Curso: Sistemas de Informação	
Disciplina: Mapeamento Objeto Relacional	**Ciclo: 8**
Professor: Luiz Gustavo Dias	**Tipo:** Material Complementar
Objetivo: Estudo do JPA	

JPA – JAVA PERSISTENCE API

JPA (ou Java Persistence API) é uma especificação oficial que descreve como deve ser o comportamento dos frameworks de persistência Java que desejarem implementá-la.

A Persistência de dados é um meio para que um aplicativo persista e recupere informações de um sistema de armazenamento não volátil. A Java™ Persistence API (JPA) fornece um mecanismo para gerenciar persistência e mapeamento e funções relacionais de objeto desde as especificações EJB 3.0.

Ser uma especificação significa que a JPA não possui código que possa ser executado. Por analogia, você pode pensar na especificação JPA como uma interface que possui algumas assinaturas, mas que precisa que alguém a implemente.

Apesar de não ter nada executável, a especificação possui algumas classes, interfaces e anotações que ajudam o desenvolvedor a abstrair o código. São artefatos do pacote javax.persistence que ajudam a manter o código independente das implementações da especificação.

Tudo isso retira a necessite de realizar a importação de códigos de terceiros para dentro do código.

Implementação é quem dá vida para a especificação. É o código que podemos executar que chamamos de framework.

Para persistir dados com JPA é preciso escolher uma implementação que é quem, de fato, vai fazer todo o trabalho.

Antes do JPA

Era bem trabalhoso.

A ausência de um framework específico para auxiliar na comunicação com o banco de dados aumentava consideravelmente o trabalho a ser realizado, sendo necessário a inclusão de muitas linhas SQL e, também, do JDBC.

JDBC significa Java™ EE Database Connectivity. No desenvolvimento Java EE, esta é uma tecnologia bem conhecida e comumente utilizada para a implementação da interação do banco de dados.

Com isso em mente, alguns bons anos atrás, desenvolvedores começaram a criar algo que agilizasse esse trabalho. Nascia então o Hibernate. Um framework objeto relacional que simplificava a interação entre a aplicação e o banco de dados.

O Hibernate é um framework objeto relacional porque ajuda a representar tabelas de um banco de dados relacional através de classes.

A vantagem dessa estratégia é a de automatizar as tarefas com banco de dados de forma que é possível simplificar o código da aplicação.

Ele consegue gerar, em tempo de execução, o SQL necessário para interagir com o banco de dados. Para cada banco de dados (MySQL, PostgreSQL, Oracle, etc), ele tem um dialeto diferente que pode ser configurado de acordo com a necessidade da aplicação.

Isso significa também que podemos trocar o banco de dados utilizado sem ter que alterar o código-fonte. Podemos fazer somente com configuração.

Ganhamos em produtividade!

A ideia de mapeamento objeto relacional do Hibernate deu tão certo que ele serviu como a principal inspiração para criação de uma especificação Java para persistência. Nascia a especificação JPA. Também conhecida como Java Persistence API.

Implementação da JPA

A implementação é algo que pode ser executado em nossa aplicação. Qualquer pessoa ou equipe pode escrever sua própria implementação da especificação JPA. Dentre as mais famosas temos o OpenJPA da Apache, o Hibernate da Red Hat e o EclipseLink da Eclipse Foundation.

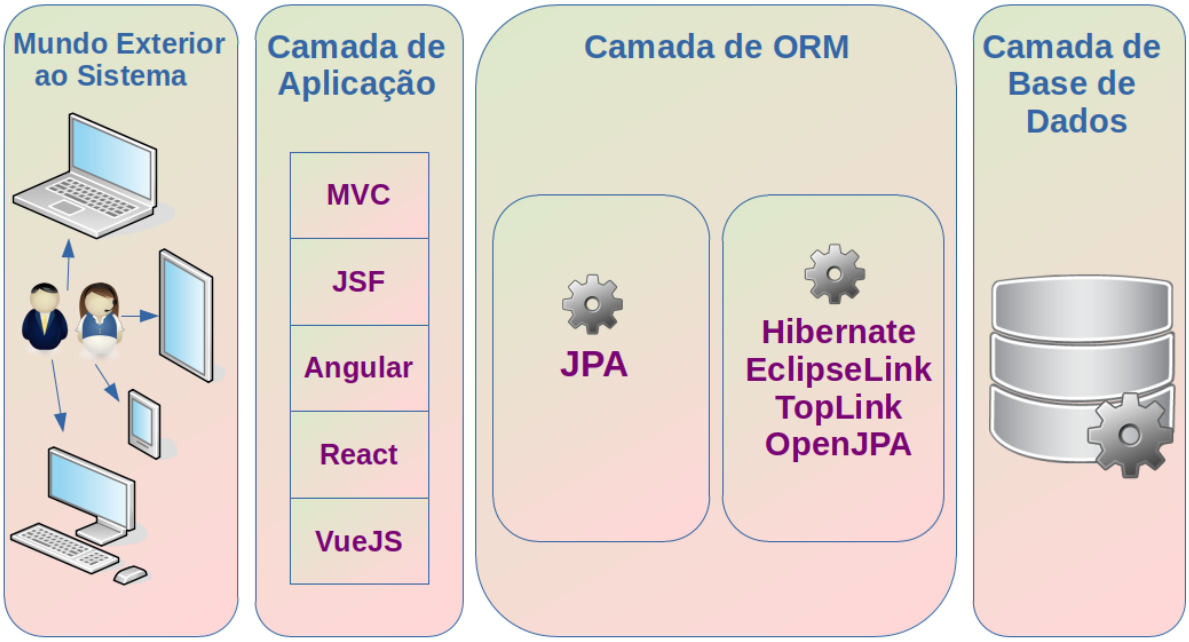
As duas consideradas mais importantes são o Hibernate, por ainda ser a mais utilizada, e o EclipseLink, por ser a implementação de referência.

Ser a referência quer dizer que ela será a primeira a validar as mudanças e novas ideias que surgirem na especificação. É um papel importante para mostrar para a comunidade que a especificação é possível e viável.

A grande ideia da especificação JPA é que a aplicação possa trocar de implementação sem que precise de mudanças no código. Apenas um pouco de configuração.

Uma mesma aplicação que poderia estar rodando no GlassFish com EclipseLink poderia também funcionar no WildFly com Hibernate.

O diagrama apresentado na figura abaixo representa o funcionamento do JPA:



Nesse diagrama podemos ver o mundo exterior ao sistema onde os usuários podem acessar a aplicação de qualquer dispositivo, seja um computador, um tablet, um smartphone ou um notebook, eles vão através da aplicação chamar a camada ORM, onde e localiza a interface JPA e o framework utilizado seja o Hibernate, o EclipseLink,

o TopLink, o OpenJPA, etc. E nesse diagrama podemos ver detalhadamente a divisão entre a interface JPA e o framework ORM utilizado.

Mapeamento Objeto Relacional

Mapeamento Objeto Relacional é a representação de uma tabela de um banco de dados relacional através de classes Java. É também conhecido como ORM ou Object Relational Mapping.

Existem dois mundos: o relacional e o orientado a objetos.

No mundo relacional prevalecem princípios matemáticos com a finalidade de armazenar e gerenciar corretamente os dados, de forma segura e se trabalha com a linguagem SQL que é utilizada para dizer o banco de dados “O QUE?” fazer e não como fazer.

Já no mundo orientado a objetos trabalhamos com classes e métodos, ou seja, trabalhamos fundamentados na engenharia de software e seus princípios que nos dizem “COMO” fazer. O ORM é justamente, a ponte entre estes dois mundos, ou seja, é ele quem vai permitir que você armazene os seus objetos no banco de dados.

Repare como isso começa a ser possível:

Banco de dados	Linguagem Orientada a Objetos
Tabela	Classe
Coluna	Atributo
Registro	Objeto

Enquanto que no banco de dados temos tabelas, colunas e registros, em uma linguagem orientada a objetos, como o Java, temos o equivalente com classes, atributos e objetos.

Essa equivalência já é boa parte do caminho, mas não o suficiente para automatizar todo o trabalho.

Para completar, temos as anotações que adicionarão metadados às classes e permitirão os frameworks ORM, como Hibernate ou EclipseLink, entrarem em ação.

Algumas muito usadas são:

@Entity (indica que a classe também é uma entidade);

@Table (indica a tabela correspondente);

@Id (indica o atributo correspondente à chave primária);

@Column (indica detalhes da coluna que um campo ou propriedade será mapeado).

Por Que Utilizar JPA?

Escrever SQL para consultas, inserções e atualizações, por mais que não seja complexo, é uma tarefa repetitiva e chata.

A primeira vantagem percebida é que já ganhamos isso pronto. O SQL para as operações é gerado em tempo de execução.

Uma segunda coisa que logo reparamos é na simplificação do código-fonte da aplicação. Como muita coisa é automatizada, bem menos código é escrito para as mesmas funções.

Em aplicações desenvolvidas sem ORM, trocar de banco de dados pode ser bem complexo, pois exigiria verificação e correção de todo o SQL da aplicação.

Com JPA, se houver a necessidade de alterar o sistema de banco de dados, basta trocar o dialeto que um novo e compatível SQL é gerado em tempo de execução.

Interessante notar também que o uso do JPA permite que a implementação da especificação seja alterada sem impactos no fonte da aplicação.

Isso tudo permite ter uma manutenabilidade excelente em nossa aplicação, facilitando o dia a dia do desenvolvedor.