceso debugging; Los Tests unitarios ayudan en el proceso de debugging. Cuando se encuentra un error o bug en el código, solo es necesario desglosar el trozo de código testeado. Esta es uno de los motivos principales por los que los tests unitarios se hacen en pequeños trozos de código, simplifica mucho la tarea de resolver problemas.
7. El diseño; Si primero se crean los tests, es mucho más fácil saber con anterioridad cómo debemos enfocar el diseño y ver qué necesidades debemos cumplir. Testeando una pieza del código, también puedes saber que requisitos debe cumplir, y por eso mismo te será mucho más fácil llegar a una cohesión entre el código y el diseño.
8. Reduce el coste; Partiendo de la base que los errores se detectan a tiempo, lo cual implica tener que escribir menos código, poder diseñar a la vez que se crea y optimizar los tiempos de entrega, vemos una clara relación con una reducción económica.

Es necesario saber que las pruebas unitarias por sí solas, no son perfectas, puesto que comprueban el código en pequeños grupos, pero no la integración total del mismo. Para ver si hay errores de integración es necesario realizar otro tipo de pruebas de software conjuntas y de esta manera comprobar la efectividad total del código.

Ejemplos

Test simple: assertEquals, assertNotNull y assertNull, assertNotSame and assertSame, assertTrue and assertFalse, assertThat

```
<li> Test simple: mensajes personalizados </li>
<li> Test de excepciones </li>
<li> Cómo ignorar un test</li>
<li> Before y After </li>
```

2 Gracias a todos por asistir y participar!!!

3 liferreira@vates.com

4 liferreira90@gmail.com

4.1 Equipo de Intel, 8vo piso.