

---

<b>Arquiteturas e Frameworks Java para MOR .....</b>	<b>2</b>
Objetivos .....	3
Compreendendo os Termos .....	4
O que é um Java Bean? .....	4
O que são Beans de Entidades? .....	6
O que é JPA – Java Persistence API .....	7
Regras de Beans de Entidade para JPA.....	7
Considerações Quanto aos Atributos de Entidades .....	8
O Que São POJOs? .....	9
Mapeamento Objeto Relacional em Java .....	10
JPA.....	11
OpenJPA .....	11
Hibernate .....	11
TopLink .....	12
EclipseLink.....	12
O Gerenciamento de Persistência de Frameworks Java para MOR .....	13
Contexto .....	13
Gerenciador.....	15
Ciclo de Vida .....	16
Compreendendo a Arquitetura .....	18
O Que é DAO e DTO?.....	19
Ambiente Java EE e SE .....	20
Instalação e Configuração da Arquitetura .....	20

# Arquiteturas e Frameworks Java para MOR

# Objetivos

- Introduzir os conceitos de desenvolvimento de aplicações em N camadas;
- Introduzir as soluções para MOR em Java;
- Exemplificar o desenvolvimento de aplicações N camadas com MOR;
- Entender o contexto de gerenciamento de objetos de uma aplicação tratada por um framework de MOR.

# Compreendendo os Termos

## O que é um Java Bean?

Um Java Bean é uma classe que armazena dados. Tipicamente, em uma aplicação Java, um Java Bean é a classe que irá transportar dados da aplicação correspondentes a um objeto da camada de persistência para a camada de apresentação.

Um Java Bean obedece aos seguintes padrões:

- É uma classe não abstrata e pública;
- Tem um construtor sem parâmetros;
- Para cada propriedade da classe, existe um método acessor (*getter*) e um transformador (*setter*) correspondente;
- Implementa a interface `java.io.Serializable`.

Exemplo:

```
public class Animal implements Serializable { //serelizado

    private static final long serialVersionUID = 1L;
    private String apelido;
    private float peso;
    private int altura;
    private boolean vivo;

    //Construtor
    public Animal() {
    }
    /**
     * @return the apelido
     */
    public String getApelido() {
        return apelido;
    }
    /**
     * @param apelido the apelido to set
     */
    public void setApelido(String apelido) {
        this.apelido = apelido;
    }
    /**
     * @return the peso
     */
    public float getPeso() {
        return peso;
    }
    /**
     * @param peso the peso to set
     */
    public void setPeso(float peso) {
        this.peso = peso;
    }
}
```

```
}  
/**  
 * @return the altura  
 */  
public int getAltura() {  
    return altura;  
}  
/**  
 * @param altura the altura to set  
 */  
public void setAltura(int altura) {  
    this.altura = altura;  
}  
/**  
 * @return the vivo  
 */  
public boolean isVivo() {  
    return vivo;  
}  
/**  
 * @param vivo the vivo to set  
 */  
public void setVivo(boolean vivo) {  
    this.vivo = vivo;  
}  
}
```

## O que são Beans de Entidades?

Beans de Entidade (Entity Beans) modelam conceitos de negócios. São utilizados para modelar entidades do mundo real (aluno, curso, pedido, venda, etc.) ou entidades que já estejam representadas no banco de dados através de tabelas. Estes tipos de beans modelam realmente um dado no Banco de Dados onde uma instância do bean representa uma linha na tabela do Banco. Então podemos questionar: Por que não acessar o banco de dados diretamente? Existem muitas vantagens em se usar Beans de Entidade, ao invés de acessar a base de dados diretamente.

Os beans promovem um mecanismo simples de acesso e alteração de dados. É muito mais fácil, por exemplo, executar um método para atualizar um campo de uma entidade do que fazer um comando SQL para isto. Quando um novo bean é criado, um novo registro deve ser inserido na base de dados e uma instância do bean é associada a este dado. Conforme o bean é usado e seu estado é alterado, estas mudanças devem estar sincronizadas com a base de dados. Este processo de coordenação entre os dados no Banco de Dados e aqueles na instância do bean denomina-se persistência.

Finalizando, o que temos que entender de um modo geral sobre Beans de Entidade seria o seguinte:

- É uma representação em memória de um dado persistido em um meio de armazenamento.
- É esperto o suficiente para saber quando ler o dado de um armazenamento e salvar os campos no mesmo.
- Um Objeto pode ser modificado em memória para mudar os valores do dado.

### Observações:

O termo entity bean é usado de uma maneira genérica, pois algumas vezes o mesmo se refere ao dado em memória e em outro momento ao dado armazenado onde alguns autores como ED Roman em seu livro "Mastering Enterprise Beans" faz uma distinção: - A "instância" do bean de entidade é o dado em memória e o "dado" do bean de entidade é o conjunto de dados armazenados fisicamente.

## O que é JPA – Java Persistence API

JPA era parte da especificação EJB3 até o final de 2008, a partir de 2009 a especificação JPA se torna própria, independente de EJB, pois a única relação que JPA possui com EJB são os Beans de entidade.

JPA esta na versão 2.1 e tem sua definição na JSR 338 (Java Specification Requests): JavaTM Persistence API, Version 2.1. Hoje a JPA é a principal referência para MOR. (<https://jcp.org/en/jsr/detail?id=338>)

*“Enterprise JavaBeans (EJB) é um componente da plataforma JEE que roda em um container de um servidor de aplicação. Seu principal objetivo consiste em fornecer um desenvolvimento rápido e simplificado de aplicações Java, com base em componentes distribuídos, transacionais, seguros e portáteis. Atualmente, na versão 3.1, o EJB tem seu futuro definido conjuntamente entre grandes empresas como IBM, Oracle e HP, como também por uma vasta comunidade de programadores numa rede mundial de colaboração sob o portal do JCP.”*

## Regras de Beans de Entidade para JPA

Todo Bean de Entidade deve atender a uma lista de requisitos para que possa ser encarado como um EJB Entity Bean e ser persistido (no caso de se utilizar EJB3). Os requisitos são listados abaixo:

1. A classe deve estar anotada com `@javax.persistence.Entity`.
2. A classe deve ter um identificador unico anotado com `@javax.persistence.Id`
3. A classe deve ser pública ou protegida e possuir um construtor sem parâmetros (default). Pode haver outros construtores, mas o default deve estar presente.
4. A classe não pode ser declarada como final, bem como métodos ou atributos.
5. A classe deve implementar a interface `Serializable`. No caso de se utilizar EJB3.
6. As classes podem herdar de outras classes de entidade ou de classes que não sejam de entidade.
7. Atributos de classes que representam entidades precisam ser declarados como privados, protegidos ou pacote e serem acessados por somente através de métodos públicos.

## Considerações Quanto aos Atributos de Entidades

Os atributos das entidades persistentes podem ser dos seguintes tipos:

- Tipos primitivos Java
- `java.lang.String`
- Outros tipos serializáveis, incluindo:
  - classes wrappers dos tipos primitivos
  - `java.math.BigInteger`
  - `java.math.BigDecimal`
  - `java.util.Date`
  - `java.util.Calendar`
  - `java.sql.Date`
  - `java.sql.Time`
  - `java.sql.Timestamp`
  - tipos de dados serializáveis definidos pelo usuário
  - `byte[]`
  - `Byte[]`
  - `char[]`
  - `Character[]`
- Tipos enum
- Outras entidades e/ou coleções de entidades

Os atributos que se deseja persistir junto com a entidade devem ser compatíveis com as regras/tipos apresentados acima e devem ser *anotados* com suas respectivas anotações, mostradas mais adiante. Já aqueles atributos que não se deseja persistir devem ser anotados com `@javax.persistence.Transient`.

As propriedades da entidade que se deseja persistir devem seguir as convenções de componentes JavaBeans.



## O Que São POJOs?

POJO significa Plain Old Java Object, ou seja, classes simples escritas em Java. Na especificação Java Persistence API, as entidades devem ser escritas como POJOs. Objetos destas classes são criados com o operador new, como já estamos acostumados a fazer, porém, não são persistidos em um banco de dados até que estejam associados com um EntityManager. Porém, para isto algumas anotações devem ser acrescentadas nas classes. Veja abaixo o exemplo:

```
package br.com.targettrust.cadastro.entidades;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;

/**
 * Classe de exemplo do Curso de Fundamentos Java
 * @author Cássio Trindade
 * @since Setembro/2014
 */
@Entity
public class AnimalEntityBean implements Serializable { //serelizado

    private static final long serialVersionUID = 1L;
    @Id
    private long id;
    private String apelido;
    private float peso;
    private int altura;
    private boolean vivo;

    //Construtor
    public AnimalEntityBean() {

    }

    ... gets e sets ...
}
```

# Mapeamento Objeto Relacional em Java

Frameworks para MOR abordam o problema de armazenar estados de objetos em um SGBDR. Um dos objetivos desses frameworks é reduzir o tempo necessário para concluir o processo de persistência dos dados de uma aplicação, que ao modo tradicional parece consumir tempo demais. Para Brian Sam-Bodden, no livro intitulado “Desenvolvendo em POJOs, do Iniciante ao Profissional”, o processo de persistir os dados usando frameworks MOR depende de diversos fatores:

- O quão proximamente o esquema do banco de dados está relacionado ao modelo de objetos.
- Se o esquema é anterior à aplicação.
- Se o esquema é um legado com chaves naturais ou chaves primárias compostas.
- Qual o grau de normalização do esquema.
- Quanta influência o esquema teve no projeto do modelo de objetos e na interface do usuário (e vice-versa).

É importante o modelador perceber que a qualidade da aplicação depende de uma série de fatores. Um dos fatores principais é a definição de um modelo de domínio, “as classes de análise definidas no processo RUP por exemplo”. Um modelo de domínio bem definido certamente possibilitará que o código da aplicação, assim como a base de dados, seja flexível a mudanças nos requisitos do sistema. No entanto, de acordo com Brian:

“A modelagem do domínio é apenas uma abordagem para resolver o problema. Muitas das aplicações típicas da internet que usamos diariamente são mais orientadas a dados relacionais/procedurais, e são mais apropriadas para serem construídas de uma forma que explore essas características, usando uma linguagem que possa aproveitar o modelo relacional ... Para aplicações simples ... onde o desempenho seja prioridade número um, SQL bruto ainda pode ser a melhor escolha”.

O modelador deve portanto saber que ao utilizar a tecnologia de persistência com JPA estará tomando uma importante decisão, que poderá implicar em muitos efeitos indesejados nas aplicações desenvolvidas, caso os requisitos não funcionais das aplicações não tenham sido corretamente analisados.

A seguir são apresentados alguns frameworks e recursos para MOR, usados no desenvolvimento de aplicações Java.

## JPA

A Java Persistence API é uma especificação da Sun Microsystems para persistência de objetos Java em qualquer banco de dados relacional. Para utilizar a JPA, é necessária a versão 1.5 (também conhecida como Java 5) de J2SE ou uma versão maior, dado que ela utiliza generics e anotações. Existem muitos fornecedores de implementações da API JPA.

## OpenJPA

OpenJPA é uma implementação livre desenvolvida pela Apache Software Foundation da JPA. Qualquer framework para MOR pode fazer uso da OpenJPA. As anotações que você utilizará nos exemplos desse curso são interfaces JPA implementadas pelo OpenJPA.

OpenJPA pode ser utilizada em ambiente J2SE e em ambiente J2EE, integrada com o JBoss EJB3. Para utilizar um framework para MOR com a JPA, são necessárias bibliotecas/jars que contenham a implementação das interfaces de JPA. Assim, procure utilizar a biblioteca OpenJPA.

Para mais informações sobre a OpenJPA, consulte <http://openjpa.apache.org>

## Hibernate

O Hibernate é um framework open-source, que fornece tanto persistência quanto a capacidade de consulta de objetos. Ele foi desenvolvido por Gavin King, no final de 2001, como resultado de suas experiências trabalhando com versões anteriores de EJBs. Em 2003, o Hibernate se tornou parte do servidor JBoss e é usado como base do mecanismo EJB3 CMP do mesmo servidor.

Para utilizar o Hibernate, o programador não precisa implementar nenhuma interface especial ou estender uma classe específica. Assim, o programador deve trabalhar com POJOs, o que facilita o desenvolvimento orientado a objetos e a modularização do sistema.

O Hibernate oferece um modelo de declaração programática, em que o desenvolvedor escreve código para informar ao sistema de gerenciamento do mesmo que determinadas classes da aplicação são entidades a serem persistidas num banco de dados. Além disso, o programador pode optar por configurar a persistência em arquivos XML.

Os mapeamentos podem ser feitos no Hibernate de muitas formas. Uma forma muito utilizada é especificar os mapeamentos através de anotações (@Annotations) ou em arquivos XML com extensão (.hbm).. Além disso, o Hibernate suporta mapeamentos com o padrão JPA, adicionando outros recursos e anotações (@Annotations) para suprir algumas lacunas nesse padrão, como consultas por critério.

Para mais informações, consulte <http://www.hibernate.org>

## TopLink

O TopLink é um framework desenvolvido pela Oracle para implementar as características de MOR. Atualmente é disponibilizado por esta empresa, com a licença free.

Tanto o TopLink como o Hibernate podem ser encaixados em uma arquitetura, como

As características do TopLink são semelhantes às do Hibernate. No entanto, TopLink têm um diferencial: além de administrar a persistência dos dados em um SGBDR, o mesmo gerencia também para bancos orientado a objetos. As seguintes características estão presentes neste framework:

- Persiste POJOs para virtualmente qualquer banco de dados suportado pelo driver JDBC;
- Persiste POJOs para virtualmente qualquer fonte de dados não relacional pelo conector adaptador J2C usando sistemas de informação (Enterprise Information Systems - EIS) indexados, mapeados ou gravados em XML;
- Aplica conversões em memória entre POJOs e esquemas XML;
- Utiliza a ferramenta de mapeamento gráfico TopLink Workbenck para mapear qualquer definição relacional ou não relacional de um banco.

Para mais informações, consulte:

<http://www.oracle.com/technetwork/middleware/toplink/overview/index.html>

## EclipseLink

O EclipseLink JPA é um plugin disponível no Eclipse que fornece aos desenvolvedores normas baseadas em solução de persistência objeto-relacional com suporte adicional para muitas características avançadas, ele tem sua construção baseada no TopLink da Oracle. EclipseLink JPA fornece suporte avançado para os principais bancos de dados relacionais e recipientes de Java.

Para mais informações, consulte:

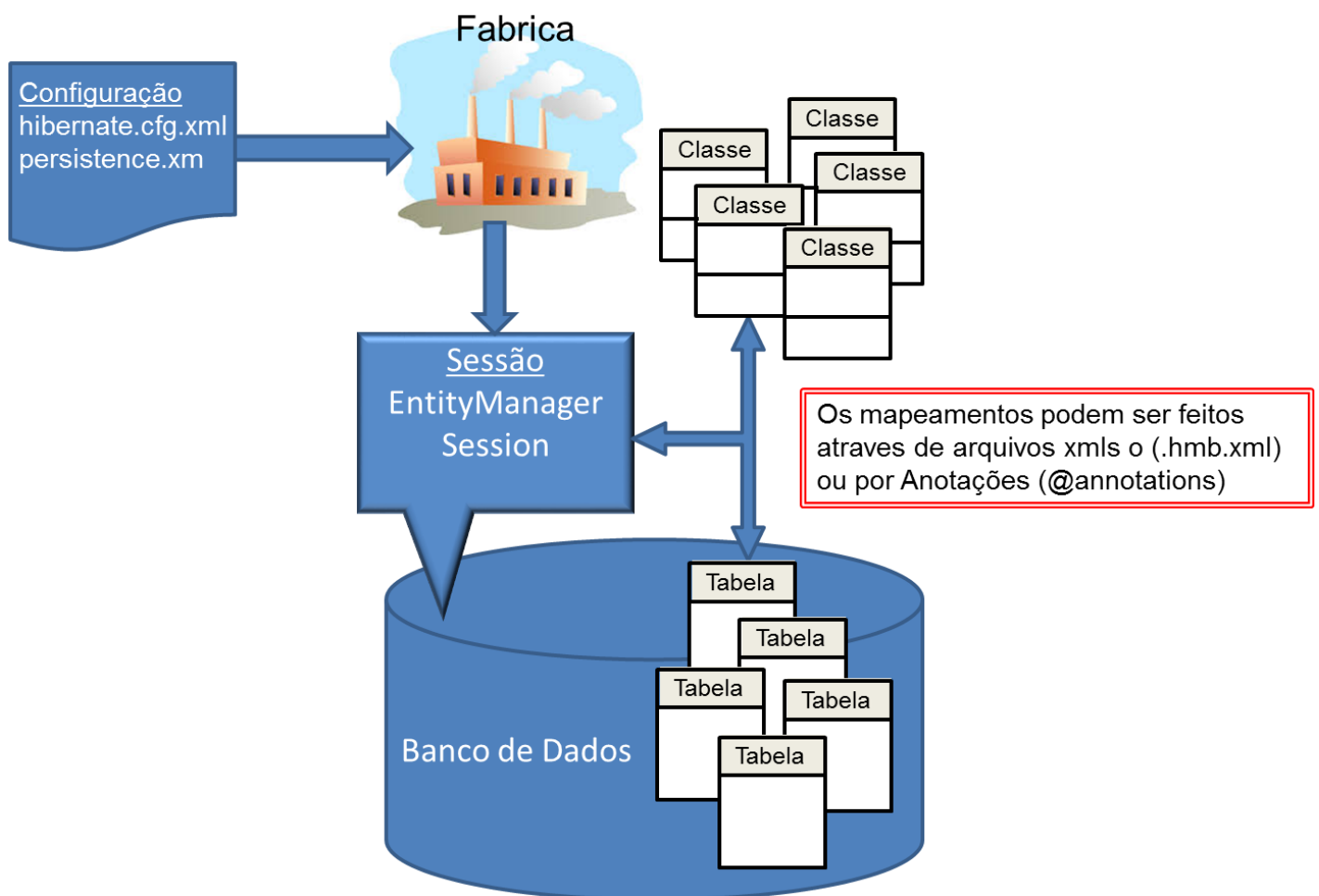
<http://www.eclipse.org/eclipselink/jpa.php>

# O Gerenciamento de Persistência de Frameworks Java para MOR

Os frameworks Hibernate, TopLink e EclipseLink seguem a mesma estrutura de organização. Assim, o contexto do framework, o ciclo de vida dos objetos e o gerenciador de persistência se aplicam em ambos os casos.

## Contexto

O contexto define como o framework, em conjunto com sua aplicação, irá gerenciar os objetos que precisam ser persistidos em um determinado banco de dados. A figura apresenta esse contexto para o contexto do Hibernate. No entanto, o mesmo exemplo, com leves mudanças, pode ser aplicado para o contexto do TopLink.



Claro que cada framework possui um arquivo em especial com nomes e propriedades diferentes. No entanto, todos precisam lidar basicamente com as mesmas informações.

Na figura percebe-se a existência de arquivos `hibernate.cfg.xml` ou `presistence.xml`. Estes são arquivos que informam ao Hibernate / JPA as características relacionadas com a persistência dos objetos em bancos de dados relacionais. Estes arquivos, no caso de uma aplicação WEB ser gerenciada pelo Springframework, não são necessários. Assim, o conteúdo de tal arquivo seria especificado em um arquivo de configuração do Spring, como o `applicationContext.xml`, por exemplo.

As classes representadas na figura são as que o modelador irá marcar com as anotações JPA. (elas podem constar nos arquivos `.hbm.xml` para persistência, usado anteriormente da existência das anotações).

Além de especificarmos as classes entidades no arquivo `cfg` (`hibernate.cfg.xml` ou `presistence.xml`), também precisamos configurar o banco de dados. Assim, requisitos como o nome do banco, url, usuário e senha precisam constar neste arquivo. Todos os arquivos `cfg` são gerenciados por uma única instância do framework de persistência. Esta instância é chamada de fábrica de sessão, que intermediará tudo o que está relacionado com banco de dados na aplicação.

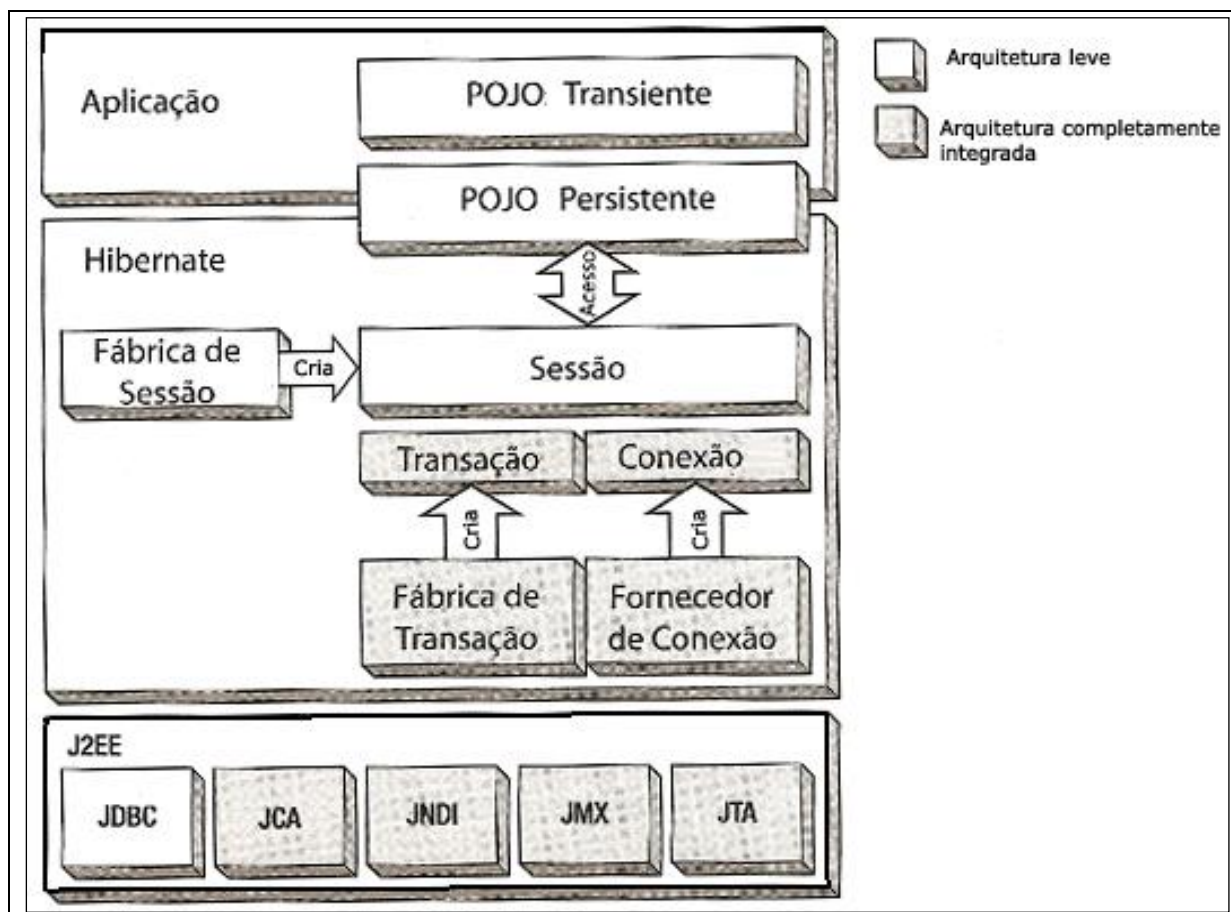
Para cada serviço que queremos que o framework de persistência realize, seja um consulta ou uma atualização de dados de um objeto, precisamos requisitar à fábrica de sessão (ou Application Manager) que ela nos abra uma sessão. Esta sessão é similar à conexão que abrimos quando vamos executar um comando SQL. No entanto, esta sessão serve para executarmos comandos tanto SQL quanto comandos do próprio framework de persistência.

Cada sessão pode durar o tempo que for necessário para realizar as operações de persistência de determinados objetos. Você pode manter uma sessão aberta para salvar todos os objetos que estão sendo manipulados em uma determinada tela/formulário, por exemplo. A sessão manterá todos os objetos inicializados dentro dela com um estado em que qualquer modificação pode ser propagada para o banco com simples comandos de atualização.

No momento em que o usuário concluir sua atividade, então a sessão pode ser liberada, comitando as modificações.

## Gerenciador

A Figura abaixo apresenta o esquema de gerenciamento de persistência do Hibernate. Os elementos em destaque na figura apresentam arquiteturas que são completamente integradas com uma determinada tecnologia, como, por exemplo, com um SGBDR, enquanto os outros elementos possuem uma arquitetura leve, transparente, e que funcionam de maneira independente de tecnologia.



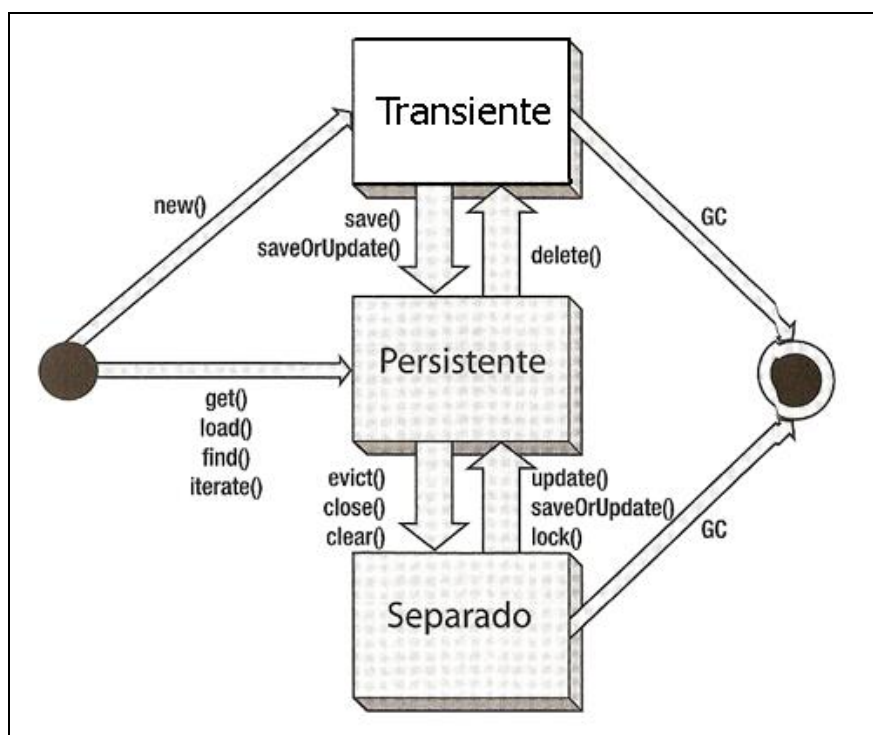
A camada de aplicação possui objetos (POJOs) que precisam ser persistidos no banco de dados. Tais objetos estão em um estado transiente, ou seja, seus dados ainda não foram persistidos em tabelas. No momento em que precisarmos persistir estes dados, então abriremos uma sessão com o framework de persistência. Utilizaremos esta sessão aberta para gravar os dados dos POJOs nas tabelas em que estes objetos estão mapeados. Quando persistimos objetos, então a sessão do framework torna-os persistente

Estar no estado de persistente significa que a sessão gravou os dados do objeto e, ao mesmo tempo, está mantendo um vínculo entre o objeto na memória e o banco de dados. Assim, enquanto a sessão estiver aberta, você poderá requisitar para ela que atualize no banco as modificações feitas no objeto. No entanto, uma sessão aberta pode consumir bastante memória RAM, dado que precisa guardar o estado de persistência do objeto.

Repare que na figura a sessão é que gerencia as transações, assim como a conexão com o banco de dados. Logo, a própria sessão trata de aspectos como pool de conexões de banco. O desenvolvedor pode deixar que a sessão gerencie as transações do banco, ou se preferir pode controlar programaticamente as transações.

## Ciclo de Vida

Agora que você teve o primeiro contato com sessões de frameworks de persistência, é possível compreender os estados em que os objetos POJOs podem estar durante o ciclo de vida de uma aplicação. São três estados possíveis: Transiente, Persistente e Separado (Detached).



A figura apresenta um diagrama de estados de POJOs no contexto de um framework de MOR. Quando o objeto é criado, ele entra em um estado transiente. O estado transiente informa que o objeto não tem relação alguma com o banco de dados relacional. Tipicamente, POJOs conterão um atributo que representa a chave primária do objeto. No estado transiente, os POJOs não possuem valores de chaves em seus atributos que representam as chaves primárias.

Para atingir o estado de persistente, deve-se primeiro requisitar para a fábrica de sessão ou 'Application Manager', dependendo do contexto do framework, que abra uma sessão. Tendo a sessão é possível salvar ou atualizar este objeto, antes transiente, mas agora persistente. Ao salvar o objeto no banco de dados, a sessão irá setar no(s) atributo(s) chave primária os valores das chaves do banco. Isso significa que o POJO atingiu o estado de persistente. Ao deletar o objeto do banco, a sessão elimina a chave primária, sincronizando isso também no POJO. Ao remover o POJO da memória, o *garbage collector* irá eliminar a memória alocada pelo mesmo e a sessão irá desvinculá-lo do gerenciamento.

Do estado de persistente, nosso POJO pode atingir o estado de separado (detached) ou de transiente. O estado detached é atingido quando fechamos ou limpamos a sessão. Nesse caso, não existe mais sessão para gerenciar as modificações que são feitas no objeto.



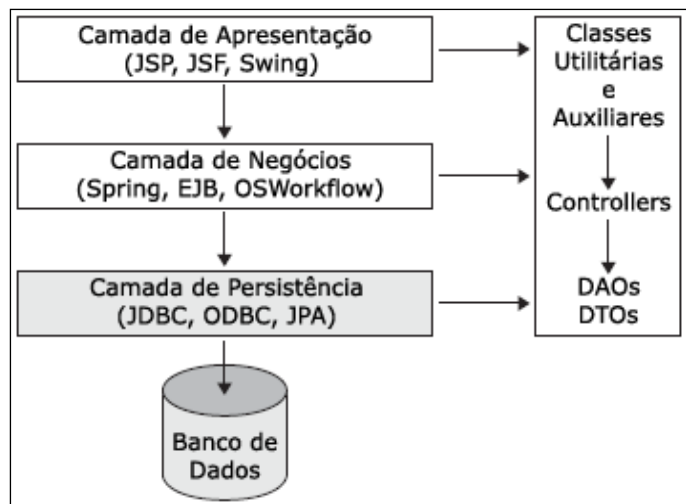
Diferente do estado de transiente, um objeto separado ainda tem seu estado no banco de dados, porém não é gerenciado mais pela sessão como era no estado persistente.

Do estado separado, o POJO pode tornar-se novamente persistente, desde que uma nova sessão seja aberta e que uma das operações da sessão seja chamada: gravação por (salvar, salvar ou atualizar, travar) e recuperar estado por (carregar, encontrar e iterar sobre coleções). A operação carregar irá popular o POJO com os dados do banco de dados de acordo com a chave primária contida no objeto, em seu atributo que guarda a chave primária. Já as operações salvar e salvar ou atualizar irão gravar os dados no banco de dados já sincronizando no objeto o estado do banco.

Se apagarmos o objeto no estado detached, o dado no banco não é apagado. Apenas o *garbage collector* irá eliminar o espaço de memória alocado por ele.

## Compreendendo a Arquitetura

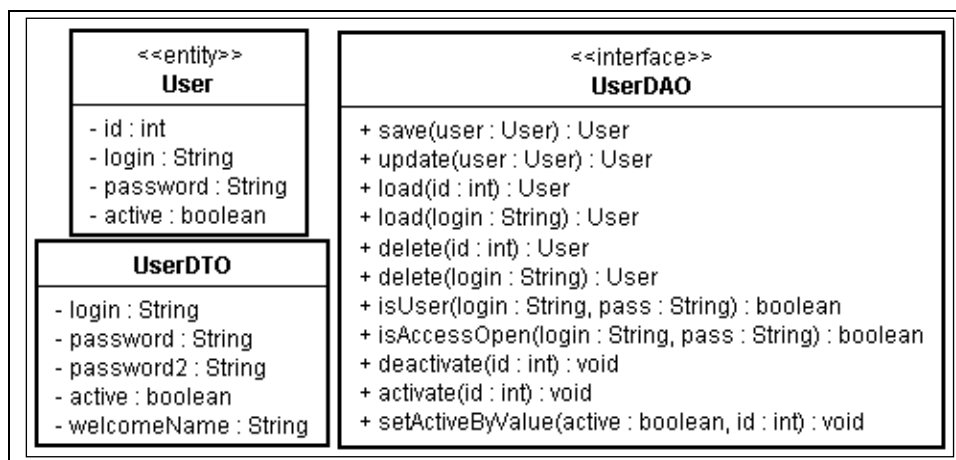
O esquema de uma arquitetura que recomendamos ao desenvolvedor é apresentado na figura ao lado. Este esquema define três camadas para uma aplicação: camada de apresentação, camada de negócios e camada de persistência. Em cada camada é possível utilizar muitas tecnologias, dentre as quais algumas foram especificadas entre parênteses nas camadas. Perceba que o conteúdo de JPA é abordado na última camada da aplicação, a camada de persistência.



Nesse curso, o aluno estará engajado em construir a camada de persistência em um nível independente de banco de dados e mesmo de frameworks para MOR. Para desenvolver a camada de persistência, além de marcarmos as entidades com anotações JPA, podemos desenvolver classes DAO e também DTO para organizar melhor o nosso trabalho.

## O Que é DAO e DTO?

DAO significa *Data Access Object*. DAO é um padrão arquitetural que é utilizado para representar interfaces que irão oferecer funcionalidades relacionadas com entidades da aplicação na camada de persistência. Abaixo apresentmos um exemplo de DAO, uma interface chamada *UserDAO*, que define operações relacionadas com a entidade *User* e com persistência e recuperação de dados dessa entidade. O padrão DAO define que para cada classe de entidade é possível especificar uma interface que define todas as operações de persistência que serão utilizadas na aplicação e que estão relacionadas com cada classe de entidade. O nome da interface DAO inicia pelo nome da entidade seguido de DAO ou Persistence, como preferir o programador.



DTO significa *Data Transfer Object*. Significa que um objeto DTO tem um único propósito: transferir dados de uma camada da aplicação para outra. Analisando a questão da entidade *User*, é provável que na camada de apresentação, durante um cadastro de usuário, seja necessário que o usuário informe a senha duas vezes para que possamos conferir na camada de negócios se as senhas são iguais. No entanto, a entidade usuário possui apenas uma senha. Seria incorreto colocarmos outro atributo na entidade *User* para guardar a segunda senha, que está presente apenas no formulário de cadastro de usuário e que é checada na camada de negócios. Perceba que na camada de persistência não tem sentido termos duas senhas para o usuário.

Para este caso é importante o uso de DTOs. Assim, pode-se escrever uma nova classe chamada de *UserDTO*, apenas para transportar os dados da interface de usuário para a camada de negócio. Na camada de negócio, pode-se criar e popular DTO com dados e chegar as regras de negócio. Em caso de conformidade com tais regras, deve-se então criar um objeto do tipo da entidade e repassá-lo para a camada de persistência.

Uma vantagem do DTO é que ele também economiza o tráfego de dados quando se está utilizando chamadas remotas de operações, como RMI de EJB ou http Remote do Springframework.













## Ambiente Java EE e SE

A JPA pode ser utilizada tanto em ambiente Java EE como em SE, ou ambos. Para utilizá-la em ambiente EE, basta que configuremos o framework de persistência em arquivos XMLs de inicialização. As entidades marcadas com a JPA também devem ser informadas neste arquivo. Para ambiente SE, podemos configurar o framework em XMLs ou de maneira programática, ou seja, escrevendo código Java. Ambientes EE e SE podem ser integrados com muitas tecnologias, como EJB e Spring, por exemplo.

## Instalação e Configuração da Arquitetura

Para iniciarmos nossos trabalhos de Mapeamento Objeto Relacional, temos que primeiramente definir qual framework a ser utilizado, neste curso faremos o uso do Hibernate 4.3.6 que podem ser obtidos do site <http://www.hibernate.org/downloads>.

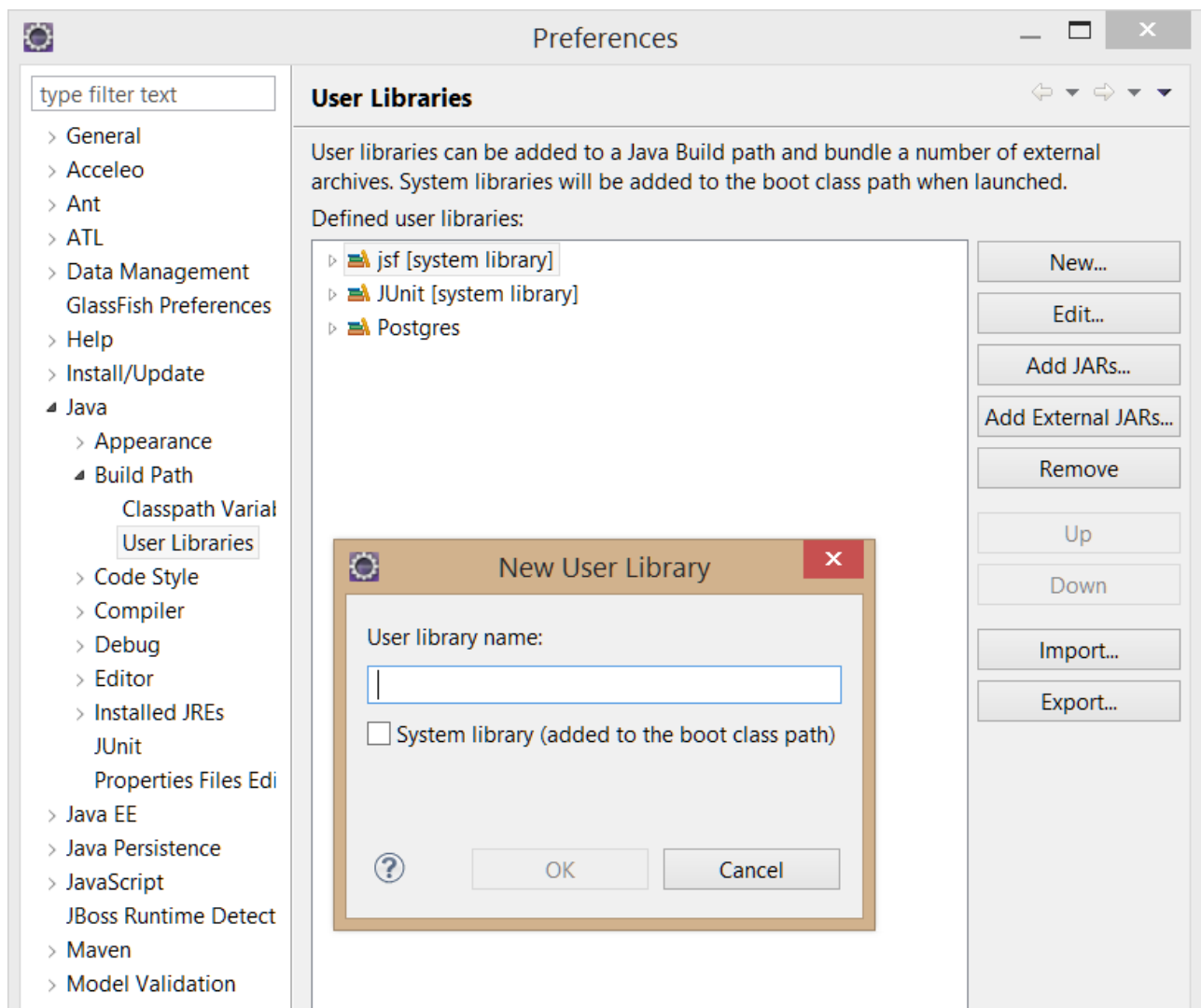
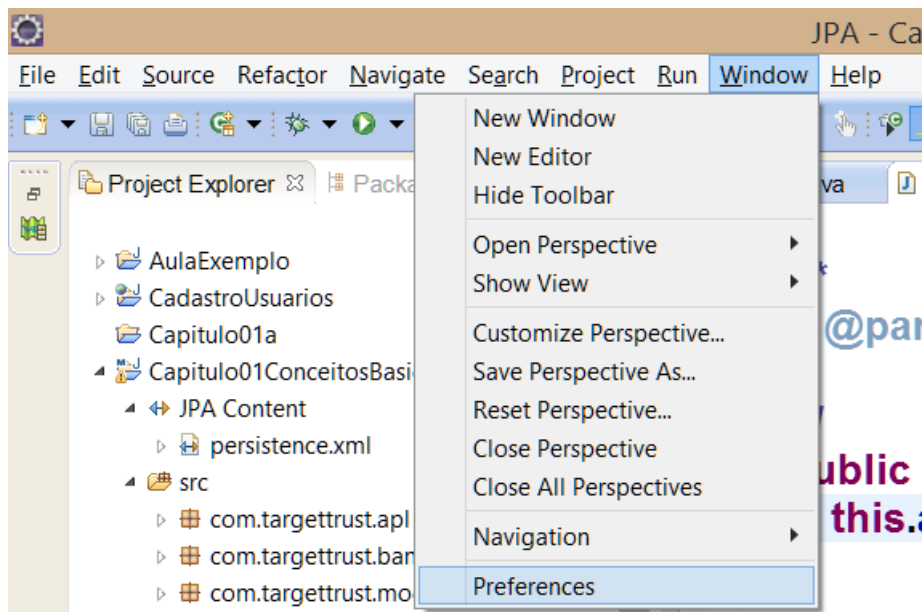
Para esse curso está disponível um arquivo compactado Hibernate.4.3.6.Final.zip, em Anexos, quando realizamos o download os arquivos necessários se encontram em várias pastas, esse anexo já está com os arquivos necessários separados.

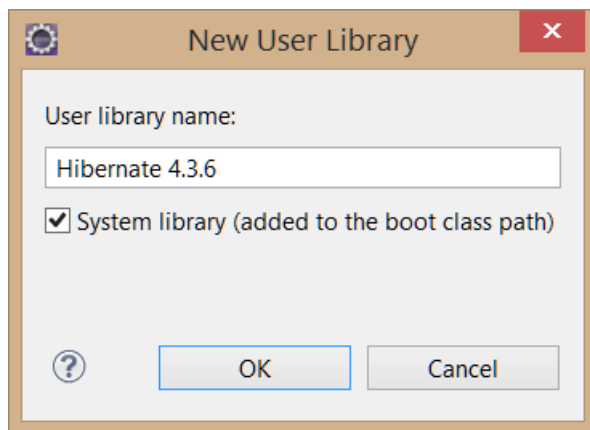
-  antlr-2.7.7.jar
-  dom4j-1.6.1.jar
-  hibernate-commons-annotations-4.0.5.Final.jar
-  hibernate-core-4.3.6.Final.jar
-  hibernate-entitymanager-4.3.6.Final.jar
-  hibernate-jpa-2.1-api-1.0.0.Final.jar
-  jandex-1.1.0.Final.jar
-  javassist-3.18.1-GA.jar
-  jboss-logging-3.1.3.GA.jar
-  jboss-logging-annotations-1.2.0.Beta1.jar
-  jboss-transaction-api\_1.2\_spec-1.0.0.Final.jar
-  xml-apis-1.0.b2.jar

Descompacte o arquivo Hibernate.4.3.6.Final.zip em uma pasta no seu desktop.

Próximo passo é criar uma Lib de usuário no Eclipse para utilizarmos essas bibliotecas em nossos projetos JPA.

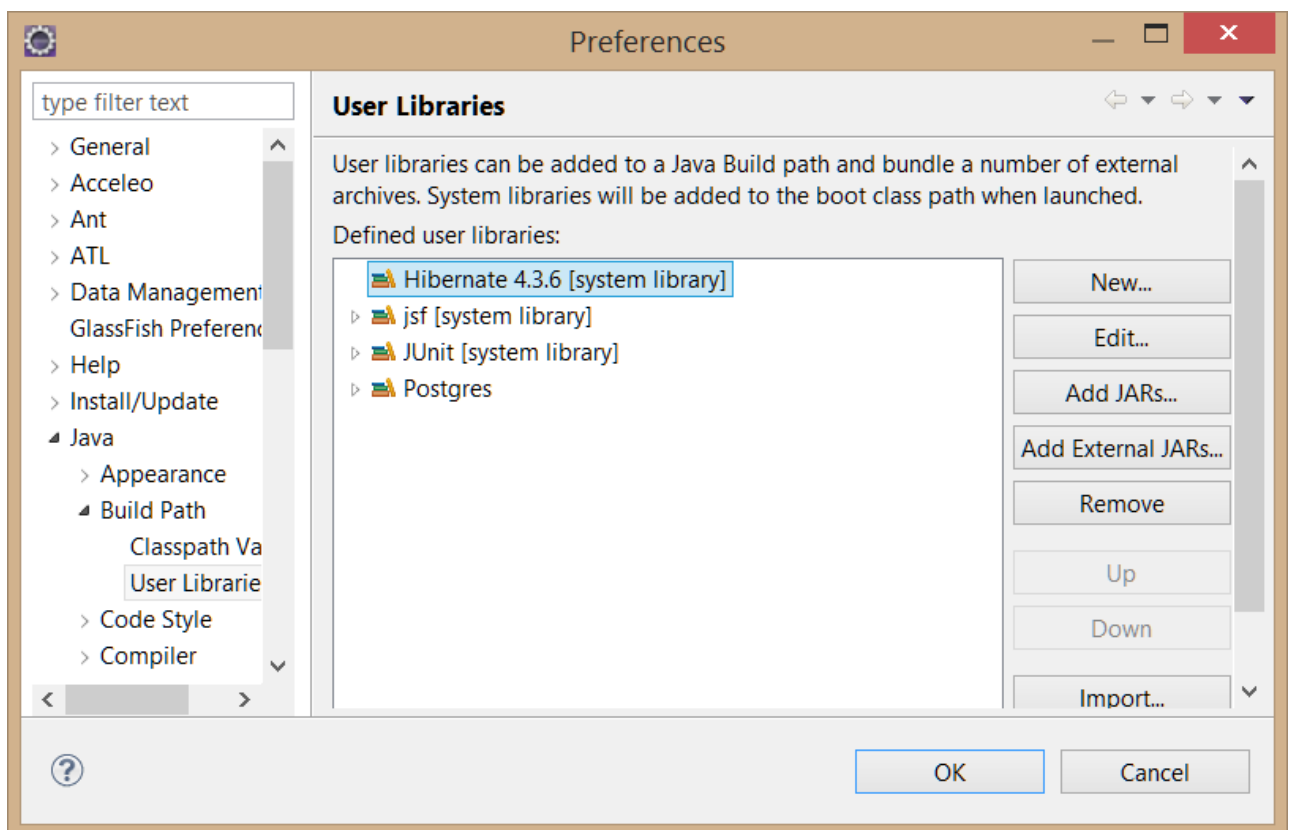
1. Vá no menu **Windows / Preferences** no Eclipse (Eclipse versão Kepler JavaEE).
2. Na tela que abrir vá na opção **Java / Build Path / User Libraries**.
3. Clique no Botão **NEW**.
4. Na caixa de diálogo coloque em **User library** name: Hibernate 4.3.6 marque o **flag System library** e clique no botão **OK**.

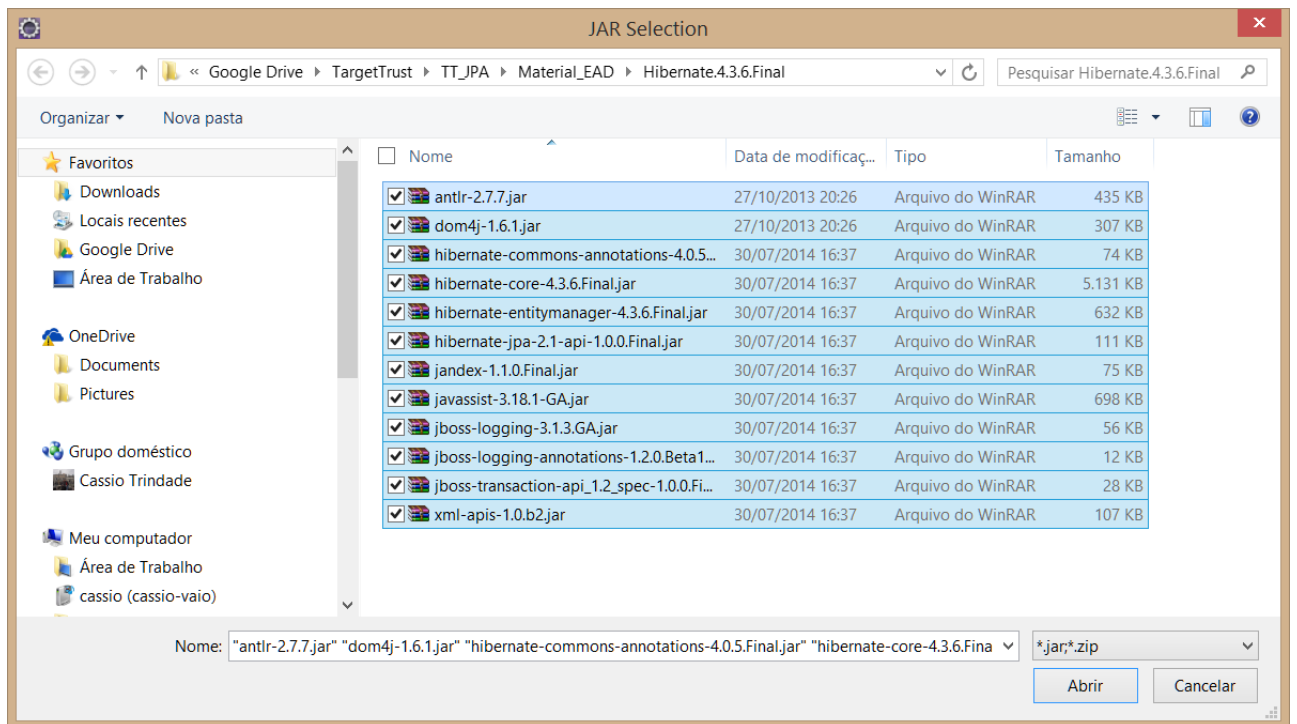


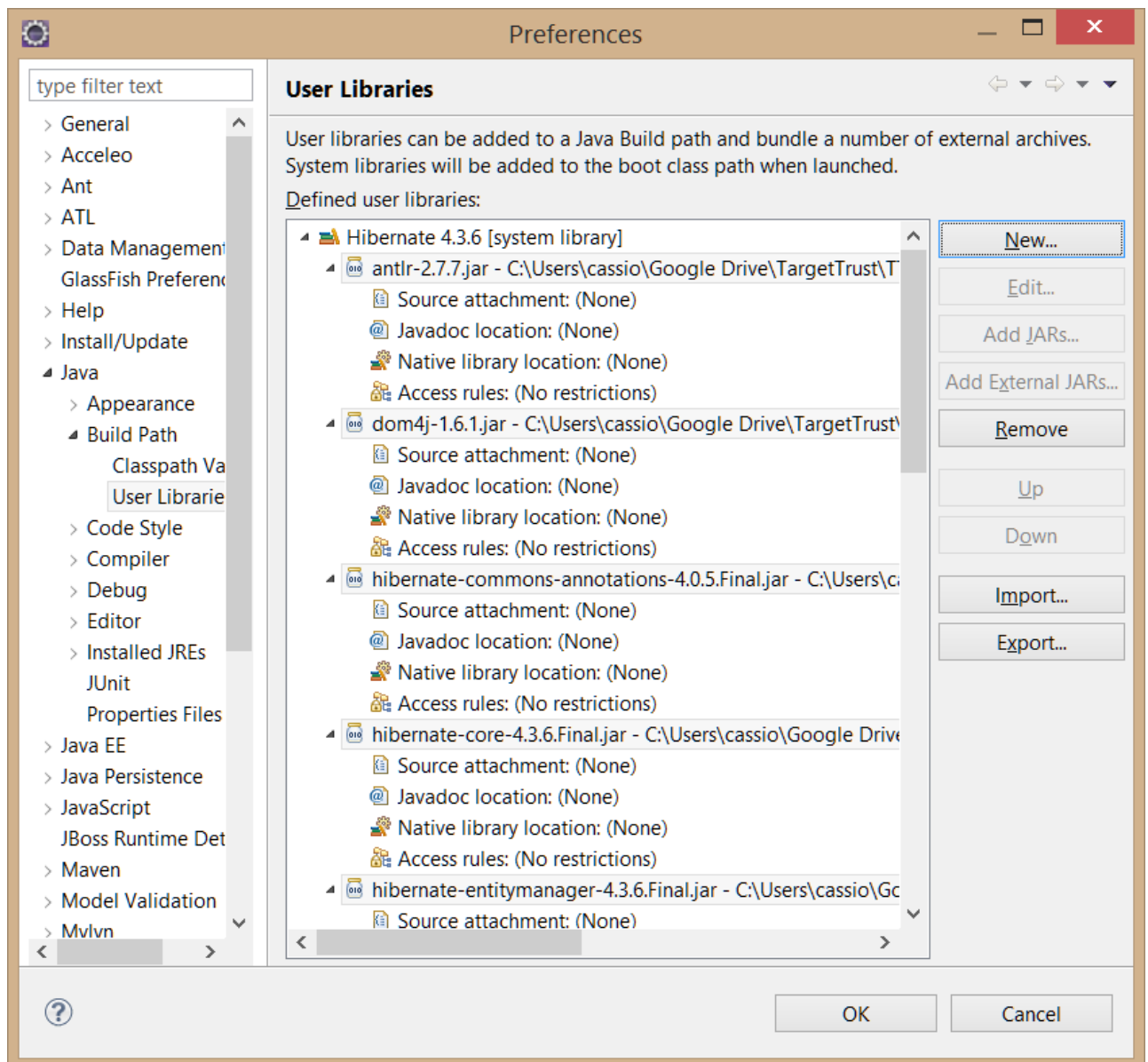


Agora vamos adicionar os arquivos do Hibernate

1. Com a Lib do Hibernate 4.3.6 selecionado clique no Botão [Add External JARs](#).
2. Vá até a pasta onde estão os arquivos que você descompactou do arquivo Hibernate.4.3.6.Final.zip
3. Selecione todos os arquivos e clique em [Abrir](#)
4. Após os arquivos listados clique em [OK](#)







Pronto agora estamos prontos para iniciarmos nos trabalhos de Mapeamento Objeto Relacional.