UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORDENAÇÃO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ÉDINA MARIA DAS NEVES
THYAGO HENRIQUE PACHER

ANÁLISE E COMPARAÇÃO DE *FRAMEWORKS* DE PERSISTÊNCIA

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2012

ÉDINA MARIA DAS NEVES
THYAGO HENRIQUE PACHER

ANÁLISE E COMPARAÇÃO DE *FRAMEWORKS* DE PERSISTÊNCIA

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas da COADS da Universidade Tecnológica Federal do Paraná.

Orientadora: Profª. MSc. Simone de Almeida

PONTA GROSSA
2012

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa

Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

ANÁLISE E COMPARAÇÃO DE FRAMEWORKS DE PERSISTÊNCIA

por

ÉDINA MARIA DAS NEVES
THYAGO HENRIQUE PACHER

Este Trabalho de Conclusão de Curso (TCC) foi apresentado(a) em 06 de junho de 2012 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O(a) candidato(a) foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Simone de Almeida
Profª. Orientadora

_____
Marcos Vinicius Fidelis
Membro titular

_____
Cristian Cosmoski Rangel de Abreu
Membro titular

_____
Helyane Bronoski Borges
Responsável pelos Trabalhos de Conclusão de Curso

_____
Simone de Almeida
Coordenadora do Curso
UTFPR - Câmpus Ponta Grossa

- O Termo de Aprovação assinado encontra-se na Coordenação

Dedicamos este trabalho à nossas
famílias, pelos momentos de ausência.

AGRADECIMENTOS

Agradeço a nossa orientadora Profª. Dr. Simone de Almeida, pela sabedoria com que me guiou nesta trajetória, e a todos os que de alguma forma contribuíram para a realização desta pesquisa.

RESUMO

NEVES, Édina M.; PACHER, Thyago H. Análise e Comparação de *Frameworks* de Persistência. 2012. 110 f. Trabalho de Conclusão de Curso em Tecnologia em Análise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2012.

Devido às diferenças existentes entre o modelo Orientado a Objetos, utilizado no desenvolvimento de *software*, e o modelo Relacional dos Bancos de Dados, surgiram técnicas de mapeamento objeto-relacional (ORM) que permitem uma melhor comunicação entre os dois modelos. Contudo, existem várias especificações e *frameworks* de mapeamento objeto-relacional que implementam as técnicas de ORM, por isso se faz necessário decidir qual implementação utilizar ao desenvolver um *software*. Este trabalho tem como objetivo analisar e comparar algumas das soluções existentes para a plataforma Java. Foram escolhidos os *frameworks* de persistência: Hibernate, Spring, TopLink e DataNucleus. As comparações foram feitas com base nas informações coletadas durante a confecção do referencial teórico e segundo critérios gerais para análise de *frameworks* específicos de mapeamento objeto-relacional existentes na literatura sobre o assunto e testados em uma aplicação que se utiliza dos recursos disponíveis na Orientação a Objeto. Ao final é apresentado um resumo dos principais benefícios e restrições de cada *framework* analisado neste estudo.

Palavras-chave: *Framework* de Persistência. Hibernate. Spring. Toplink. DataNucleus.

ABSTRACT

NEVES, Édina M.; PACHER, Thyago H. Analysis and Comparison of Persistence *Frameworks*. 2012. 110 f. Completion of Course Work in Technology Analysis and Systems Development – Federal Technological University of Parana. Ponta Grossa, 2012.

Due to differences between the Object Oriented model, used in software development, and the model of Relational Databases, emerged techniques object-relational mapping(ORM) that allow better communication between the two models. However, there are several specifications and frameworks of object-relational mapping techniques that implement ORM, so it is necessary to decide which implementation to use when developing software. This study aims to analyze and compare some of the existing solutions for the Java platform. We chose the persistence frameworks: Spring, Hibernate, TopLink and DataNucleus. Comparisons were made based on information collected during the making of the theoretical framework and criteria for analysis of general and specific frameworks of object-relational mapping in the literature on the subject and tested in an application using resources available in the Guidance object. At the end is a summary of the main benefits and limitations of each framework analyzed in this study.

Keywords: Persistence Framework. Hibernate. Spring. Toplink. DataNucleus.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS

AOP	*Aspect Oriented Programming* (Programação Orientada a Aspectos)
API	*Application Program Interface* (Interface de Programa Aplicativo)
DAO	*Data Access Object* (Acesso aos Dados do Objeto)
EJB	*Enterprise JavaBean*
HQL	*Hibernate Query Language* (Linguagem de Seleção Hibernate)
JAR	*Java archive* (Arquivo Java)
JDB	*Java Database Connectivity* (Conectividade Java Banco de Dados)
JDO	*Java Data Object* (Objeto de Dados Java)
JDOQL	*JDO Query Language* (JDO Linguagem de Seleção)
JNDI	*Java Naming and Directory Interface*
JPA	*Java Persistence API* (API de Persistência Java)
JPQL	*Java Persistence Query Language* (Linguagem de Seleção da Persistência Java)
OO	Orientação a Objetos
ORM	*Object Relational Mapping* (Mapeamento Objeto Relacional)
POJO	*Plain Old Java Object* (Objeto Java no Plano Velho)
SGBD	Sistema Gerenciador de Banco de Dados
SQL	*Structured Query Language* (Linguagem de Consulta Estruturada)
XML	*Extensible Markup Language* (Linguagem de Marcação Extensível)

SUMÁRIO

1. INTRODUÇÃO

O uso da metodologia de orientação a objetos (OO) se proliferou no desenvolvimento de software, provocando uma mudança na estruturação e organização da informação. Contudo, a maioria das aplicações demanda o armazenamento e a recuperação de informações em um mecanismo de persistência. Devido à prevalência do banco de dados relacionais no gerenciamento de dados, seu uso é frequentemente exigido, em vez dos bancos de dados de objetos, pois é perceptível a maturidade e confiabilidade dos SGBD' s adquirida após anos de desenvolvimento e ajustes de desempenho.

A lógica da aplicação, que representa os processos de negócio, é projetada e implementada utilizando ferramentas orientadas a objetos. Já a informação tratada pelos processos de negócio utiliza SQL (*Structure of Query Language*) para armazenar, recuperar e manipular dados em uma base de dados relacional. Portanto, cada informação recuperada deve passar por um processo de tradução de sua representação original para sua representação no modelo OO. Inversamente, para dados representados no modelo OO que devem ser persistidos, isto é, neste caso, gravados no banco de dados, deverá ocorrer à tradução da informação da representação OO para a representação relacional.

As propostas de solução para este desencontro tecnológico convergem para o conceito de uma camada de abstração de acesso a dados, diminuindo o acoplamento da aplicação em relação ao mecanismo de armazenamento de dados.

O desenvolvimento desse trabalho visa fornecer parâmetros aos desenvolvedores de sistemas quanto ao uso de *frameworks* de persistência, sendo o seu escopo limitado ao estudo de *frameworks* voltados ao desenvolvimento para ambientes *desktops,*e ainda uma noção de como desenvolver um programa com os *frameworks* em todos os métodos que envolvem a manipulação com bancos de dados relacionais, que seriam:

- inserirObjeto: possibilita inserir um novo objeto, já mapeado dentro do banco;
- excluirObjeto: traz a possibilidade de excluir um objeto, mas antes disso ele deve estar devidamente instanciado, que pode ser realizado através de uma busca;
- buscarObjeto: por meio de um critério específico, ele procura determinado objeto dentro do banco de dados;

- atualizarObjeto: possibilita a atualização de dados dentro do banco de dados, obviamente, esse objeto deve estar instanciado.

Dos *frameworks* estudados por esse trabalho, quatro foram escolhidos por serem os que estão em maior evidência em sites de buscas que são Hibernate, Spring, TopLink e DataNucleus. Os *frameworks* estudados têm quase sempre uma comunidade, ou empresa que os desenvolve e os mantêm atualizados. Os representantes dos *frameworks* estudados são:

- Hibernate: desenvolvido pela comunidade JBoss;
- Spring: tem uma comunidade que o representa com este mesmo nome;
- TopLink: desenvolvido pela empresa Oracle;
- DataNucleus: tem uma comunidade que o representa com este mesmo nome.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Realizar uma análise comparativa dos *frameworks* de persistência que são Hibernate, DataNucleus, TopLink e Spring, apresentando suas estruturas, formas de mapeamento, tipos de especificação, apresentando seus pontos fortes e fracos.

1.1.2 Objetivos Específicos

- Levantar na literatura trabalhos similares dos *frameworks* que são foco do desenvolvimento deste trabalho;
- Realizar uma análise detalhada da estrutura, apresentando suas principais características;
- Identificar as especificações que cada *framework* utiliza em sua implementação;
- Estabelecer as características que serão ensaiadas neste trabalho;
- Estabelecer os critérios de comparação entre os *frameworks* estudados;
- Demonstrar através de um exemplo prático o uso de cada *framework* na construção de um aplicativo;

- Apresentar as principais vantagens e desvantagens no uso particular de cada *framework*.

1.2 ESTRUTURA DO TRABALHO

A estrutura deste trabalho é composta de seis capítulos onde este primeiro apresenta o que motivou a utilização de *frameworks* de persistência no desenvolvimento de aplicativos voltados ao ambiente *desktop*, assim como os objetivos estabelecidos para este trabalho. O segundo capítulo explora os conceitos necessários para a utilização de *frameworks* de persistência, apresentando particularmente os *frameworks* Hibernate, DataNucleus, TopLink e Spring.

O terceiro capítulo aprofunda o estudo de cada *framework* objeto de estudo, neste é apresentado às particularidades e limites de cada *framework*, sua estrutura e ainda uma visão rápida da implementação dos métodos de persistência.

O quarto capítulo relata o experimento utilizado como base de pesquisa e demonstra em detalhes a utilização dos *frameworks* no desenvolvimento de um aplicativo, apresentando as formas possíveis de mapeamento objeto-relacional e a forma como é realizado pelos *frameworks* estudados.

O quinto capítulo analisa os resultados obtidos no ensaio com os *frameworks* em estudo, apresenta os critérios estabelecidos para realizar uma análise comparativa dos resultados obtidos.

O sexto capítulo apresenta as conclusões e considerações sobre futuros trabalhos, que poderão complementar e/ou aprofundar o escopo deste.

2 BASE CONCEITUAL

Este capítulo aborda os principais conceitos que formam a base para compreensão dos *frameworks* em geral, inclusive o de persistência. Posteriormente a luz destes conceitos, será descrito os *frameworks* de persistência que serão analisados, testados e comparados neste trabalho.

2.1 *FRAMEWORK*

Várias definições sobre *framework* são descritas na literatura, segundo Gamma et al. (2002, p.42), "um *framework* é um conjunto de classes que cooperam entre si provendo assim um projeto reutilizável para uma específica classe de software".

Os *frameworks* estão se tornando cada vez mais comuns e importantes. Eles são a maneira pela qual os sistemas Orientados a Objetos conseguem a maior reutilização (GAMMA et al, 2002). Segundo o mesmo autor, o *framework* dita a arquitetura de sua aplicação. Ele irá definir a estrutura geral, sua divisão em classes e objetos e em consequência as responsabilidades-chave das classes de objetos, como elas colaboram, e o fluxo de controle.

Um *framework* pode incluir programas de suporte, bibliotecas de código, linguagens de *script* e outros softwares para ajudar a desenvolver e juntar diferentes componentes de um projeto de software.

Os *frameworks* são projetados com o propósito de facilitar o desenvolvimento de *software*, habilitando projetistas e programadores a gastarem mais tempo detalhando as exigências de negócio do *software* do que com detalhes tediosos de baixo nível do sistema.

> Projetar software orientado a objetos é difícil, mas projetar software reutilizável orientado a objetos é mais difícil ainda. É necessário achar objetos pertinentes, fatorá-los em classes no nível correto de granularidade, definir as interfaces das classes e as hierarquias de herança e estabelecer as relações chave entre eles. O projeto deve ser específico para o problema a resolver, mas também genérico o suficiente para atender futuros problemas e requisitos. Também deve evitar o reprojeto, ou pelo menos minimizá-lo (GAMMA et al., 2002, p. 17).

2.1.1 Tipos de *Frameworks*

Classifica-se um *framework* de acordo com duas dimensões: como ele é utilizado e onde é utilizado. No tratamento de como um *framework* pode ser utilizado, será analisado o ponto de como introduzir as particularidades de uma aplicação. Neste sentido Willemann e Ibarra (2007) os classificam em:

- Caixa branca: são baseados na especialização por herança e sobrescrita de métodos, modificando assim as funcionalidades básicas do *framework*.
- Caixa preta: são os *frameworks* focados na composição, devendo utilizar as funcionalidades já presentes no *framework*, ou seja, neste tipo de *framework* as funcionalidades internas não podem ser vistas nem modificadas e devem-se utilizar as interfaces fornecidas pelo *framework*. Neste *framework* as instanciações e composições feitas é o que determinam as particularidades da aplicação.
- Caixa cinza: são *frameworks* híbridos, misturam os dois focos, herança e composição, ou seja, são *frameworks* baseados em herança (caixa branca) com algumas funcionalidades prontas.

A seguir são apresentadas algumas formas de utilização de um *framework*.

- *Frameworks* de suporte ou de integração *midleware:* oferecem serviços de sistema de baixo nível, tais como dispositivos de interface para periféricos (*drivers*) e de acesso a arquivos, sendo normalmente usados para integrar aplicações e componentes distribuídos, como *frameworks* BORBA ORB, DCOM, implementações do padrão ODMG, entre outros (MATOS, 2008).
- *Frameworks* de aplicação ou de infraestrutura: são *frameworks* que cobrem funcionalidades que podem ser aplicadas a diferentes domínios. Ou seja, são independentes do domínio ao qual será endereçado, como por exemplo, os *frameworks* para sistemas operacionais, comunicação, redes e para construção de interfaces (MATOS, 2008).
- *Frameworks* de domínio: capturam conhecimento e especialidade em um domínio de problema particular. Representam um projeto geral de aplicações para domínios específicos, como telecomunicações, manufatura, jogos, controle de produção, multimídia e engenharia financeira (MATOS, 2008).

2.1.2 Vantagens e Desvantagens

De acordo com Willemann e Ibarra(2007), a principal vantagem na utilização de *frameworks* é a redução de custos, sendo que já existe uma estrutura definida e que o desenvolvimento pode concentrar-se em desenvolver as regras específicas do negócio em que o sistema deve atuar. Um *framework* ainda proporciona uma maior reutilização de códigos e a fatoração de problemas em aspectos comuns a várias aplicações, permite também obter sistemas com códigos menos frágeis e com menos defeitos.

Entretanto, caso se decida construir um *framework* deve ter em mente que é uma tarefa complexa, pois o reuso não acontece por acaso, devendo ser adequadamente planejado. Iniciar a construção de um *framework* sem um bom planejamento pode trazer mais prejuízos do que vantagens.

Com certeza, construir uma aplicação e construir um *framework* paralelamente, demora muito mais do que construir uma aplicação isolada. Isso tudo pelo fato de que quando se constrói um *framework,* deve-se planejá-lo de forma que atenda a mais do que uma aplicação, ou seja, atenda a um domínio específico de aplicações e não somente uma. As vantagens de um *framework* só aparecem em longo prazo, na medida em que a estrutura torna-se consistente e de domínio das equipes de desenvolvimento.

2.2 ORIENTAÇÃO A OBJETOS

A orientação a objetos possibilita aos programadores o reaproveitamento de código fonte que seria seu principal benefício. Segundo Booch (2000, p. 456), um objeto é “Uma manifestação concreta de uma abstração; uma entidade com uma fronteira bem-definida e uma identidade que encapsula estado e comportamento; a instância de uma classe”.

Entre outras palavras um objeto pode ser descrito como sendo “algo” no mundo que pode ser concreto ou não, tal como um automóvel que é um objeto concreto, ou uma fatura, que é considerada um objeto não concreto.

Quando se tem um objeto que foi persistido, de acordo com Booch (2000), é todo objeto que existe depois que o processo ou *thread* que o criaram, deixar de existir.

Uma classe para Lima (2009, p.22), é a “definição dos atributos e das ações de um tipo de objeto; ela descreve um conjunto de objetos individuais em qualquer contexto”. É obtida pela classificação de objetos com a mesma estrutura de dados e o mesmo comportamento.

2.3 MODELO ENTIDADE E RELACIONAMENTO

De acordo com Guimarães (2003, p.32), “a modelagem conceitual utilizando o MER (Modelo de Entidade e Relacionamento) consiste em se projetar uma série de diagramas entidade-relacionamento que descrevem os dados da BD (Base de Dados) e os seus inter-relacionamentos”. Entre outras palavras ele é utilizado para fazer a projeção do banco. Quanto mais complexo o banco a ser criado, a sua projeção se torna mais essencial. Na construção de um MER deve-se ter três características a serem analisadas: entidades, atributos dessas entidades, e o relacionamento entre as entidades.

As entidades de acordo com Guimarães (2003), são elementos do mundo real que possui existência própria e cujas características ou propriedades que se desejam registrar. Em um contexto geral, pode-se dizer que entidade tem o significado igual a de um objeto, e por isso na maioria das aplicações para cada objeto será criada quase que automaticamente uma entidade no banco de dados.

Os atributos de acordo com Mello (2011, p.2), significam um “elemento de dado que contém informação que descreve uma entidade”. De acordo com o conceito, atributos são as características relevantes dentro de um determinado grupo de objetos que compõe uma entidade. Os tipos de atributos podem ser (PIRES, 2011):

- Atributo simples: demonstra uma característica comum da entidade;
- Atributo composto: característica que pode ser divida em partes que formam uma hierarquia. Exemplo: Telefone quando também se grava o número de DDD;

- Atributo multivalorado: composto por mais de um valor associado a ele. Exemplo: Telefone quando se deseja armazenar mais um número de telefone para uma entidade específica, como Cliente;
- Atributo determinante: faz o controle de ordenação e distinção entre entidades, normalmente usando uma numeração sequencial de inteiros.

Os relacionamentos entre entidades são a maneira como cada entidade precisa uma da outra. De acordo com Campos (2010, p.17), “os relacionamentos possuem como características obrigatoriedade (se é obrigatório ou não) e cardinalidade (um para um, um para muitos e muitos para muitos)”. Os tipos de relacionamentos estão divididos em (PIRES, 2011):

- 1 : 1 (um para um): quando uma entidade ao se relacionar com outra entidade só pode se associar com um único elemento da outra entidade;
- 1 : M (um para muitos): quando uma entidade se relaciona a um ou mais elementos de outra entidade. Por exemplo: uma pessoa pode ter um ou mais telefones;
- 0 : 1 (nenhum ou no máximo 1 elemento): quando uma entidade pode ou não se associar com outra entidade, mas caso isso ocorra, ela só pode se relacionar a um único elemento da segunda entidade. Por exemplo, se no mundo inteiro as pessoas só pudessem não ter ou ter somente um telefone;
- 0 : M (nenhum ou muitos): quando o relacionamento entre uma entidade pode não existir, ou pode se relacionar com apenas um ou com vários elementos de uma segunda entidade. Por exemplo: um funcionário pode não ter dependentes, ou poderia ter somente um ou vários dependentes.
- M : M (muitos para muitos):quando o relacionamento entre uma entidade e outras várias vezes de qualquer lado. Por exemplo: uma pessoa pode comprar um ou mais produtos e um produto pode ser adquirido por uma ou mais pessoas.

A letra “M” usada tem o significado de dizer a representatividade de muitos quando esta associada a uma cardinalidade de 1, 0 ou M.

2.4 BANCO DE DADOS

De acordo com Korth e Silberschatz (1995), banco de dados é um conjunto de dados integrados que tem por objetivo atender a uma comunidade específica.

Pode considerar sua utilização sempre que é necessário guardar alguma informação. Nas palavras de Elmasri (2005, p.3), a “construção de um banco de dados é o processo de armazenar os dados em alguma mídia apropriada controlada pelo SGBD” (Sistema Gerenciador de Banco de Dados). O que torna necessário a utilização do SGBD para controle do acesso.

2.4.1 Tipos de Bancos

Nos anos 60 surgiram os modelos de bancos de dados hierárquicos, os quais registravam os dados numa estrutura de árvore. Nos anos 70 surgiram os modelos de banco de dados em rede, que armazenavam os dados em registro. Também apareceram os modelos de banco de dados relacional, cuja estrutura era de tabelas/relações. Mas com a vinda da linguagem orientada a objetos começou a se criar bancos com suporte a orientação objetos, para ter reaproveitamento de sua estrutura que seria um dos principais conceitos da orientação a objetos.

Para Galante (2011, p.6), um banco de dados orientado a objetos é “um banco em que cada informação é armazenada na forma de objetos, e só pode ser manipuladas através de métodos definidos pela classe que esteja o objeto”. Então começa a se entender que um banco de dados OO pode trazer todo o benefício da orientação a objetos.

De acordo com Lima (2008), a utilização de um banco de dados relacional apresenta um conjunto de dificuldades para traduzí-lo em um modelo OO.Tendo em vista essa problemática, surgiram os sistemas objeto-relacionais capazes de utilizar um banco relacional e uma programação de aplicativo criado com a orientação a objetos, como exemplo desses sistemas pode citar os *frameworks* que fazem o mapeamento objeto-relacional.

2.5 SISTEMA GERENCIADOR DE BANCO DE DADOS

De acordo com Elmasri (2005, p.3) o conceito de SGBD “é uma coleção de programas que permite aos usuários criar e manter um banco de dados”. Pode-se dizer então que é um programa com o objetivo de facilitar a construção e manutenção do banco de dados.

Quando se constrói um banco de dados através de um SGBD temos diferentes funcionalidades, as quais se pode destacar a utilização da linguagem de manipulação de dados geral que seria o SQL e *triggers.*

De acordo com Guimarães (2003, p.199), *triggers "*São procedimentos remotos especiais escritos pelo usuário (ou pelo administrador), mas que são invocados automaticamente pelo SGBD quando são efetuadas certas operações de modificação".

Para Guimarães (2003, p.99), a linguagem SQL "é uma linguagem de definição e de manipulação de dados relacionais, desenvolvida nos laboratórios da IBM nos anos 70 e hoje padronizada pelos comitês ISO/ANSI". É uma linguagem usada em qualquer tipo de banco que tem suporte a todas as ações necessárias para manipulá-lo.

De acordo com Bauer (2005, p. 5), o termo persistência significa: "normalmente estamos falando sobre armazenar dados em um banco de dados relacional usando SQL".

3 ESTUDO SOBRE OS *FRAMEWORKS*

O paradigma de orientação a objetos ocasionou uma mudança radical na estruturação e organização da informação. Entretanto, os bancos de dados mais utilizados continuaram sendo relacionais. Devido a isto, é comum a adaptação dos modelos de objetos na tentativa de adequá-los com o modelo relacional. Além disso, é notório o esforço aplicado no processo de persistência manual dos objetos nos bancos de dados – onde os desenvolvedores precisam dominar a linguagem SQL e utilizá-la para realizar acessos ao banco de dados.

Como consequência ocorre uma redução considerável na qualidade do produto final, construção de uma modelagem "orientada a objetos" inconsistente e a um desperdício considerável de tempo no desenvolvimento manual das classes de persistência.

Para permitir um processo de mapeamento entre sistemas orientados a objetos e bases de dados relacionais, foram propostas diversas ideias que conduziriam para o conceito de Camada de Persistência. Este estudo visa analisar e comparar quatro *frameworks* de persistência existentes, para identificar suas principais características e devidas restrições por meio de um teste.

3.1 MAPEAMENTO OBJETO-RELACIONAL

Existem diferentes tipos de mapeamentos possíveis que os *frameworks* fazem para as tabelas no banco de dados, dentre os quais se podem citar:

- Quando se usa linguagem XML (*Extensible Markup Language*), deve-se tomar cuidado em função de que cada *framework* possui uma notação diferenciada na geração do arquivo XML.
- Usando direto na classe POJO (*Plain Old Java Objects*) que seria classes simples com declaração de atributos representativos a uma entidade e declaração de *setters* (utilizado para fazer uma atribuição em um atributo da classe) e *getters* (utilizado para resgatar valores do atributo na classe), que é chamado mapeamento via anotações.

Dentre os *frameworks* estudados sempre haverá divergências em seus mapeamentos, mas quando se segue um padrão de persistência, ou seja, uma

especificação para persistir seus dados então se encontra um padrão que poderá ser usado nos mais diferentes *frameworks*, tendo raras modificações.

3.2 ESPECIFICAÇÕES DE PERSISTÊNCIA

De acordo com Campos (2010), são regras feitas para serem utilizadas no desenvolvimento de aplicações ORM em Java, essas especificações são divididas em JDO (*Java Data Object*), que é mantida pela Apache e JPA (*Java Persistence* API), que é mantida pela Sun. Devido a seus desenvolvedores serem diferentes o código dos métodos utilizados na persistência ficou bem diferenciado, assim para explicar devidamente as maneiras como pode ser feito as especificações estão mais detalhadas a seguir:

3.2.1 Especificação JDO

De acordo com Campos (2010, p.27), “o controle da persistência dos objetos é feito utilizando a *PersistenceManagerFactory* e a *PersistenceManager*, ambas são encontradas no pacote javax.jdo”.

- *Persistence Manager Factory*

De acordo com Datanucleus (2011) é tipicamente usado para criação do banco, ela provê o acesso a *PersistenceManagers*com objetos a serem persistidos, para ser utilizado com a especificação JDO.

- *Persistence Manager*

De acordo com Campos (2010, p.27), ela “fornece métodos para gerenciar a persistência de objetos, alterando o ciclo de vida deles”. Para Datanucleus (2011), a criação da persistência é realizada da seguinte maneira: depois de instanciado um *Persistence Manager Factory*, isso lhe dá a possibilidade da criação de *PersistenceManager* de acordo com Quadro 1, onde esses são usados para chamadas dos métodos de persistência.

Declaração
PersistenceManager pm = pmf.getPersistenceManager();

Quadro 1 - Criação de PersistenceManager
Fonte - Autoria própria

Os métodos usados para persistir dados usando esta especificação são demonstrados no Quadro 2.

Operação	Método
Persistir ou atualizar um objeto	pm.makePersistent(obj);
Encontrando um objeto	Object id = pm.getObjectId(obj); ou Object obj = pm.getObjectById(id);
Excluir um objeto	pm.deletePersistent(obj);
Para atualizar no banco	em.refresh(obj);

Quadro 2 - Métodos de persistência com especificação JDO
Fonte - Datanucleus (2011)

3.2.2 Especificação JPA

De acordo com Datanucleus (2011), cria-se uma unidade de persistência por ser um componente requerido para comunicação com o banco de dados.Para realizar a persistência de dados utilizando essa especificação utilizam-se duas classes que de acordo com Campos (2010), fazem o controle da persistência, são elas: *Entity Manager Factory* e *Entity Manager*.

- *Entity Manager Factory*

Para Campos (2010), é a *EntityManagerFactory* que permite a utilização do banco de dados por via de*EntityManager*, para outros bancos de dados é atribuída uma instância diferente dela.

- *Entity Manager*

De acordo com Datanucleus (2011),*EntityManager* faz as operações de persistência com o banco de dados para seus objetos. Para obter um *EntityManager* pode ser feito conforme apresentado na Quadro 3:

Declaração
EntityManager em = emf.createEntityManager();

Quadro 3 - Obtendo EntityManager
Fonte - Autoria própria

Para persistir os dados por meio desta especificação podem-se usar os métodos que estão no Quadro 4.

Operação	Método
Persistir objetos	em.persist(obj);
Encontrando um objeto	Object obj = em.find(cls, id);
Atualizar objetos	Object updatedObj = em.merge(obj);
Atualizar no banco	em.refresh(obj);
Excluindo um objeto	Object obj = em.find(cls, id); e após em.remove(obj);

Quadro 4 - Métodos de persistência especificação JPA
Fonte – Datanucleus (2011)

Quando for necessário excluir um objeto dentro do banco de dados os passos seguintes devem ser adotados:

- Primeiramente deve-se achá-lo no banco através do comando: *Object* obj = em.*find*(cls, id);
- E em seguida com o objeto encontrado, deve-se executar o comando para excluí-lo: em.*remove*(obj);

3.3 *FRAMEWORKS* ESTUDADOS

Para o desenvolvimento deste trabalho procurou-se analisar quais *frameworks* estavam em evidência e a partir disso discutir quais seriam utilizados para realizar o estudo comparativo de *frameworks* de persistência. Foram escolhidos os seguintes *frameworks*: Hibernate, Spring, TopLink e DataNucleus.

3.3.1 Hibernate

Segundo Bauer e King (2005), o estudo do *framework* Hibernate “visa ser uma solução completa para o problema de gerenciamento de dados persistentes em Java”, portanto o Hibernate é um *framework* que será analisado neste trabalho.

De acordo com Pereira (2009, p. 36),

> Usando o Hibernate o desenvolvedor se livra de desenvolver muito código de acesso a banco de dados e de SQL que ele escreveria se não estivesse usando essa ferramenta, acelerando a velocidade do seu desenvolvimento de uma forma fantástica.

A utilização desse *framework* facilita o armazenamento e recuperação de objetos em Java com seu mapeamento objeto-relacional. Também trabalha com o modelo de objetos POJO que segundo Shaefer e Isaia (2006), são objetos Java, sem nenhuma lógica dentro de seu código sendo assim eles têm apenas atributos e seus respectivos métodos de acesso (*getters* e *setters*).

Ainda segundo Pereira (2009, p. 44),

> Hibernate não impõe restrições ao modelo de objetos baseados em POJOS (*Plain Old Java Object*). A Aplicação pode ser implementada sobre qualquer camada de persistência. Com o Hibernate pode-se mapear o modelo a um banco sem alterar sua estrutura.

Esse *framework* além de ser *open source* o que possibilita aos desenvolvedores modificá-lo para atender às suas necessidades particulares, também tem uma comunidade chamada JBoss para auxílio aos novos integrantes dessa tecnologia de desenvolvimento e também onde pode ser encontrada a documentação referente ao mesmo.

Quanto ao suporte em banco de dados, pode-se dizer que o Hibernate tem um suporte amplo e quando ele não consegue dar suporte a algum tipo de banco, é possível se criar um *driver* que permita a compatibilidade.

3.3.1.1 Estrutura do *framework*

Segundo Kraemer e Vogt (2005), o Hibernate utiliza um número de APIs dependendo da complexidade do projeto. Segundo a documentação do King, et al. (2011), a arquitetura está dividida em:

- *SessionFactory* (org.hibernate.SessionFactory)

Para Fernandes e Lima (2007, p.10), “é aquele que mantém o mapeamento objeto relacional em memória. Permite a criação de objetos *Session*, a partir dos quais os dados são acessados”.

- *Session* (org.hibernate.Session)

De acordo com Bauer (2005), ela é a principal interface usada para fazer aplicativos com Hibernate. Pode-se considerar que uma sessão seja equivalente a algo entre conexão e transação.

- Objetos persistentes e coleções

Segundo Fernandes e Lima (2007, p.10), “é utilizada para representar uma unidade indivisível de uma operação de manipulação de dados”.

- Objetos e coleções desanexados e transientes

De acordo com Hibernate (2011), são consideradas instanciações de classes utilizadas para persistência que ainda não foram associados a uma *Session*.

- *Transaction* (org.hibernate.Transaction)

Para Bauer (2005), é considerada como opcional sua utilização dentro de projetos. A sua função seria abstrair o código da transação. Ela permite que o aplicativo faça o controle de limites envolvidos com a transação possuindo uma API consistente.

- *ConnectionProvider* (org.hibernate.connection.ConnectionProvider)

Segundo Hibernate (2011), é considerada de uso opcional para fazer o aplicativo, sua função seria abstrair a aplicação do *Datasource* ou *DriverManager* adjacentes.

- *TransactionFactory*(org.hibernate.TransactionFactory)

De acordo com o Hibernate (2011), é opcional do ponto de vista de se ter ou não na aplicação. Ele faz uma fábrica de instâncias de *Transaction*.

- *Extension Interfaces*

Ainda de acordo com Hibernate (2011), tem a oferecer vários tipos de versões de interfaces que podem ser estendidas para que o programador possa customizar a sua persistência. A Figura 1 mostra a arquitetura desse *framework*, os papéis das interfaces mais importantes, dentro das camadas de persistência e de negócios.

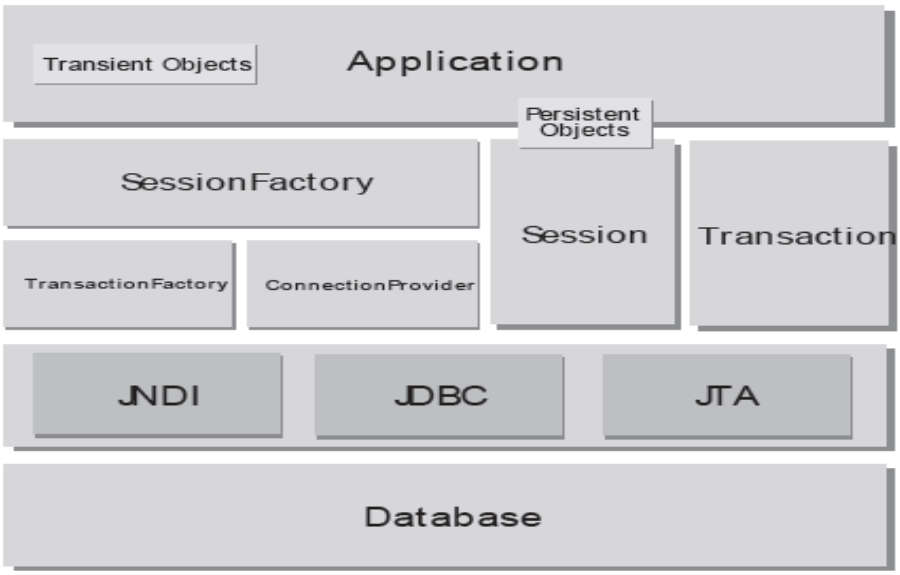

Figura 1 - Arquitetura *framework* Hibernate
Fonte - Hibernate (2011)

3.3.1.2 Detalhes da implementação

Para o desenvolvimento de novos aplicativos com este *framework* é necessário o uso das seguintes classes:

- ***Configuration*:** De acordo com Bauer e King (2005), um objeto *configuration* pode ser usado para inicialização do Hibernate. O aplicativo instancia o *configuration* para dizer onde estão os mapeamentos e propriedades do Hibernate e criar a *SessionFactory*.
- ***SessionFactory*:** Segundo Bauer e King (2005, p.52), "ela foi criada para ser compartilhada entre muitas *threads* do aplicativo". Ainda de acordo com Bauer e King (2005) ter um *SessionFactory* traz a possibilidade de acessar múltiplos bancos com o *framework*.
- ***Session*:** Para Bauer e King (2005, p.52), "algumas vezes chamamos a *session* de gerenciador de persistência porque ela é também a interface para operações relacionadas com a persistência".
- ***Transaction*:** De acordo com Bauer e King (2005, p.53), permite "ao aplicativo controlar limites de transação por meio de uma API consistente. Isso ajuda a manter os aplicativos do Hibernate portáveis".
- **Linguagem de Manipulação Própria:** Segundo Bauer e King (2005, p.183), a linguagem de manipulação utilizada pelo *framework* Hibernate é "a Linguagem de

Consulta do Hibernate (HQL) é um dialeto orientado para objetos da linguagem de consulta relacional familiar SQL". Portando pelo fato da HQL ser familiar à SQL, ajuda muito os desenvolvedores a se adaptarem a realização de consultas com essa nova ferramenta.

Hibernate (2011, p.275), diz que quando "comparado com o SQL o HQL é totalmente orientado a objetos, e compreende noções de herança, polimorfismo e associações". Com base nesta afirmativa, como a orientação a objetos é o reaproveitamento de códigos facilita e muito o trabalho do programador.

3.3.1.3 Persistência com hibernate

A persistência de dados através deste *framework* pode ser feito de duas maneiras que seriam com API própria do *framework* ou utilizando a especificação que ele tem suporte que seria a JPA.Para fazer a persistência de dados de um aplicativo com o *framework* e utilizando a especificação JPA basta seguir a seção 3.2.2 que conterá tudo o que precise saber para isso.

3.3.1.3.1 Persistência hibernate utilizando API

Para realizar a conexão com o banco de dados e facilitar sua utilização, dividiu-se em dois métodos distintos, um deve ser invocado para realizar a conexão com o banco de dados desejado e o segundo método para desconectá-lo. Esses métodos são apresentados a seguir:

- **Conectar**: onde foi estipulado o que seria necessário para fazer a efetiva conexão com o banco de dados e, possibilitar com que o *session* use isso para persistir os dados. Um exemplo é mostrado na Quadro 5.

Método

public void conectar(){
SessionFactory factory = new Configuration().
configure("VO/hibernate.cfg.xml").
buildSessionFactory();
session = factory.openSession();
t = session.beginTransaction();
}

Quadro 5 - Método de conexão
Fonte - Autoria própria

- **Desconectar**: onde se estipulou os comandos necessários para realizar a desconexão do programa com o banco de dados e, atualização desses dados no banco, para que se o usuário do programa volte àquela página os dados já estejam atualizados.

Método

public void desconectar(){
t.commit();
session.flush();
session.close();
}

Quadro 6 - Método de desconexão
Fonte - Autoria própria

Para realizar a conexão tem dois tipos de arquivos onde se definem as propriedades relacionadas ao banco:

- Arquivo de configuração Hibernate, que é chamado de (hibernate.cfg.xml);
- Arquivo de propriedades com as configurações que serão utilizadas, neste caso usado com anotações que são feitas dentro das classes de POJO do projeto. O que muda é no método de conexão que ao invés de:

Definição
SessionFactory factory = new Configuration(). configure("VO/hibernate.cfg.xml"). buildSessionFactory();

Quadro 7 - Definindo propriedade configure
Fonte - Autoria própria

Ele usa quase a mesma linha, porém, teria ao invés de “VO/hibernate.cfg.xml” teria um novo pacote e nome de arquivo de propriedades para esse lugar.

Quando o método *configure* tem algum parâmetro sendo passado quer dizer que seu arquivo de propriedades está fora do pacote padrão e, portanto esse

parâmetro deverá ser passado para o pacote em questão onde o arquivo de propriedades se encontra. Os métodos para persistir são apresentados no Quadro 8:

Operação	Método
Persistir objeto	session.save(obj);
Atualizar objeto	session.update(obj);
Excluir objeto	session.delete(obj);
Buscar objeto	User u=(User) session.createQuery(“from User u where u.name=:username”).setString(“userName”, username).uniqueResult();

Quadro 8 - Métodos de Persistência API Hibernate
Fonte - Hibernate (2011)

Quando precisar buscar um objeto o comando pode ser assim explicado:

- No comando para buscar um objeto (*User*) significa o *cast* do resultado obtido com para ser atribuído ao objeto *User*.
- Em *createQuery* é possível criar um comando HQL embutido, o qual é semelhante aos comandos SQL só que ao invés de usar os nomes das tabelas se usa o nome das classes que as correspondem.
- Em *setString* está sendo definido o parâmetro do comando de seleção feito. Então no primeiro *userName* que está entre aspas significa o parâmetro que está dentro do comando de seleção, e o segundo *userName* é a *String* sendo fornecida para ser usada como parâmetro dentro do arquivo de seleção.

3.3.2 Spring

O Spring é um *framework* de persistência *open-source*, simples e de baixo acoplamento, criado por Rod Johnson. Segundo Walls e Breidenbach (2006) ele foi criado para simplificar a complexidade de desenvolvimento de aplicativos *enterprise*. O Spring torna possível usar objetos POJO (*Plain Old Java Object*) para alcançar coisas que previamente só eram possíveis com EJBs (*Java Beans Enterprise*).

De acordo com Santos (2007, p.30), o *framework* de persistência “provê diversas soluções para persistência de dados, programação orientada a aspectos, entre outras”. De maneira geral o Spring *framework* é mais voltado à área de desenvolvimento web.

3.3.2.1 Suporte

O Spring vem com uma família de *frameworks* de acesso a dados, que se integra com uma variedade de tecnologias. Para Adamatti (2006, p.25),

> O *Framework* Spring suporta a integração de classes remotamente usando várias tecnologias. A utilizada atualmente pelo Fumigant é a RMI. Todas as chamadas de RMI são abstraídas pelo *Spring*, sendo desnecessários conhecimentos aprofundados na tecnologia para uso da mesma.

3.3.2.2 Estrutura do *framework*

A Figura 2 mostra a arquitetura do Spring com todos os seus módulos. Podemos ainda ver nessa mesma figura que o *framework* tem um módulo que se chama *Web* como o próprio nome diz ele é voltado para programação *Web.*

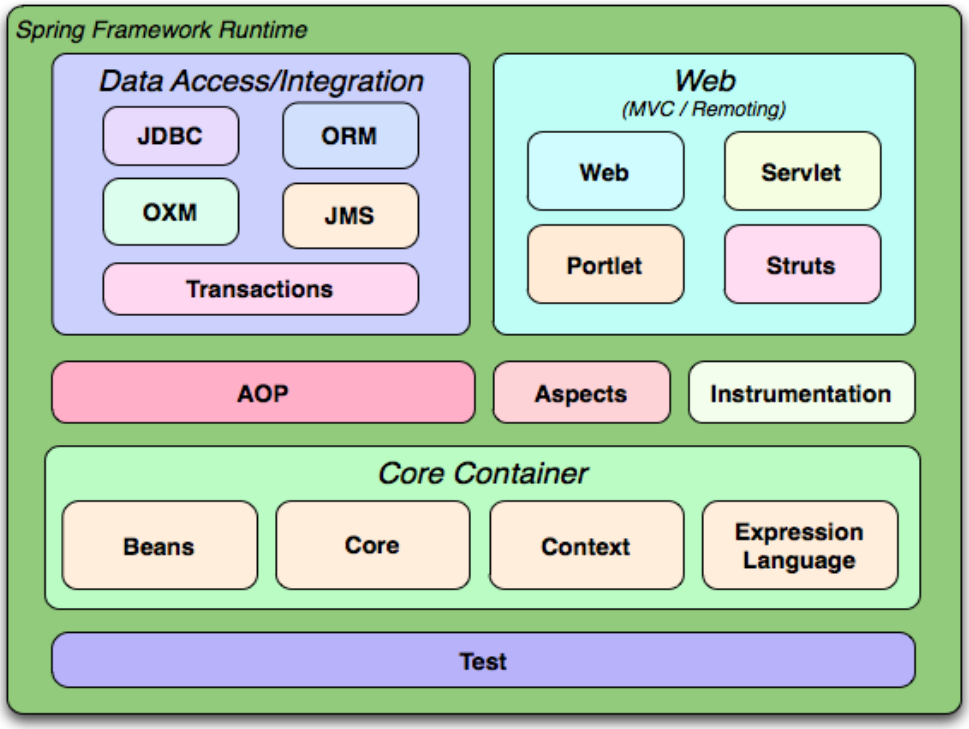

Figura 2 - Arquitetura Spring
Fonte – Spring (2011)

De acordo com SPRING (2011), a estrutura deste *framework* pode ser assim descrito:

- AOP (*Aspect Oriented Programming*):Este módulo fornece a possibilidade dodesenvolvimento de programasorientados a aspectos.

- *Aspects*: módulo que provê a integração com AspectJ.
- *Instrumentation*: este módulo provê a classe de instrumentação e a possibilidade do desenvolvimento de *classloader* para ser certamente usado em servidores de aplicativos.
- *Data Access/Integration*, ele está dividido em:
    - JDBC: módulo que provê uma abstração JDBC, que remove a necessidade de se fazer o tedioso código JDBC, e passa a olhar especificamente a regra de negócios.
    - ORM: este módulo provê a integração para API de mapeamento objeto-relacional, incluindo JPA, Hibernate, e iBatis. Usando o pacote ORM você pode usar todos esses *frameworks* junto com o que o Spring tem a oferecer.
    - OXM: é um módulo que provê a abstração que suporta implementações objeto/XML para JAXB, Castor, XML, *Bean*s, JiBX e *XStream*.
    - O Serviço de Mensagens Java (JMS): é um módulo que contêm recursos para produzir e ver mensagens.
    - *Transaction*: módulo que suporta o gerenciamento de transações de forma programática e declarativa para as classes queprogramar com interfaces especiais e para todos os seus POJOs (*plain old Java objects*).
- *Web*: está dividido nos seguintes módulos
    - *Web*: ele oferece recursos básicos de integração orientados à web, tais como a possibilidade de fazer *upload* de arquivos através de *multipart* e a inicialização do *container*IoC.
    - *Servlet*: é o módulo que contém o desenvolvimento do MVC para aplicações web. O Spring MVC provê uma clara separação entre modelos de códigos e janelas da web, e integra-se com todos os outros recursos do *framework*.
    - *Portlet*: fornece o desenvolvimento para uso em um ambiente de *portlet* e se espelha na funcionalidade do módulo de *Servlet*.
    - *Struts*: é o módulo que contém as classes de suporte para integração do *Struts* clássico dentro de uma aplicação Spring. Esse suporte tornou-se

obsoleto a partir do Spring 3.0 onde se deve considerar a migração se você usar o *Struts* 2.0 com esse *framework*.

- *Core Container*: está dividido nos seguintes módulos
    - *Core* e *Beans*: são módulos que provêem a parte fundamental do *framework*, incluindo a injeção de dependências para objetos.
    - *Context*: éo módulo que tem base no Núcleo e no módulo*Bean*s: ele é um meio para acessar objetos de uma maneira que é semelhante a um registro JNDI.O módulo *Context*herda suas características a partir do módulo *Bean*s e adiciona suporte para a internacionalização (usando, por exemplo, pacotes de recursos), evento-propagação, o recurso de carregamento, ea criação de contextos transparente, porexemplo, um servlet container. O módulo *Context* também suporta recursos Java EE, como EJB, JMX, e*remoting* básica. A interface *ApplicationContext* é o ponto foco do módulo *Context*.
    - Linguagem de expressão: é o módulo que fornece uma poderosa linguagem de expressão para consultar emanipular um objeto gráfico em tempo de execução. Ele também suporta a projeção lista e seleção, bem como agregações de lista comum.

De acordo Barreto e Lucena (2005, p.6) o Spring é,

> Dividido em módulos que podem ser usados separadamente ou em conjunto conforme a necessidade do projeto. O módulo de mapeamento objeto-relacional integra-se ao Hibernate, controlando a abertura e o fechamento de sessões.

Tendo-se então uma integração ao Hibernate fica mais fácil para alguns programadores começarem a utilizar o *Spring* se antes já utilizavam o Hibernate.

Sendo uma parte da estrutura deste *framework* a parte de validação de acordo com Santos (2007), tem uma interface que se chama *Validator*, usada para validação de objetos. Dentro dele também se encontra a classe *Errors* utilizada para o recebimento de mensagens de falha durante a validação feita por *Validador*.

3.3.2.3 Módulo de mapeamento objeto relacional

De acordo com Walls e Breidenbach (2006, p.11),

> Para aqueles que preferem usar uma ferramenta que faz um mapeamento objeto/relacional (ORM) diretamente sobre o JDBC, o Spring oferece o módulo ORM. O Spring não tenta implementar sua própria solução ORM, mas provê ganchos a vários *frameworks* populares de ORM, incluindo Hibernate, JDO e iBATIS SQL Maps.

O gerenciamento transacional do Spring dá suporte a cada um destes *frameworks*, como também o JDBC.

3.3.2.4 Detalhes da implementação

Para executar qualquer operação JDBC num banco de dados, você precisa de um *connection*. Segundo Walls e Breidenbach (2006, p.137) no “Spring, objetos *Connection* são obtidos através de um *DataSource*”.

Frequentemente as aplicações Spring são executadas dentro de um servidor de aplicação J2EE ou até mesmo, num servidor web como o Tomcat. Uma coisa que esses servidores podem prover é um *DataSource* por JNDI (*Java Naming and Directory Interface*) que segundo Oracle (2011) “é a parte da plataforma Java, fornecendo aplicações baseadas na tecnologia Java com uma interface unificada para nomeação múltipla e serviços de diretório”.

Com o Spring, isto é tratado como qualquer outro objeto de serviço, dentro da aplicação – como um *bean* do Spring. Neste caso, usa-se um*JndiObjectFactoryBean*. Tudo o que se precisa fazer é configurá-lo com o nome do JNDI do *DataSource*. Segue um exemplo no Quadro 9.

Configuração
<bean id = "dataSource" Class = "org.springframework.jndi.jndiObjectFactoryBean">   <property name = "jndiName>   <value>java:comp/env/jdbc/myDatasource</value>   </property> </bean>

Quadro 9 - DataSource do JNDI
Fonte - Walls e Breidenbach (2006)

3.3.2.5 Usando data source na fase de teste

De acordo com Walls e Breidenbach (2006) o Spring vem com uma implementação simplificada de *DataSource*, específica para isto: *DriverManagerDataSource*. Esta classe pode ser facilmente configurada e usada com um teste unitário ou um conjunto de testes unitários. Agora se tem um

DataSource para usar ao testar o código de acesso aos dados, como pode ser visto na Quadro 10.

Definições

DriverManagerDataSource ds = new DriverManagerDataSource();
ds.setDriverClassName(driver);
ds.setUrl(url);
ds.setUserName(usuario);
ds.setPassword(senha);

Quadro 10 - Classe DriverManagerDataSource
Fonte - Walls e Breidenbach (2006)

3.3.2.6 Classes necessárias

De acordo com Walls e Breidenbach (2006), o Spring separa as partes fixas e variantes do processo de acesso a dados, em duas classes distintas: *Templates* e *Callbacks*.

- *Templates*: controlam as partes fixas do processo como, transações, exceções e gerencia os recursos.
- *Callbacks*: fica nos detalhes da implementação como, criação de *statements*, parâmetros de ligação e ordenação de conjuntos de resultados.

Em cima do *design* t*emplate-callback*, cada *framework* provê uma classe de suporte para ser estendida por suas próprias classes de acesso a dados. A relação entre sua classe, à classe de suporte e a classe *template* é ilustrada na Figura 3.

As classes de suporte possuem uma propriedade para conter uma classe *template*, assim terá que criar esta propriedade para cada uma de suas classes DAO. Além disso, cada classe de suporte lhe permite adquirir acesso direto a qualquer classe que é usada para se comunicar com o banco de dados.

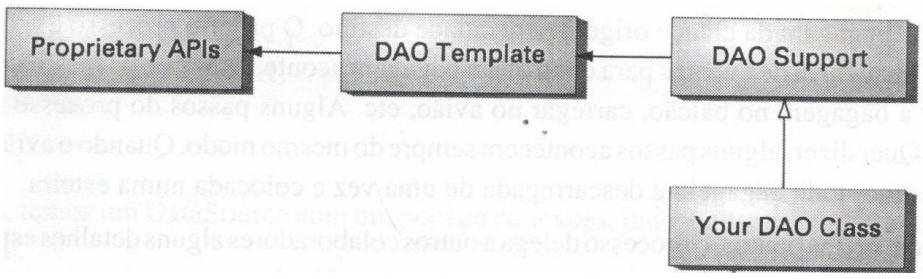

Figura 3 – Relação: API de Persistência, template, suporte DAO e seu DAO.
Fonte - Walls e Breidenbach (2006)

A relação de classes necessárias para persistir o Spring:

- *DriverManagerDataSource*: É voltado para conexão.
- *RowMapper*: Responsável por mapear uma linha do *ResultSet* em um objeto.
- *RowMappersResultReader*: Busca múltiplos objetos em uma *query*.
- *CallableStatmentCallback*: Utilizada em *procedures*.
- *MappingSqlQuery*: Usada para modelar uma pesquisa com um objeto.

3.3.2.7 Persistência utilizando *framework*

De acordo com SPRING (2011), a persistência de dados pode ser feito por meio do *JdbcTemplate*, e também com ajuda de outros *frameworks* através das especificações JPA e JDO. Para persistir seus dados através das especificações é só seguir o que já foi disposto nos tópicos 3.2.1 para JDO e 3.2.2 para JPA.

Ainda de acordo com SPRING (2011), o *framework* pode ser integrado a API do Hibernate que se tiver realmente integrado, você poderá utilizar métodos da API do Hibernate para persistência de dados dentro de seu aplicativo. A persistência que é própria do *framework*sem a utilização de nenhuma ajuda de outro pode ser feita através do *JdbcTemplate*.

3.3.2.7.1 Métodos de persistência

Os métodos que podem ser feitos na persistência de dados com o *JdbcTemplate,* utilizados através de uma sintaxe igual para todos variando somente no comando SQL passado. Quando está sendo feito comando de alteração de dados a sintaxe passada para executar o comando SQL é feita conforme ilustrado no Quadro 11:

Método
this.jdbcTemplate.update (sql, new Object[]{a.getCodalimento(), a.getTipoalimento(), a.getCusto()});

Quadro 11 - Comandos para métodos de alteração
Fonte - Spring (2011)

Pode-se ver que na imagem um dos parâmetros é “sql”, esta é passada uma frase do tipo Java criado em String com o comando SQL. Os outros parâmetros passados são os atributos passados que serão usados no comando. Quando é preciso fazer comandos de seleção ainda precisa-se fazer uma atribuição do que foi

pego pelo método de seleção para o objeto. Para realizar a seleção de dados pode ser feito através do método:

Método

```
public Alimento procuraNomeExata(String tipo) {
    String   sql = "select * from alimento where tipoalimento = '" + tipo + "'";
    Alimento a;
    try{
        a = (Alimento) jdbcTemplate.queryForObject(sql,new RowMapper<Alimento>(){
        @Override
            public Alimento mapRow(ResultSet rs, int rowNum) throws SQLException{
                Alimento a = new Alimento();
                a.setCodalimento(rs.getInt("codalimento"));
                a.setCusto(rs.getString("custo"));
                a.setTipoalimento(rs.getString("tipoalimento"));
                return a;
            }
        });
    }catch(EmptyResultDataAccessException e){
        a = null;
    }
    return a;
}
```

Quadro 12 - Comando de seleção específico
Fonte - Autoria própria

O comando de seleção que foi feito é para um objeto específico onde pode ser utilizado somente quando se tem uma procura por código de chave primária ou por um nome exato quando se sabe que este não repete. Já para fazer comandos de seleção que sabe que o retorno é mais de um objeto, então o método que já foi apresentado precisa sofrer algumas alterações.

O que muda de um para o outro é a primeira linha que, onde antes tinha a atribuição a um objeto agora teremos a mesma linha sendo atribuída a um vetor de objetos, conforme pode ser visto no Quadro 13:

Método
List<Alimento> obj = (List<Alimento>) jdbcTemplate.query(sql,new RowMapper<Alimento>()

Quadro 13 - Mudança no comando de seleção
Fonte - Autoria própria

3.3.3 TopLink

Para Trancoso e Pereira (2009, p.35), “TopLink permite armazenamento de objetos Java em bases de dados relacionais ou para a conversão de objetos Java em documentos no formato XML”.Segundo Barros e Cortes (2009, p.5),

> O TopLink tem como características um *framework* de consultas muito rico, que suporta grande parte dos *frameworks* de expressões, como por exemplo, a SQL. Além disso, possui um *cache* para a identificação de objetos de entidades, suporte as metalinguagens Anotações e mapeamento XML.

De acordo com a Oracle (2011),

> É um *framework* de persistência que permite que aplicações Java para acessar bancos de dados relacionais e as fontes de dados não relacionais. O TopLink mapeia objetos e *Enterprise Java Beans* (EJBs) ao banco de dados de forma não-intrusiva e permite que o desenvolvedor trabalhar no nível do objeto.

Tem suporte a bancos de dados relacionais, MySQL, PostgreSQL dentre outros.

3.3.3.1 Estrutura do *framework*

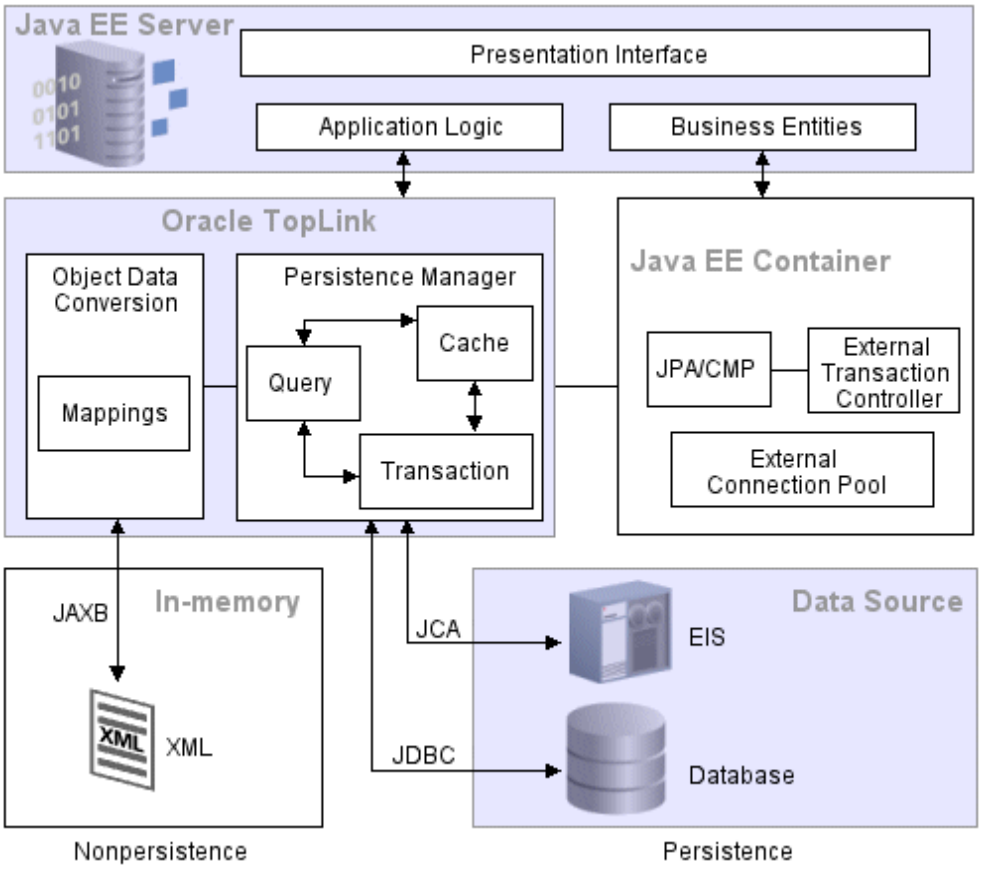

Figura 4 - Arquitetura TopLink runtime
Fonte - TOPLINK (2012)

A explicação a arquitetura apresentada pela Figura 4 de acordo com Toplink (2012), mostra como o *framework* se encaixa em uma arquitetura típica Java EE composta de uma aplicação servidora, TopLink, um opcional Java EE *Container*, e uma fonte de dados.

O *framework* é composto de uma sessão de *front-end* e um acesso a dados *back-end*. Por meio da sessão os componentes de acesso podem fazer uso de dados de mapeamentos (projeto XML de metadados), quadros de consulta, *Cache*, e componentes de Transação.

3.3.3.2 Detalhes da implementação

Segundo Dallacqua (2009), o arquivo *persistence*.xml contém as configurações para acesso à base de dados. As propriedades referentes à implementação desse *framework* são definidas assim:

- *toplink.jdbc.user* - nome do usuário que acessa o banco de dados;
- *toplink.jdbc.password* - senha do usuário estipulado;
- *toplink.jdbc.url* - URL aonde está o banco junto com a estipulação de qual jdbc foi usado;
- *toplink.jdbc.driver* - nome do *driver* para acessar o banco;
- *toplink.logging.level* - foi definido que tipo de *log* as consultas e acessos ao banco de dados devem gerar.

Durante o desenvolvimento de um aplicativo com este *framework,* se necessitar de uma linguagem de manipulação ele pode usar a linguagem SQL. Pois de acordo com Dallacqua (2009, p. 7), para o TopLink “os desenvolvedores podem definir *queries* usando qualquer expressão, como EJB QL, SQL e *Stored Procedures*”.

3.3.3.3 Classes necessárias

Este *framework* possibilita ser utilizado através da especificação JPA. Onde as classes necessárias para desenvolver um novo aplicativo são feitas de um conjunto de bibliotecas do TopLink voltadas a especificação JPA. Então para saber quais classes serão usadas é só seguir o que ficou disposto no tópico de especificação JPA.

3.3.3.4 Persistência utilizando *framework*

Como este *framework* oferece suporte à especificação JPA então poderá usá-lo para persistir seus dados seguindo o tópico 3.2.2.

3.3.4 DataNucleus

Segundo Campos (2010, p.30), “é um *framework* de persistência objeto-relacional que anteriormente era conhecido como JPOX, é desenvolvido pela comunidade de software livre e disponibilizado sem custos para ser utilizado”.

3.3.4.1 SGBD suportado

De acordo com Datanucleus (2011), esse *framework* dá suporte a vários bancos de dados e especificações de persistência.

- Bancos Relacionais (RDBMS): Oracle, MySQL, Postgress, entre outros.
- Banco Orientado a objeto (OODBMS): DB4o.
- Outros: Google Big Table, HBase, Arquivos XML, Excel, planilhas OpenDocument, JDO (Java Data *Objects*) e JPA (Java *Persistence* API).

3.3.4.2 Estrutura do *framework*

Na arquitetura do Datanucleus (2011) como pode ser visto na Figura 5, podem-se identificar os módulos que seriam os mais interessantes à persistência:

- *PersistenceEngine*: Pode ser baseada nos padrões JDO e JPA;
- *TypeSupport*: tipos de linguagens suportadas pelo *framework*;
- *StoreManagement*: manipulação de banco de dados utilizando JDOQL, SQL e JPQL.

Ao lado esquerdo na Figura 5 encontra-se os módulos que dizem que o *framework* tem suporte as especificações JPA e JDO, onde JDOPersistence demonstra que tem suporte as linguagens JDOQL, SQL, e JPQL. E mais abaixo na mesma figura encontra-se JPAPersistence que da suporte a JPQL e a SQL.

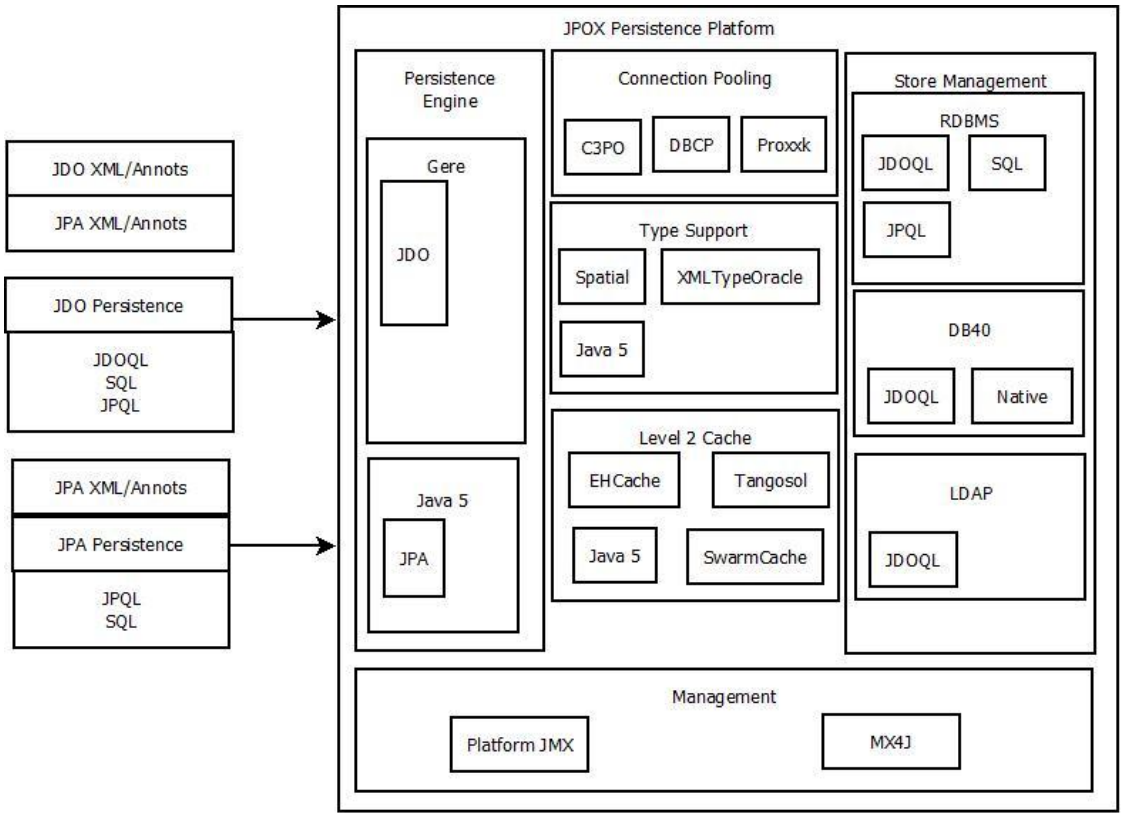

Figura 5 - Arquitetura do *Framework* DATANUCLEUS
Fonte – Datanucleus (2011)

3.3.4.3 Detalhes de implementação

De acordo com Datanucleus (2011), é possível gerar classes dinamicamente na memória para persistir objetos, tudo isso sem qualquer dano físico “classe” de arquivos. Para isso, é preciso usar a classe *ClassLoader* personalizado, bem como a utilização de várias API’ s para JDO.

Para fazer uso da classe gerada, é preciso usar metadados de como ele será mantido. Em primeiro lugar é preciso obter um objeto *JDOMetadata* para preencher. Uma vez com esse objeto obtido é preciso preenchê-lo, por isso usa-se a API do (MetadataJDO2.3).

3.3.4.4 Classes necessárias

Esse *framework* pode ser utilizado com as especificações JPA ou JDO, onde de acordo com a especificação escolhida vai ser requerido um conjunto de classes

específicas. No caso do desenvolvedor precisar usar uma linguagem de manipulação de dados, Campos (2010) diz que o *framework* oferece suporte as seguintes: JDOQL, SQL e JPQL.

3.3.4.5 Persistência utilizando *framework*

Esse *framework* permite a utilização das especificações JPA e JDO para persistência de dados em um aplicativo. Para persistir seus dados através das especificações é só seguir o que já foi disposto nas seções 3.2.1 para JDO e 3.2.2 para JPA.

4 DESENVOLVENDO APLICAÇÕES COM OS *FRAMEWORKS*

No desenvolvimento procurou-se usar somente a capacidade específica de cada *framework* evitando assim, a necessidade de possíveis integrações a outros *frameworks*. O foco do trabalho foi o desenvolvimento de sistemas *desktop* evitando assim projetos de outros tipos que não fossem esse. O projeto foi desenvolvido em três camadas, sendo elas: persistência, negócio (igual para todas), e visão.

Esse capítulo apresenta a maneira em como realizar a persistência de dados utilizando cada um dos *frameworks* estudados, em um aplicativo que foi criado pensando-se em ter todos os relacionamentos possíveis entre tabelas.

4.1 BANCO DE DADOS

A etapa de construção do banco de dados se preocupou em selecionar a modelagem de um problema real onde fossem identificados todos os tipos de relacionamentos possíveis entre as tabelas, para que o trabalho de avaliação dos *frameworks* fosse o mais abrangente possível. Em seguida foi feita a escolha de dois SGBD' s diferentes para realizar o teste dos *frameworks* em dois gerenciadores de banco de dados distintos, com o objetivo de efetivamente validar o processo de conexão, acesso aos dados, avaliando o seu desempenho. Os dois SGBD' s selecionados foram: *MySQL* pois é um banco que muitos *frameworks* têm suporte, dentre eles, pode-se citar os do estudo e, o *Firebird,* por ser um banco que está a cada dia mais em evidência no âmbito acadêmico, mas também para demonstrar a possibilidade de seu uso.

4.1.1 Modelo Utilizado

Para criar o banco foi usado um modelo de entidade relacionamento encontrado na internet, que convertido em modelo relacional pôde finalmente ser usado para o desenvolvimento de aplicativos com os *frameworks*.

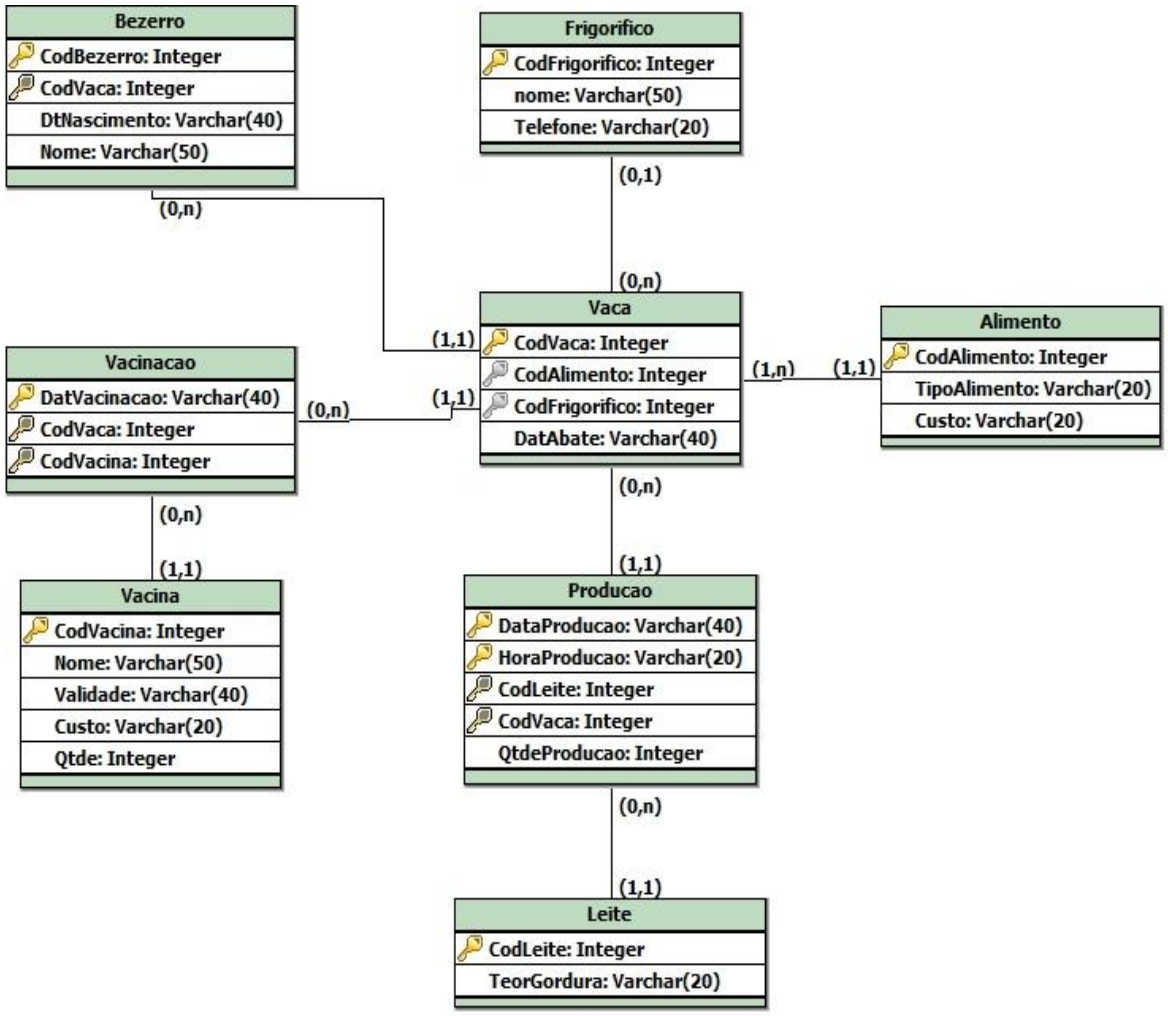

Figura 6 - Modelo para gerar o Banco de dados
Fonte - Autoria própria

Para resolver o problema da impedância entre a tecnologia orientada a objetos e o modelo relacional, surgiram técnicas de mapeamento objeto-relacional que permitem a tradução dos dados de um modelo para o outro de maneira transparente para o desenvolvedor.

Na plataforma de desenvolvimento Java, o mapeamento objeto-relacional foi padronizado por meio de especificações de persistência, que estabelecem maneiras de realizar o mapeamento em Java. As especificações fornecem as diretrizes para realizar os mapeamentos e estas são disponibilizadas na própria API da especificação ou implementadas em *frameworks* de persistência.

Com o crescente aumento na utilização de técnicas de mapeamento objeto-relacional, surgiram várias especificações e *frameworks* para a plataforma Java, por isso, para determinar qual delas utilizar é uma decisão crucial para empresas de Tecnologia da Informação.

Dentro do modelo apresentado, deve-se observar que existem diversos tipos de relacionamentos, que serão mapeados de maneira diferente pelos *frameworks* de persistências estudados.

Quando se mapeia os objetos para o modelo relacional cria-se o objeto-relacional, de modo que o programador não precisa se preocupar com a disposição dos dados nas tabelas, permitindo focar na manipulação de objetos para solução de problemas de negócio.

De acordo com Pinheiro (2005), a relação entre uma classe e uma tabela bem como propriedades e colunas, é feita de forma direta. No caso dos relacionamentos entre objetos podem ser representados no banco de dados por meio de chaves estrangeiras e restrições de acordo com a cardinalidade dos relacionamentos. Para as hierarquias de classe, deve-se selecionar uma estratégia de acordo com o modelo orientado a objetos desejado.

Com relação ao mapeamento de relacionamentos entre os objetos, de acordo com Pinheiro (2005), uma associação deve possuir as seguintes informações:

- Relacionamento do Objeto;
- Classe Origem e Classe de Destino;
- Cardinalidade (1-1, 1-n, n-n);
- Regra de Leitura, Gravação e Inclusão (varia de acordo com o tipo do relacionamento: agregação, composição, associação);
- Tabela que implementa o relacionamento no ambiente de banco dedados;
- Coluna que implementa o relacionamento no ambiente de banco dedados;
- Atributo da classe origem que implementa o relacionamento.

4.1.2 MySQL

Este banco foi escolhido devido ao fato de que todos os *frameworks* em estudo dão suporte a ele, sendo possível realizar os mesmos testes em todos os *frameworks*, exceto se o *framework* fosse voltado para aplicativos web.

4.1.2.1 Propriedades para conexão

Para se conectar a qualquer banco de dados do MySQL, deve-se ter pelo menos as seguintes propriedades no seu arquivo de configurações, que seriam:

- Usuário: root;
- Senha: em branco ou ainda pode ser admin, ou alguma que tenha sido é claro configurada na hora da instalação do servidor;
- Classe do *driver*: com.mysql.jdbc.*Driver;*
- URL: jdbc:mysql://localhost:3306/pecuarista.

4.1.2.2Condições especiais para mapeamento

Devido ao MySQL não aceitar que os *frameworks* gerenciem o auto incremento das PK's (*Primary key*), então foi feito através do próprio SGBD o qual tem a propriedade de gerar suas chaves primárias com auto incremento.O mapeamento em si foi gerado automaticamente, mas a parte que ficou diferente foi na declaração da chave primária, constante no começo da classe onde fica a declaração de atributos, conforme ilustrado no Quadro 14.

Declaração
@GeneratedValue(strategy = GenerationType.IDENTITY) @Column(name = "CodAlimento", nullable = false) private Integer codAlimento;

Quadro 14 - GenerationType para MySQL
Fonte - Autoria própria

4.1.3 Firebird

Este banco de dados foi escolhido devido a ter distribuição gratuita, também ter seu código fonte aberto para que se o desenvolvedor se interessar venha a melhorá-lo. A escolha do Firebird se concentra no fato que ele tem muitos programas que ajudam na sua manipulação e são gratuitos, ao contrário do MySQL, onde isso é mais raro.

4.1.3.1 Propriedades para conexão

Para se conectar a qualquer banco de dados do Firebird, deve-se ter pelo menos as seguintes propriedades no seu arquivo de configurações, que seriam:

- Usuário: SYSDBA;
- Senha: *masterkey;*
- Classe do *driver*: org.firebirdsql.jdbc.FB*Driver;*
- URL: jdbc:firebirdsql://localhost:3050/c:/Bancos/Pecuarista.fdb.

Referente a localização do banco de dados, ou seja, banco de dados diferentes, URL's diferentes.

4.1.3.2 Condições especiais para mapeamento

Devido ao servidor Firebird aceitar geração automática de incremento pelo aplicativo, então foi escolhido usar as facilidades disponíveis dos *frameworks* quanto ao auto-incremento.O mapeamento em si foi gerado automaticamente, mas também como o banco MySQL a parte que ficou diferente foi na declaração da chave primária, constante no começo da classe onde fica a declaração de atributos.

A declaração *SequenceGenerator* fica antes do nome da classe nas quais também podem ser encontradas as declarações :

- @Table, diz qual tabela que a classe representa no banco;
- @Entity, diz que a classe representa uma entidade do banco de dados.

Geração de valores
@SequenceGenerator(name="sequencia", sequenceName="GEN_ALIMENTO_ID", allocationSize=1)

Quadro 15 - SequenceGenerator Firebird
Fonte - Autoria própria

A declaração de *GeneratedValue* fica acima da declaração da chave primária de cada classe.

Geração de valores
@GeneratedValue(strategy = GenerationType.SEQUENCE, generator="sequencia")

Quadro 16 - GenerationType voltado ao Firebird
Fonte - Autoria própria

4.2 PROJETO UTILIZANDO HIBERNATE

No desenvolvimento de novos aplicativos com este *framework* pode se utilizar a sua API de persistência ou a especificação a que ele tem suporte que seria a JPA.

Na persistência de dados utilizando a API ou a especificação se tem a propriedade genérica aos dois que seria *dialect*. De acordo com Hibernate (2011), a propriedade *dialect* (dialeto) seria como o *framework* encapsula todas as diferenças em como ele irá se comunicar com um banco de dados específico para fazer algumas tarefas específicas que seriam: obtenção de um valor de sequência para chave primária ou a estruturação de um comando de seleção. Como o estudo foi realizado com dois SGBD's distintos, então a propriedade *dialect* poderá ser preenchida conforme ilustrado no Quadro 4.

Propriedade dialect (hibernate.dialect)	**Banco de dados usado**
org.hibernate.dialect.MySQLDialect	MySQL
org.hibernate.dialect.FirebirdDialect	Firebird

Quadro 17 - Definindo propriedade dialect Hibernate
Fonte - Autoria própria

4.2.1 Utilizando API

Na utilização da persistência de dados com a API do Hibernate, deve-se deixar claro que não será preciso fazer seu código com nenhuma especificação, devido a sua API possuir sua própria maneira de persistir dados.

4.2.1.1 Conexão

Quando se faz a utilização destes *framework*s têm-se algumas facilidades ao usar a IDE Net*Beans*, tais como:

- Já conter as bibliotecas necessárias para o desenvolvimento de aplicativos junto a IDE, bastando integrá-las ao programa que estiver sendo desenvolvido.
- Possibilidade de chamar arquivos que são do *framework* através do menu: arquivo>novo> , onde se vai em categoria Hibernate e encontram-se todos os arquivos que podem ser utilizados pelo *framework*.

Para começar uma nova conexão com o seu banco de dados utilizando este *framework* deve-se escolher a opção novo arquivo, como mostrado na Figura 7.

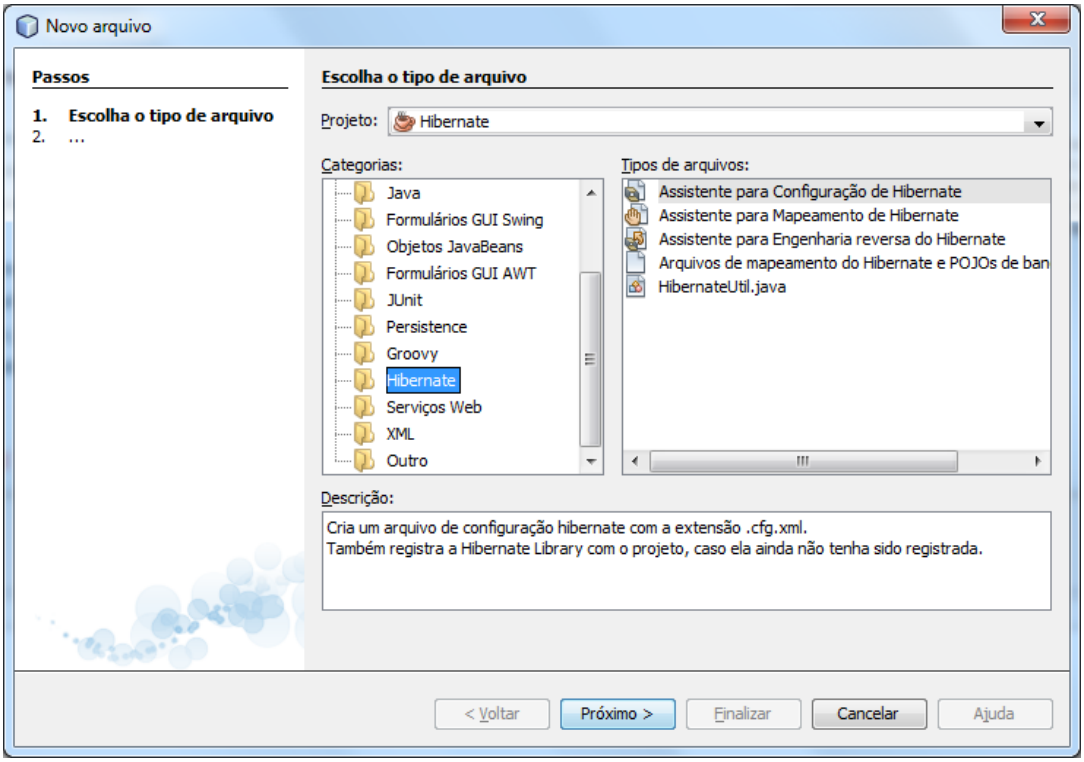

Figura 7 - Novo arquivo de configuração
Fonte - Autoria própria

O assistente para configuração do Hibernate faz a criação do arquivo necessário para conexão, onde serão colocadas as propriedades necessárias para uma conexão efetiva com o banco de dados em questão, e também o carregamento do JDBC para o banco de dados que for usado no desenvolvimento do projeto.

A figura 8 mostra opções relativas à Nome e Local onde poderá ficar o arquivo de configuração.

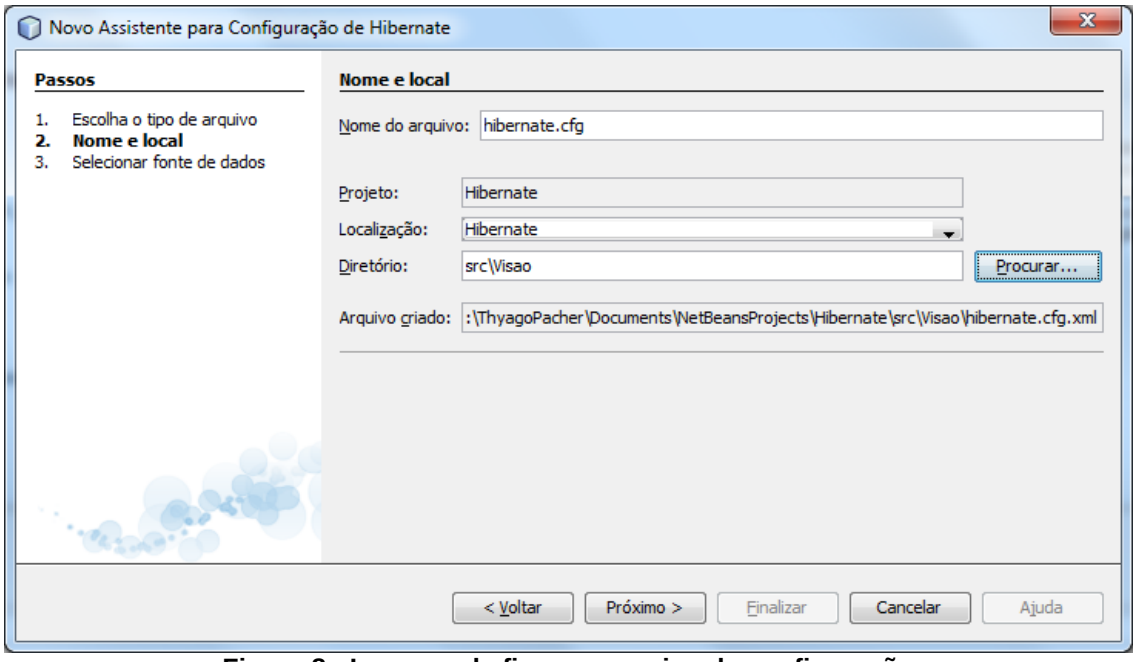

Figura 8 - Lugar onde ficara o arquivo de configuração
Fonte - Autoria própria

A figura 9 permite que uma nova conexão com um determinado banco de dados seja criada, ou ainda, é possível selecionar uma conexão já existente, a fim de ser reutilizada na nova aplicação em construção.

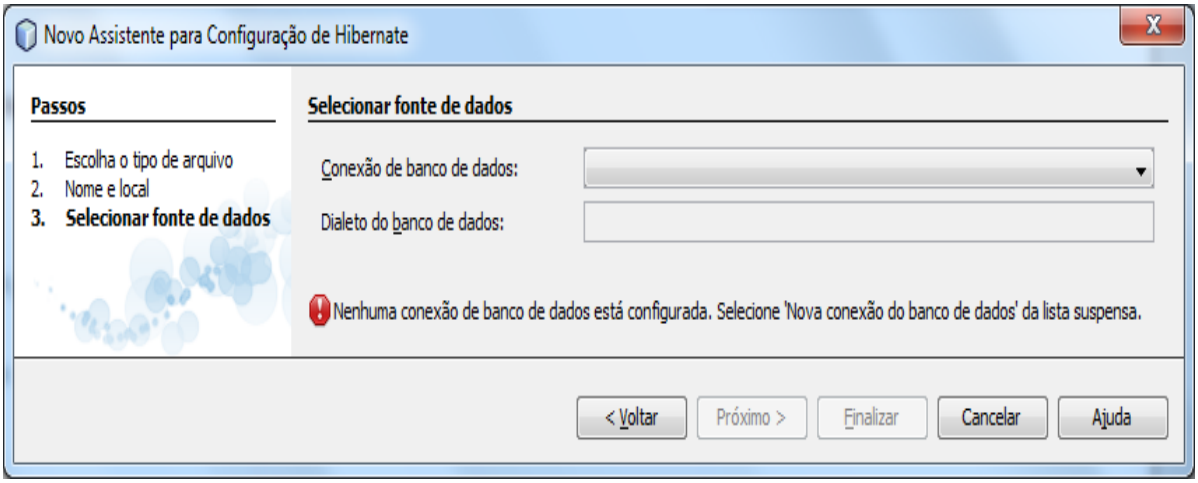

Figura 9 - Selecionar fonte de dados
Fonte - Autoria própria

Caso a opção seja de criar uma nova conexão, o Hibernate possui um conjunto de *drivers* carregados por padrão que poderiam ser utilizados, que seriam:

- Java DB(Embedded e Network)
- JDBC-ODBC Bridge

- MySQL
- Oracle OCI e Thin
- PostgreSQL

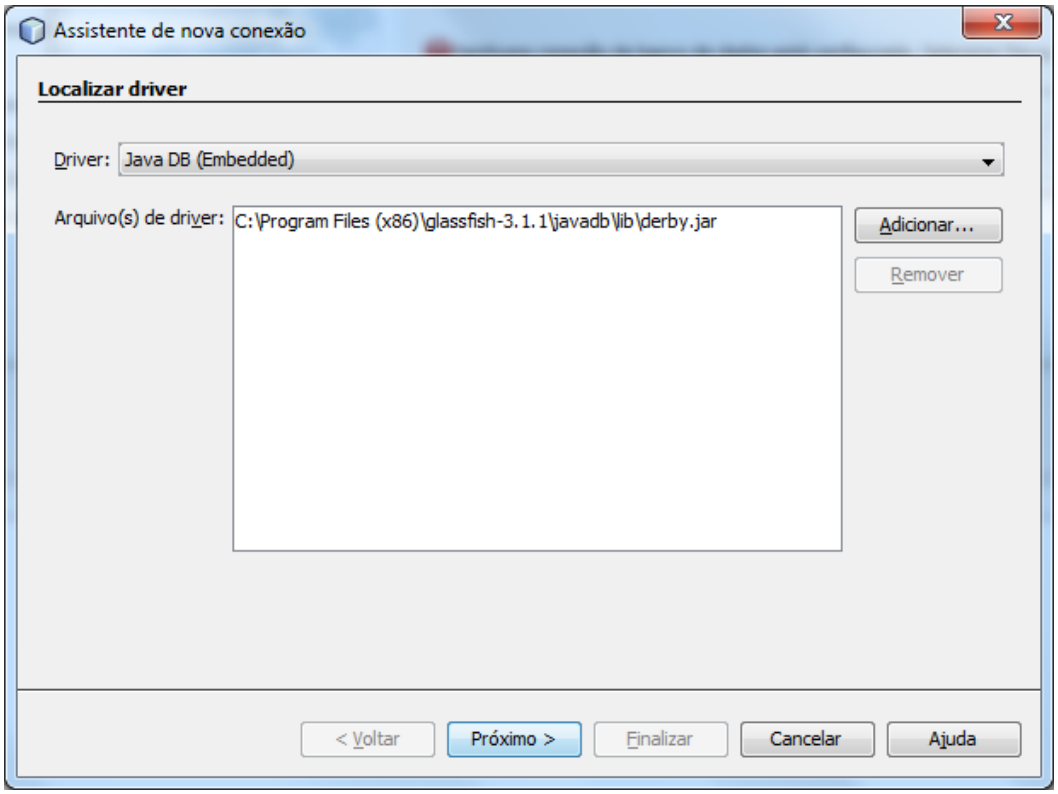

Figura 10 - Localizando *driver*
Fonte - Autoria própria

Para poder adicionar novos arquivos com extensão JAR referentes a bibliotecas do *driver,* a seguinte tela virá antes de poder efetivamente usá-lo.

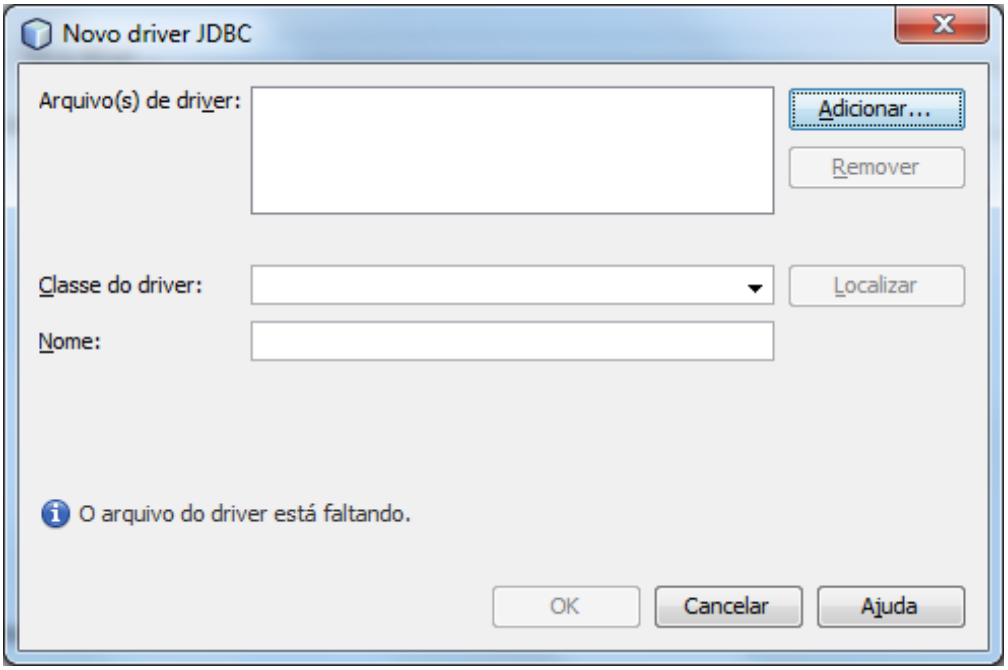

Figura 11 - Adicionando arquivos de *driver*
Fonte - Autoria própria

Se o *driver* não estiver na listagem acima ainda é possível procurar para ver se não existe algum JDBC compatível com o *framework* para adicionar. A seguir aparecerá a tela onde é igual à tela que é visto em vários programas para abrir um novo arquivo. Assim deve-se procurar um novo arquivo para carregar, no caso é permitido arquivos JAR e zip.

Na figura 12, é apresentado o conjunto de *drivers* utilizados para realizar a conexão com o SGBD Firebird. Todos devem ser selecionados seguido do *click* no botão *Open* ou Abrir, o que depende da versão do NetBeans utilizado.

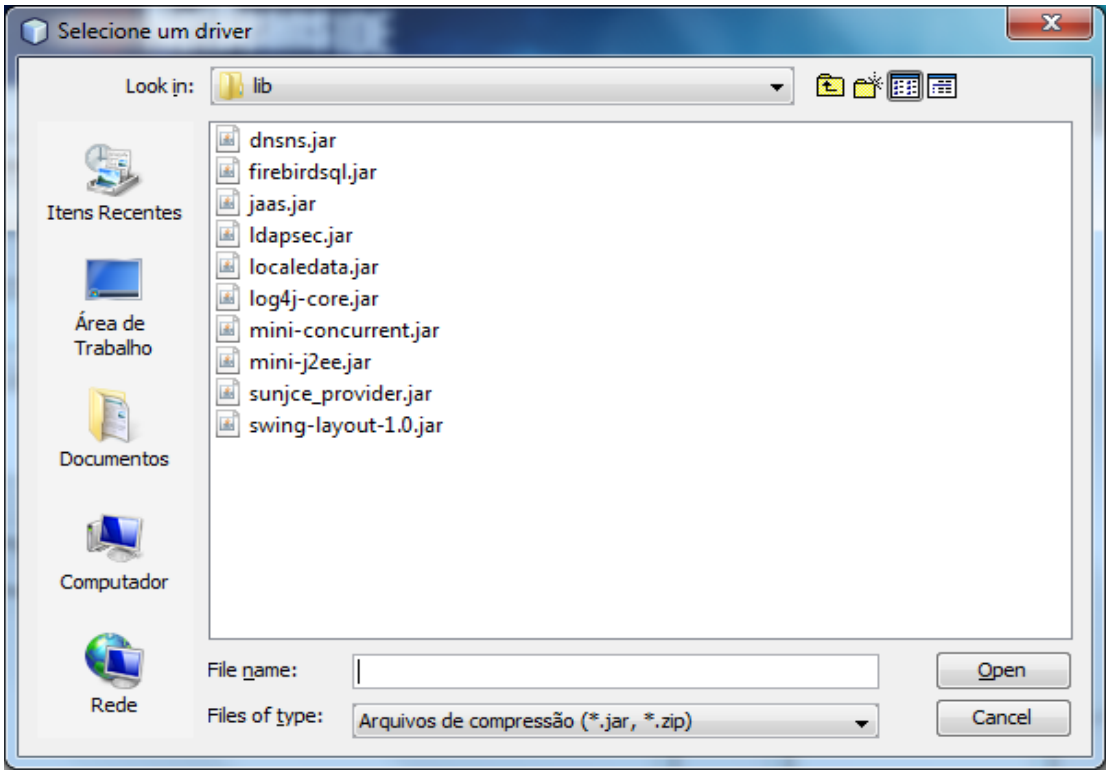

Figura 12 - Selecionando o *driver*
Fonte - Autoria própria

Quando o arquivo estiver adicionado aparecerá a seguinte tela para conferência:

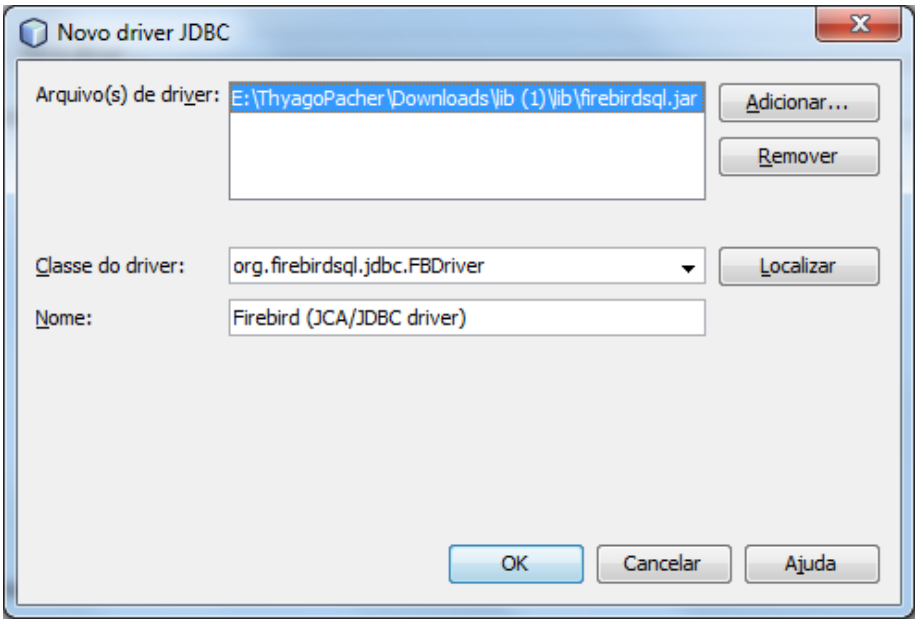

Figura 13 - Novo *driver* adicionado
Fonte - Autoria própria

O passo seguinte seria a configuração da conexão do SGBD escolhido, informando o usuário e a senha, além da localização do banco de dados com o qual

a aplicação fará acesso. É possível ainda testar a conexão criada para verificar se os dados informados estão corretos.

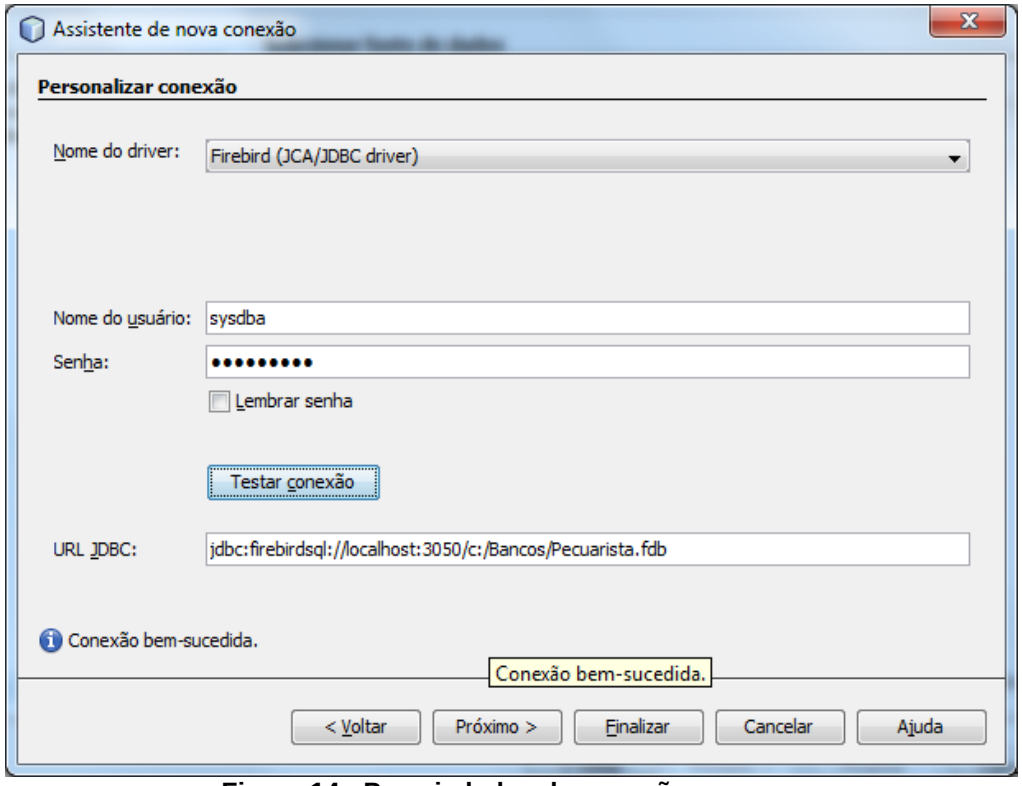

Figura 14 - Propriedades da conexão
Fonte - Autoria própria

É sempre interessante testar se os campos foram preenchidos corretamente por meio da opção testar conexão, pois assim tem-se a certeza que o Hibernate se conecta ao banco, não existindo inconsistências nos parâmetros.

4.2.1.2 Criação de POJO

O significado de POJO estaria em ser a representação de uma classe Java com uma entidade de banco de dados. A criação de classes POJO pode ser feita de duas maneiras com o *framework* Hibernate, utilizando a própria API de persistência dele, ou através da API da especificação JPA. No caso como estamos no tópico sobre como desenvolver um aplicativo com a API do Hibernate, então será falado como criar POJO através dela.

Para começar deve-se entrar em Novo Arquivo escolher a opção de categoria Hibernate e, após, Arquivos de mapeamento do Hibernate e *POJOS* de banco.

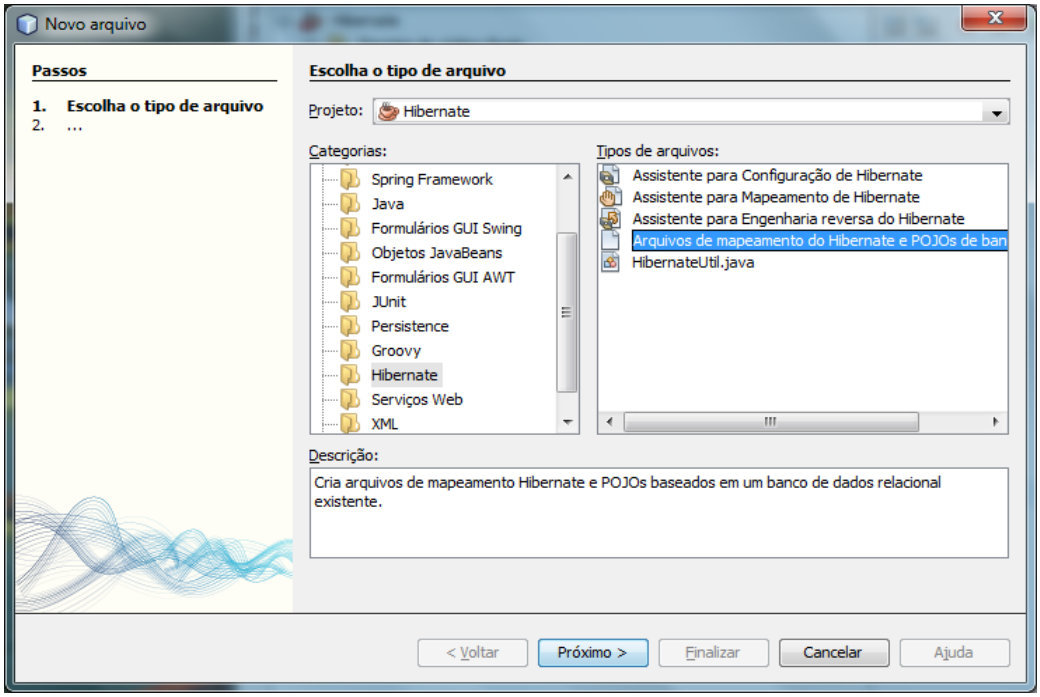

Figura 15 – Mapeamento e criação de *POJO*
Fonte - Autoria própria

Em seguida clica-se em próximo, para definir as opções relativas a mapeamento e a criação de *POJO*. A figura 15 pode ser visto que se têm duas opções:

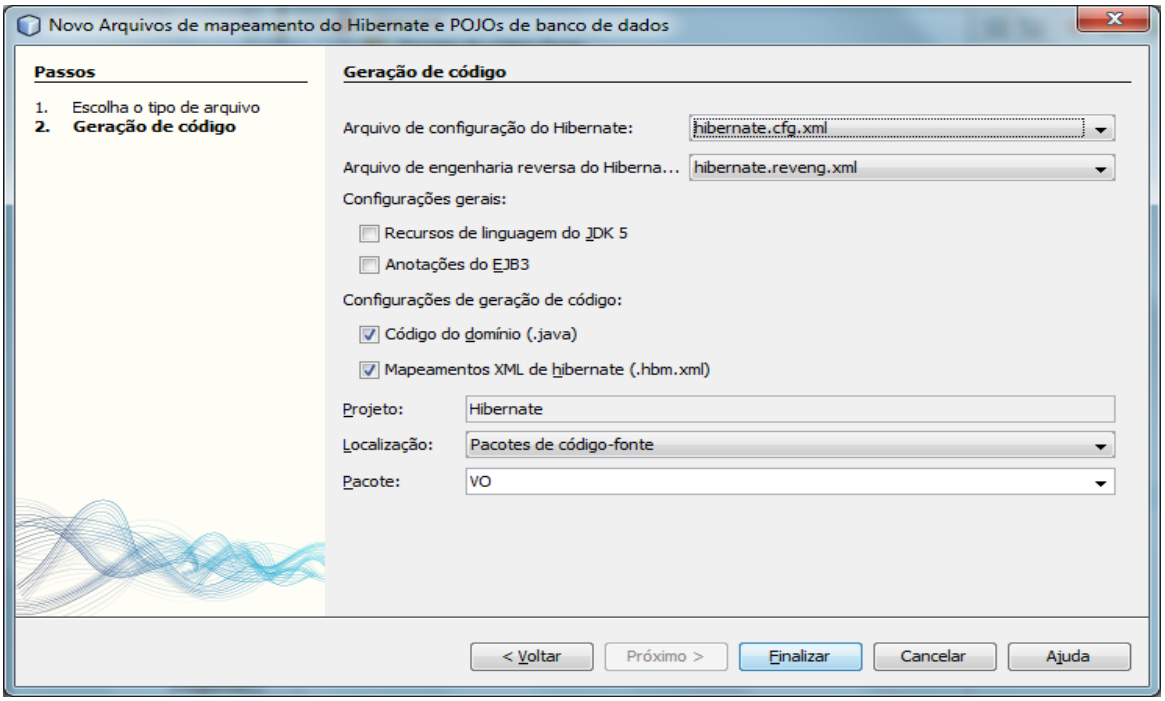

Figura 16 - Opções para mapeamentos, e POJO.
Fonte - Autoria própria

Na figura 16 há opções relativas à criação de mapeamentos com a API do Hibernate ou via anotações diretamente nas classes POJO. Dentro dessa figura ainda tem a criação de código de domínio que seria voltado para a criação de POJO. É importante também considerar onde está definido o pacote, pois ficará mais organizado.

Ao clicar sobre finalizar, tudo o que foi escolhido será feito dentro do projeto de forma automatizada.

4.2.1.3 Métodos de persistência

Os métodos envolvendo a persistência de dados não mudam quando está utilizando o mesmo *framework* ou a mesma especificação, então se souber como fazer a persistência de dados para uma classe, também saberá como fazer para outra, só mudando onde tinha o nome da classe A, por exemplo, que possa ter sido persistido primeiro, para uma classe B que haja interesse em persistir.

4.2.1.3.1 Métodos de alteração de dados

- Para inserir novos objetos dentro do banco de dados:

Método

public String inserirObjeto(Alimento a){
try{
conectar();
session.save(a);
desconectar();
resultado = "Objeto inserido com sucesso";
}catch(Exception e){
resultado = "Problemas ao inserir objeto:\n" + e;
}
return resultado;
}

Quadro 18 - Método inserir Hibernate
Fonte - Autoria própria

- Para atualizar objetos que estejam dentro do banco de dados, deve-se utilizar antes o método de procura específico, e após, finalmente poderá atualizar com os possíveis dados modificados.

Método

public String atualizarObjeto(Alimento a){
try{
conectar();
session.update(a);
desconectar();
resultado = "Objeto inserido com sucesso";
}catch(Exception e){
resultado = "Problemas ao atualizar objeto:\n" + e;
}
return resultado;
}

Quadro 19 - Método atualizar Hibernate
Fonte - Autoria própria

- Para excluir um objeto deve-se utilizar antes o método de procura específico, e após poderá excluir, deste que o objeto esteja todo preenchido em seus atributos, caso contrário, poderá ocasionar *nullpointerexception*.

Método

public String excluirObjeto(Alimento a){
try{
conectar();
session.delete(a);
desconectar();
resultado = "Objeto inserido com sucesso";
}catch(Exception e){
resultado = "Problemas ao excluir objeto:\n" + e;
}
return resultado;
}

Quadro 20 - Método excluir objeto Hibernate
Fonte - Autoria própria

4.2.1.3.2 Métodos de consulta de dados

Para fazer a consulta de objetos dentro do banco pode utilizar dos seguintes tipos:

- Pesquisa por chave primária

É usado somente quando o campo que está se fornecendo para procura for uma chave primária.

Método

public Alimento procuraCodigo(int codigo){
conectar();
Alimento a = (Alimento) session.get(Alimento.class, codigo);
desconectar();
return a;
}

Quadro 21 - Método consulta session.get
Fonte - Autoria própria

- Pesquisa utilizando createQuery

Para fazer a realização de consulta ainda pode-se utilizar da linguagem HQL que é uma forma de SQL, porém, voltada para orientação a objetos. Por ser voltada para orientação a objetos tem algumas mudanças com relação a SQL.

Procura parcial

```
public List<Alimento> procuraNomeParcial(String tipo){
        List<Alimento> obj = new ArrayList();
       Alimento       a;
conectar();
       Iterator it = session.createQuery("from Alimento where tipoalimento like '%" +
tipo + "%'").list().iterator();
while(it.hasNext()){
a = (Alimento)it.next();
obj.add(a);
}
desconectar();
return obj;
          }
```

Quadro 22 - Procura parcial com createQuery
Fonte - Autoria própria

- Pesquisa com *Query* no arquivo hbm

É a possibilidade de fazer a *query* HQL no arquivo de mapeamento (hbm) e depois só chamá-lo para ser executado e carregar o que foi procurado dentro da sua classe Java que representa a entidade. O Quadro 23, mostra como pode se montar um método de busca chamando uma *query* que esteja no arquivo de mapeamento (hbm). Para se utilizar esse tipo de consulta, é importante notar que se deve passar por parâmetro o que vai ser procurado, através da interface *Query* que se pode usar um set para passar esse parâmetro.

Procura exata

```
public Alimento procuraTipo(String tipo){
       Alimento a = new Alimento();
conectar();
       Query q = session.getNamedQuery("procuraTipo");
q.setString("p", tipo);
       Iterator it = q.list().iterator();
while(it.hasNext()){
a = (Alimento)it.next();
}
return a;
}
```

Quadro 23 - Busca através de query no hbm
Fonte - Autoria própria

A *query* dentro do arquivo de mapeamento XML do Hibernate pode ser colocada com *tag* de <*query*> onde, de maneira simplificada se consegue montar a sua seleção para trazer dados do banco.

Comando de seleção
<query name="procuraTipo"> <![CDATA[from Alimento a where a.tipoalimento = :p]]> </query>

Quadro 24 - Mapeamento de query no hbm
Fonte - Autoria própria

- Pesquisa através da API *Criteria*

  É a possibilidade de se realizar a persistência através da API *Criteria*.

Procura exata

public Alimento procuraNomeExata(String tipo){
Alimento a = new Alimento();
conectar();
Criteria crit = session.createCriteria(Alimento.class);
crit.add(Restrictions.eq("tipoalimento", tipo));
Iterator it = crit.list().iterator();
while(it.hasNext()){
a = (Alimento)it.next();
}
desconectar();
return a;
}

Quadro 25 - Pesquisa através da API Criteria
Fonte - Autoria própria

4.2.2 Especificação JPA

A utilização de uma especificação é algo genérico, então se pode sempre tomar como base um para utilização em outro *framework*. Para começar a usar o *framework* com a especificação JPA é necessário adicionarmos uma unidade de persistência.

4.2.2.1 Conexão com unidade de persistência

A conexão nesse caso pode ser feita de duas maneiras, que seriam através de um:

- Arquivo de configurações próprio do Hibernate, arquivo de propriedades. Para seguir este caminho, basta ir ao tópico 4.2.1.1 que conterá instruções de como se conectar ao banco através de um arquivo de configurações do *framework*.
- Através da unidade de persistência, onde as propriedades referentes à conexão que ficariam no arquivo de configurações do Hibernate, agora ficam no arquivo da unidade de persistência.

Para fazer uma nova unidade de persistência, que é elemento considerado essencial no caso de persistir dados com as especificações, é possível ser feito selecionado novo arquivo, escolhendo a aba *Persistence*, e após, terá ao seu lado direito a escolha para criar uma nova Unidade de Persistência.

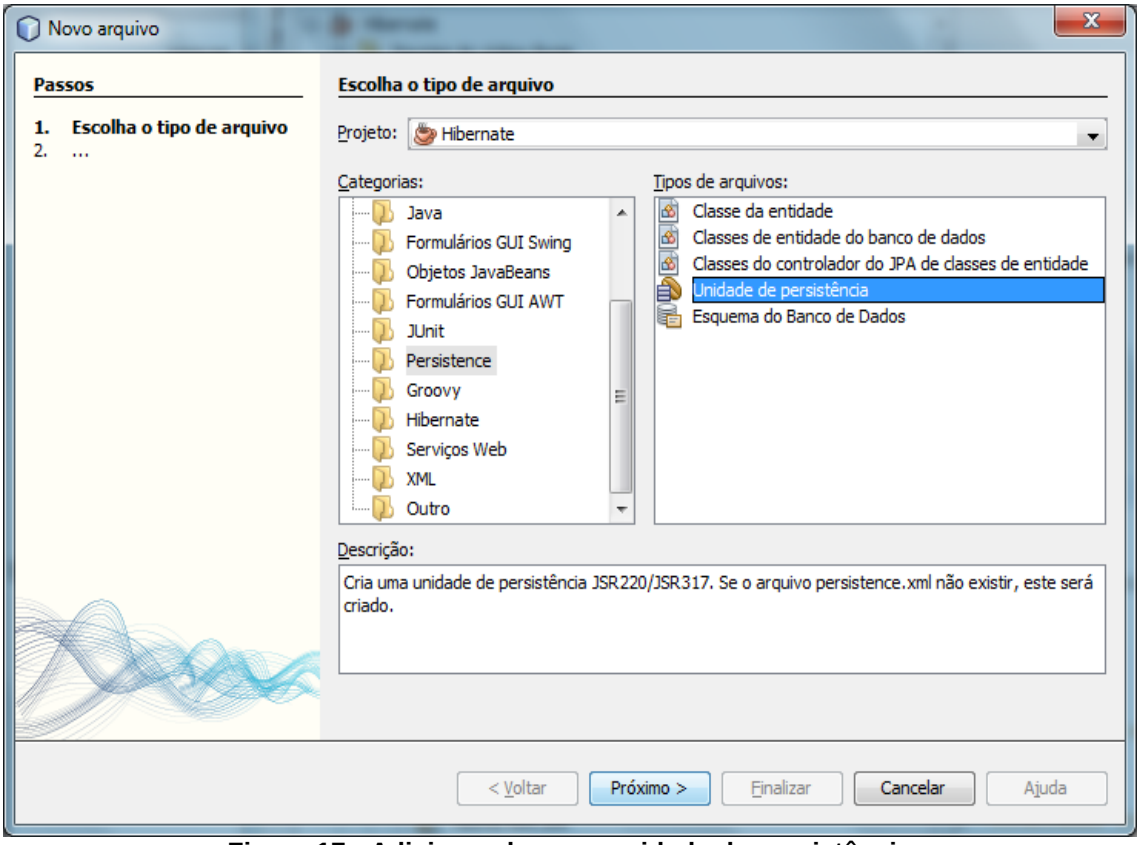

Figura 17 - Adicionando nova unidade de persistência
Fonte – Autoria própria

Na próxima etapa pode-se decidir qual provedor de persistência que será usado junto à especificação que seria escolhido na opção biblioteca de persistência, em seguida se escolhe a conexão com o banco de dados ou se cria uma nova. Também no final existe a possibilidade de criar, apagar e criar, ou não fazer nada quanto à possibilidade de se usar as classes POJO para a criação de tabelas do banco de dados.

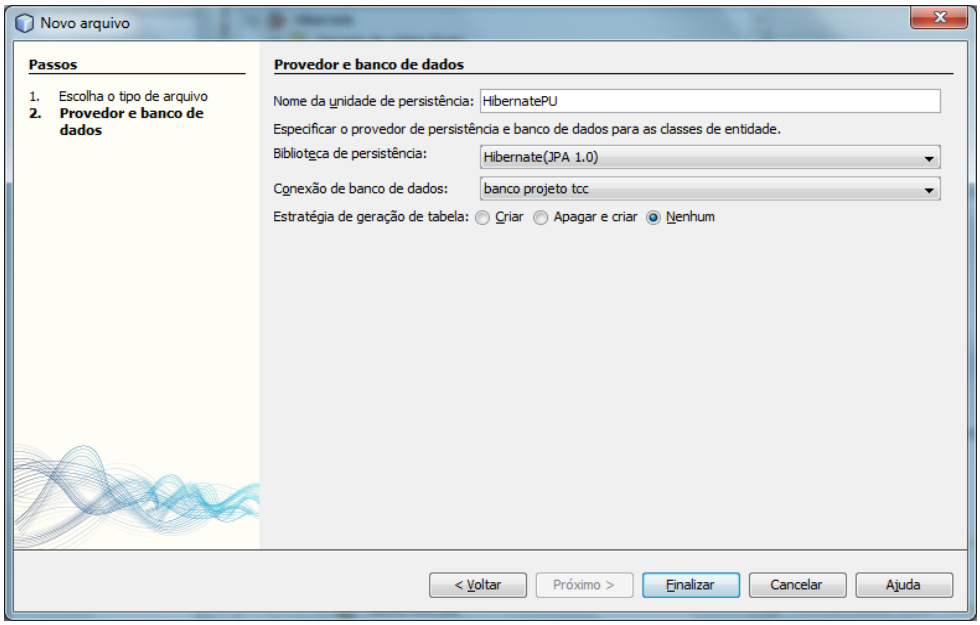

Figura 18 - Provedor e banco de dados
Fonte - Autoria própria

A figura 19 mostra como é a tela de conexão. Para saber como preencher é só seguir o tópico 4.1.2 que comenta sobre dois possíveis bancos de dados a serem utilizados, e o que preencher em cada propriedade.

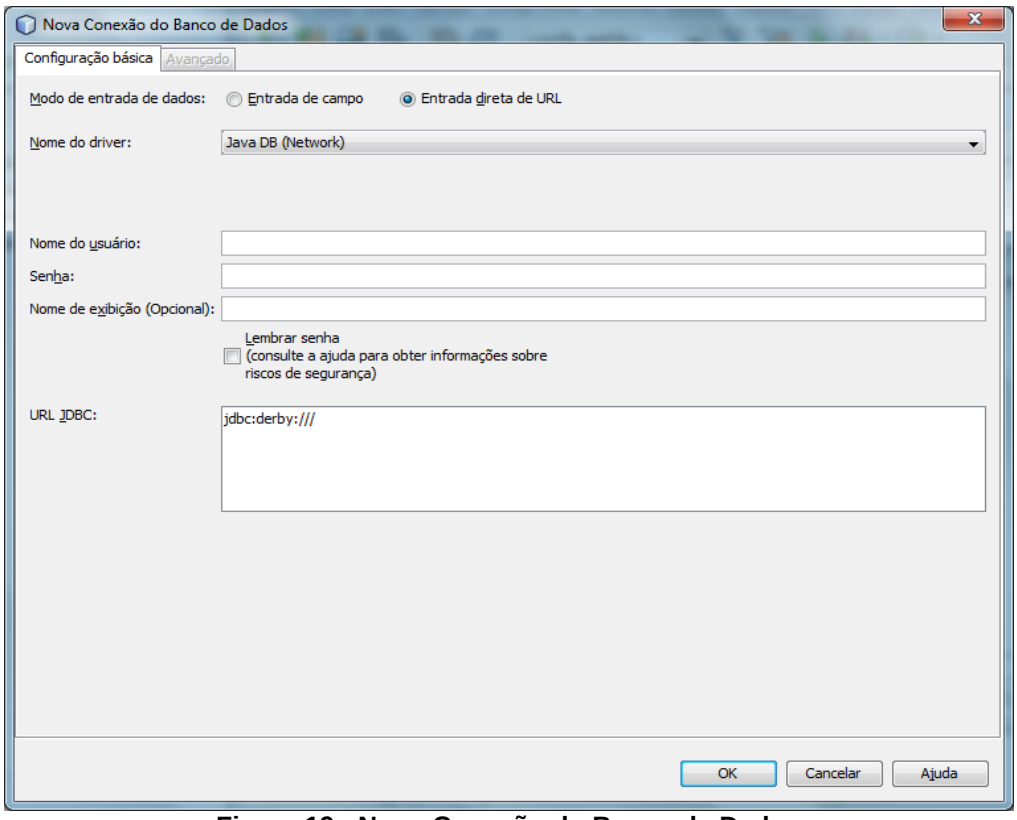

Figura 19 - Novo Conexão do Banco de Dados
Fonte - Autoria própria

4.2.2.2 Criação de POJO

Para criar suas classes de POJO utilizando essa especificação deve-se ir a novo arquivo em seguida escolher a opção *Persistence*, e depois do lado direito a opção referente a Classes de entidade do banco de dados.

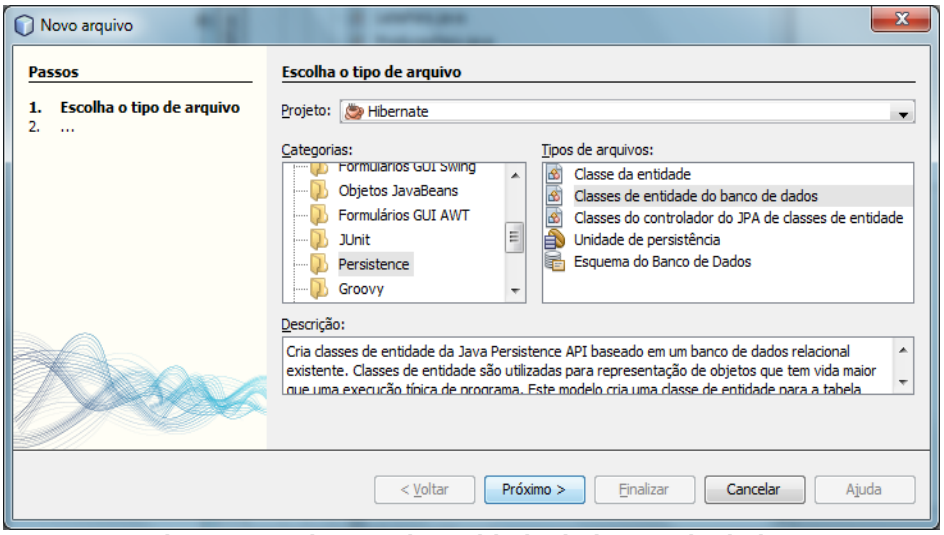

Figura 20 - Classes de entidade do banco de dados
Fonte - Autoria própria

A próxima etapa mostra a escolha do banco de dados, e quais tabelas serão escolhidas para serem usadas conforme apresentado na figura 21.

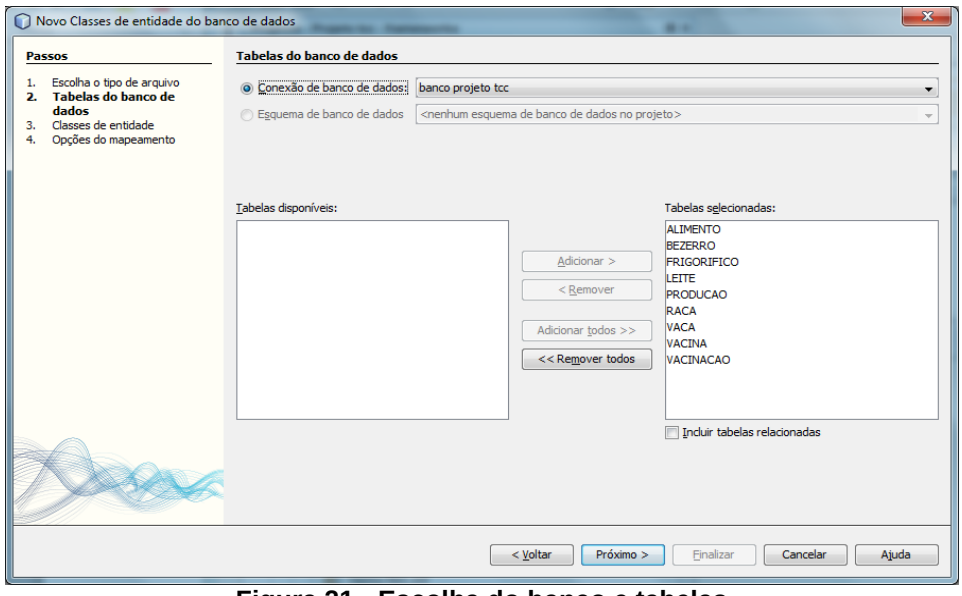

Figura 21 - Escolha do banco e tabelas
Fonte – Autoria própria

A figura 22 exemplifica como serão formadas as anotações dentro das classes de entidades.

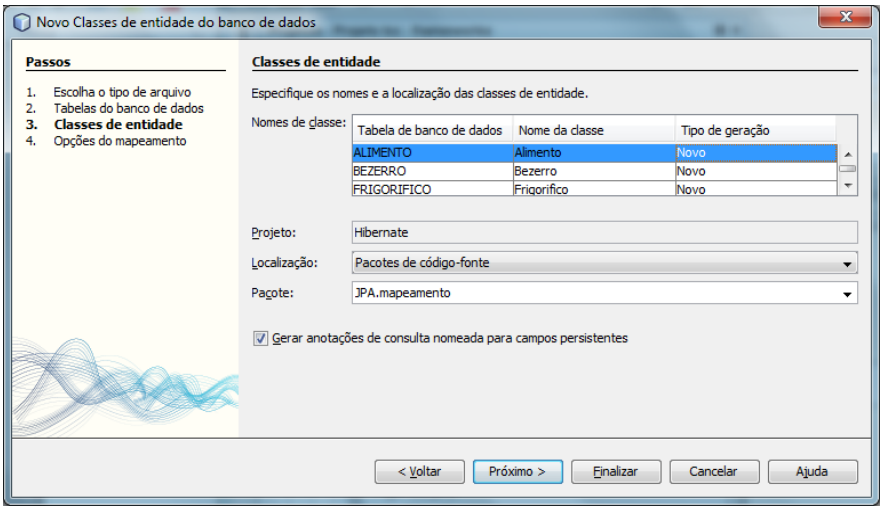

Figura 22 - Nomes classes POJO e gerar anotações
Fonte - Autoria própria

A próxima etapa deverá ser escolhida as opções relativas ao mapeamento que será feito através de suas classes de POJO.

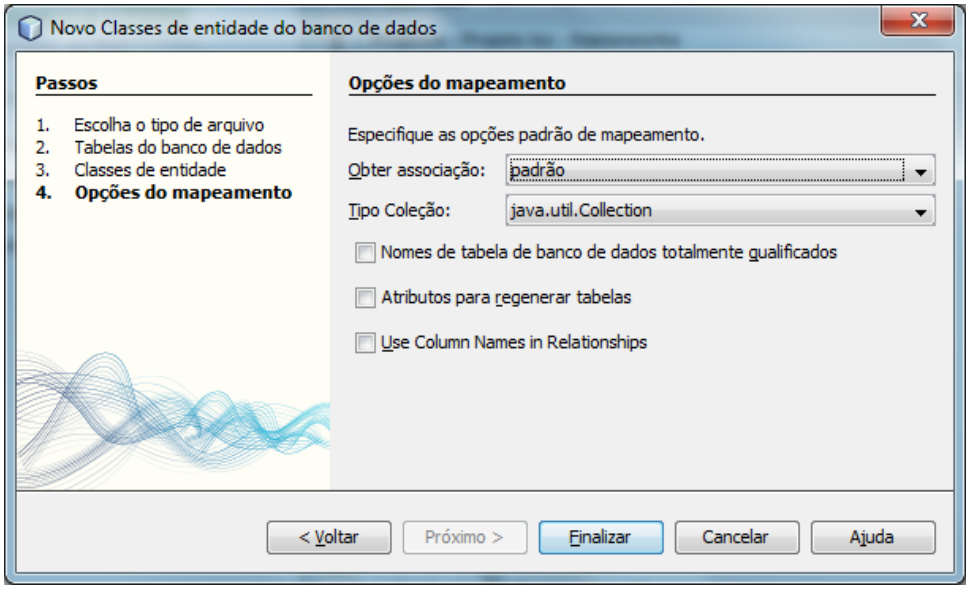

Figura 23 - Opções de mapeamento
Fonte - Autoria própria

Ao se clicar em finalizar as opções de mapeamento as classes de POJO, ou seja, de entidade serão criadas automaticamente.

4.2.2.3 Métodos de persistência

Antes de começar a persistir os dados deve-se proceder da seguinte maneira:

- Criação da fábrica de entidades, onde se declara o nome da unidade de persistência.
- Criar o gerenciador de entidades, onde este possibilita a criação que *EntityTransaction* que tem o necessário para chamar os métodos de persistência.

De maneira geral foi assim encontrado na maioria das pesquisas, mas para facilitar o desenvolvimento com esse *framework* separamos dois métodos: conectar, e desconectar. Lembrando que dessa maneira ainda é necessário com que sejam instanciadas as classes que criam a fábrica de entidades e a entidade de transação. A declaração dessas classes pode ser vista um exemplo no Quadro 26.

Declaração
EntityManagerFactory emf; EntityManager em; EntityTransaction tx;

Quadro 26 - Declaração de classes para se utilizar JPA
Fonte - Autoria própria

- Método conectar ();

Método

public void conectar(){
emf = Persistence.createEntityManagerFactory("hibernate");
em = emf.createEntityManager();
tx = em.getTransaction();
tx.begin();
}

Quadro 27 - Método para conexão com JPA
Fonte - Autoria própria

- Método desconectar ();

Método

public void desconectar(){
em.flush();
tx.commit();
em.close();
}

Quadro 28 - Método para desconectar com JPA
Fonte - Autoria própria

4.2.2.3.1 Métodos de alteração de dados

Quando se pretende realizar alguma alteração dentro do banco de dados, os métodos disponíveis seriam de inserir, atualizar, e excluir.

- Para inserir novos objetos dentro de um banco de dados utilizando a especificação JPA, tem que ser feito de acordo como Quadro 29:

Método

public String inserirObjeto(Alimento a){
try{
conectar();
em.persist(a);
desconectar();
resultado = "Objeto inserido com sucesso";
}catch(Exception e){
resultado = "Erro ao inserir objeto:\n" + e;
}
return resultado;
}

Quadro 29 - Método para inserir novos objetos JPA
Fonte - Autoria própria

- Para atualizar objetos pode ser feito através do método demonstrado no Quadro 30:

Método

public String atualizarObjeto(Alimento a){
try{
conectar();
em.merge(a);
desconectar();
resultado = "Objeto inserido com sucesso";
}catch(Exception e){
resultado = "Erro ao inserir objeto:\n" + e;
}
return resultado;
}

Quadro 30 - Método atualizar com JPA
Fonte - Autoria própria

- Para excluir objetos através da especificação JPA pode seguir no Quadro 31:

Método

public String excluirObjeto(Vacina v){
try{
conectar();
v = em.merge(v);
em.remove(v);
desconectar();
resultado = "Objeto excluido com sucesso";
}catch(Exception e){
resultado = "Erro ao excluir objeto:\n" + e;
}
return resultado;
}

Quadro 31 - Método excluir com JPA
Fonte - Autoria própria

4.2.2.3.2 Métodos de consulta de dados

Para realizar novas consultas utilizando a especificação JPA, os seguintes métodos podem ser usados:

- Procura usando *find*

Utilizado para quando o campo procurado seja uma chave primária da tabela. Para utilizar o *find* deve ser passado como parâmetros dentro dele, a classe qual representa essa tabela, e o parâmetro que vai ser comparado com a chave primária.

Método

public Vacina procuraCodigo(int codigo){
conectar();
Vacina v = em.find(Vacina.class, codigo);
return v;
}

Quadro 32 - Método *find* procura por PK
Fonte – Autoria própria

Também para quem gosta de comandos feitos em linguagem SQL é possível fazer a utilização do HQL nos métodos de seleção. Os métodos de seleção com *Query* são:

- createQuery

Utilizado quando se quer fazer um comando HQL diretamente no método.

Método

public List<Alimento> procuraNomeParcial(String tipo){
List<Alimento> obj = new ArrayList();
Alimento a;
conectar();
Iterator it = em.createQuery("from Alimento where tipoalimento like '%"
+ tipo + "%'").getResultList().iterator();
while(it.hasNext()){
a = (Alimento)it.next();
obj.add(a);
}
return obj;
}

Quadro 33 - Procura JPA com createQuery
Fonte - Autoria própria

- *createNamedQuery*

Utilizado para quando se tem o comando de seleção no mapeamento, sendo assim você chama esse comando para ser executado e após, pega o resultado.

Método

public Alimento procuraNomeExata(String nome){
Alimento a = new Alimento();
conectar();
Query q = em.createNamedQuery("Alimento.findByTipoalimento");
q.setParameter("tipoalimento", nome);
Iterator it = q.getResultList().iterator();
while(it.hasNext()){
a = (Alimento)it.next();
}
return a;
}

Quadro 34 - Procura JPA com createNamedQuery
Fonte - Autoria própria

- createNativeQuery

Outro comando de seleção que é feito dentro do método. Para usar este, devem ser passados por parâmetro o comando de seleção que no caso é o HQL e a classe a qual a representa a tabela que foi feita a seleção.

Método

public Bezerro procuraNomeExata(String nome) {
String sql = "select * from Bezerro where nome = '" + nome + "'";
Bezerro b = new Bezerro();
conectar();
Iterator it = em.createNativeQuery(sql, Bezerro.class).getResultList().iterator();
while (it.hasNext()) {
b = (Bezerro) it.next();
}
return b;
}

Quadro 35 - Procura JPA com createNativeQuery
Fonte - Autoria própria

4.3 PROJETO UTILIZANDO TOPLINK

Este *framework* utiliza para sua persistência a especificação JPA, entretanto, como se tem elaborações diferentes, deve-se pensar que é muito difícil à mesma organização ter feito mais de um *framework*, então se tem algumas peculiaridades feitas além do que a especificação diz que tem que ser feito. Tais peculiaridades são feitas para deixar o *framework* mais robusto do ponto de vista de quem o desenvolveu.

4.3.1 Especificação JPA

Para realizar sua persistência com este *framework* e utilizando a especificação JPA basta seguir o que foi dito no tópico 4.2.2 de Persistência JPA com Hibernate. Algumas coisas é claro que ficam diferentes de outras no que foi feito com Hibernate dentre elas pode-se citar:

- Geração de incremento na chave primária: como este *framework* não tem suporte ao servidor Firebird, então para esse *framework* foi usado a segunda opção de banco relacionada no trabalho que era o MySQL. Nisso também se tem um mapeamento um pouco diferenciado, pois a geração de auto-incremento não é aceito pelo banco que foi escolhido como alternativa se for feito via aplicativo, então foi deixado para o banco cuidar dessa parte com sua opção de auto-incremento. A opção mudada era de como o JPA se comportaria para gerar ou não a chave primária.

4.4 PROJETO UTILIZANDO SPRING

Um projeto utilizando este *framework* pode ser realizado com JDBC o que pode ser feito através de comandos SQL, o benefício é que ao invés de algumas linhas de código, essas podem ser evitadas e assim ter somente uma linha. Projetos com este *framework* também podem ser feitos com as especificações, que é usado com outro *framework* para Provedor de Persistência.

4.4.1 Conexão do *framework*

Para realização de uma conexão efetiva com o banco de dados precisamos de que o projeto contenha os seguintes arquivos:

- Biblioteca Java para conexão ao banco;
- Arquivo de configuração, definindo as propriedades necessárias para se conectar ao banco;
- Declaração das classes que serão usadas para persistir dados.

Para obter um novo arquivo de configuração do Spring, pode-se fazer indo em novo arquivo, e dentro da categoria Outro encontra-se (Arquivo de configuração Spring XML) como pode ser visto pela figura 24:

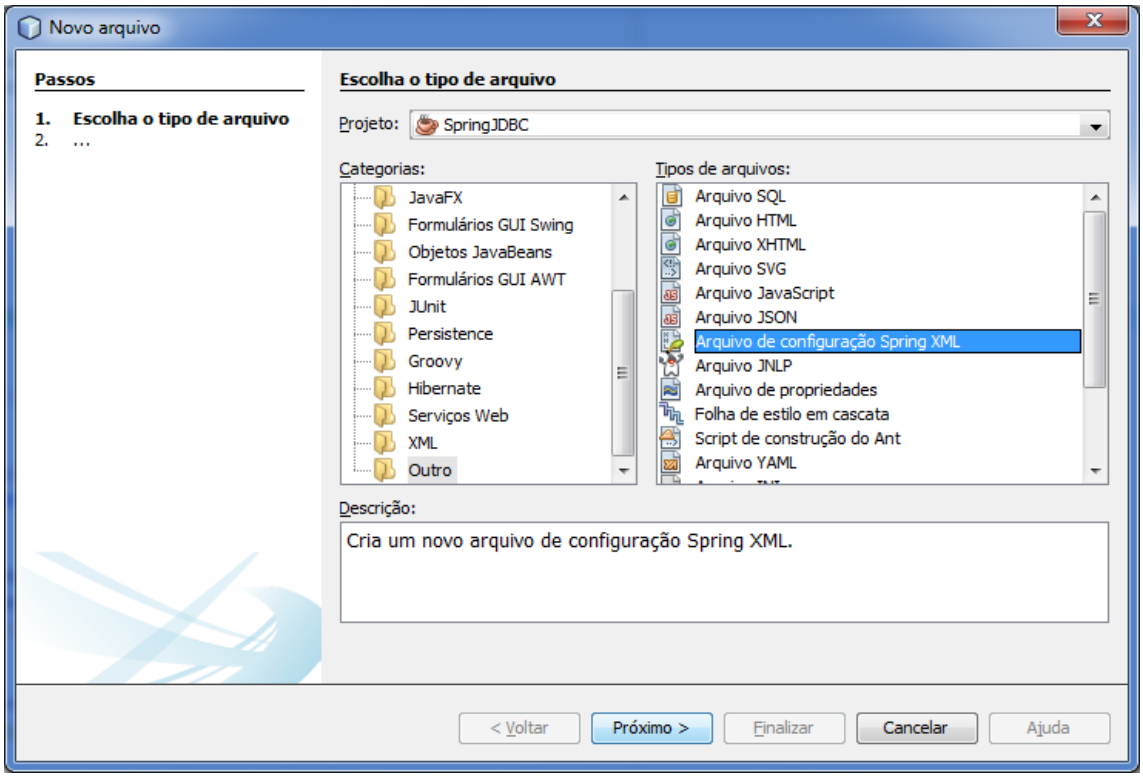

Figura 24 - Obtendo um arquivo de configuração Spring
Fonte - Autoria própria

Para realizar a conexão, é necessário ainda que sejam definidas todas as classes Java que são utilizadas para persistência de dados. A maneira como isto pode ser feito é visto através do Quadro 36:

Declaração
<bean class="Persistencia.AlimentoPers" id="alimento"> <property name="dataSource" ref="dataSource"/> </bean>

Quadro 36 - Declarando classe de persistência na configuração
Fonte - Autoria própria

Na definição das propriedades que o banco irá utilizar, já temos um *bean* com a id de *dataSource* pronto bastando somente preenchê-lo corretamente com as propriedades relativas ao seu banco de dados utilizado no momento. Este *bean* preenchido no arquivo de configurações pode ficar como visto no Quadro 37:

Configuração
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource"> <property name="driverClassName" value="org.firebirdsql.jdbc.FBDriver"/> <property name="url" value="jdbc:firebirdsql://localhost:3050/c:/Bancos/PECUARISTA.FDB"/> <property name="username" value="SYSDBA"/> <property name="password" value="masterkey"/> </bean>

Quadro 37 - Bean dataSource
Fonte - Autoria própria

Para conseguir a persistência, ainda é necessário que seja feita a construção de um método que será responsável por chamar a classe de persistência, para chamá-lo foi colocado o nome de conexão, devido ao mesmo ser uma maneira de conectar efetivamente a persistência.

Método

public AlimentoPers conectar(){
ApplicationContext context = new
ClassPathXmlApplicationContext("Persistencia/SpringXMLConfig.xml");
AlimentoPers ap = (AlimentoPers)context.getBean("alimento");
return ap;
}

Quadro 38 - Método de conexão Spring
Fonte - Autoria própria

4.4.2 Utilizando JdbcTemplate

Utiliza métodos SQL diretamente, assim fica fácil à adaptação de um desenvolvedor de softwares a esse tipo de desenvolvimento. Para realizar a persistência de dados é necessário configurar um novo arquivo de configuração, e para isso foi escolhido fazer diretamente pelo arquivo de configuração do *framework*, sendo que se pretende deixar claro que existem outras maneiras possíveis de ser feito a configuração.

Com o desenvolvimento de um aplicativo utilizando o *framework* Spring, observou-se que ele não tem nenhuma forma de mapeamento que venha diretamente dele, assim, para as classes de entidades foram definitivas a partir dos atributos que contém no banco de dados, e o nome da classe sendo estabelecido pelo mesmo nome que se tem na tabela.

4.4.2.1 Métodos de persistência

Os métodos de persistência relacionados a este *framework* com a utilização do *JdbcTemplate,* são feitos com utilizando linguagem SQL, o que torna para um desenvolvedor que nunca utilizou *frameworks* e tem conhecimento para a linguagem uma adaptação mais simples.

4.4.2.1.1 Métodos de alteração de dados

São os métodos onde de alguma maneira manipulam os dados dentro do banco.

- Inserir: ao utilizar este método, constatou-se que como não foi utilizado mapeamento para as classes POJO, então não podia definir um *generator* de qualquer tipo, então se usou de uma "artimanha" para evitar o uso de *triggers* no banco. Foi feito um método que ia ao banco e via quantos registros se tem naquela tabela, então a partir disso, somando-se mais um poderíamos ter um auto-incremento.

Método

public String inserirObjeto(Alimento a){
a.setCodalimento(maiorRegistro() + 1);
String sql = "insert into alimento (codalimento, tipoalimento, custo) values(?, ?, ?)";
try{
if(jdbcTemplate == null){
res = "JdbcTemplate está nulo você está com problemas:\n Na declaração das classes de persistência";
}else{
this.jdbcTemplate.update(sql, new Object[]{a.getCodalimento(), a.getTipoalimento(), a.getCusto()});
res = "Inserido com sucesso";
}
}catch(Exception e){
res = "Erro ao inserir objeto:\n" + e;
}
return res;
}

Quadro 39 - Inserir com Spring
Fonte - Autoria própria

- Atualizar:

Método

```
public String atualizarObjeto(Alimento a){
      String    sql = "update alimento set tipoalimento=?, custo=? where codalimento=?";
try{
this.jdbcTemplate.update(sql, new Object[]{a.getTipoalimento(), a.getCusto(),
a.getCodalimento()});
res = "Atualizado com sucesso";
}catch(Exception e){
res = "Erro ao atualizar objeto:\n" + e;
}
return res;
}
```

Quadro 40 - Atualizar com Spring
Fonte - Autoria própria

- Excluir:

Método

```
public String excluirObjeto(Alimento a){
      String    sql = "delete from alimento where codalimento=?";
try{
this.jdbcTemplate.update(sql, new Object[]{a.getCodalimento()});
res = "Excluido com sucesso";
}catch(Exception e){
res = "Erro ao excluir objeto:\n" + e;
}
return res;
}
```

Quadro 41 - Método excluir com Spring
Fonte - Autoria própria

4.4.2.1.2 Métodos de consulta de dados

- Seleção voltando um inteiro, o exemplo dado foi usado para gerar o auto incremento, que era necessário para a utilização do banco de dados Firebird com este *framework*.

Método

```
public int maiorRegistro(){
      String sql = "select max(codalimento) from alimento";
int   qtd = this.jdbcTemplate.queryForInt(sql);
return qtd;
}
```

Quadro 42 - Seleção voltando inteiro
Fonte - Autoria própria

- Seleção de um objeto por pesquisa exata:

Método

```
public Alimento procuraNomeExata(String tipo) {
        String   sql = "select * from alimento where tipoalimento = '" + tipo + "'";
        Alimento a;
try{
a = (Alimento) jdbcTemplate.queryForObject(sql,new RowMapper<Alimento>(){
            @Override
public Alimento mapRow(ResultSet rs, int rowNum) throws SQLException{
                    Alimento a = new Alimento();
a.setCodalimento(rs.getInt("codalimento"));
a.setCusto(rs.getString("custo"));
a.setTipoalimento(rs.getString("tipoalimento"));
return a;
}
});
}catch(EmptyResultDataAccessException e){
a = null;
}
return a;
}
```

Quadro 43 - Selecionando por queryForObject
Fonte - Autoria própria

- Selecionando um objeto por pesquisa parcial:

Método

```
public List<Alimento> procuraTodos() {
        String         sql = "select * from alimento";
//para trazer uma listagem
        List<Alimento> obj = (List<Alimento>) jdbcTemplate.query(sql,new RowMapper<Alimento>(){
        @Override
public Alimento mapRow(ResultSet rs, int rowNum) throws SQLException{
                Alimento a = new Alimento();
a.setCodalimento(rs.getInt("codalimento"));
a.setCusto(rs.getString("custo"));
a.setTipoalimento(rs.getString("tipoalimento"));
return a;
}
});
return obj;
}
```

Quadro 44 - Pesquisa com query Spring
Fonte - Autoria própria

4.4.3 Especificação JPA e JDO

No caso das especificações, este trabalho não verificou a razão pela qual o *framework* não utiliza as especificações JPA e JDO sem estar integrado a outro *framework*. Assim como no caso de usar as especificações, o Spring não tem um provedor de persistência e precisa usar a ajuda de outro *framework* para isto, então não é considerado algo que venha somente do Spring.

Em conceitos gerais para se usar a persistência com a especificação JPA basta seguir o tópico 3.2.2 e para outra especificação que no caso seria a JDO, basta seguir o tópico 3.2.1.

4.5 PROJETO UTILIZANDO DATANUCLEUS

Este *framework* tem o suporte a ambas as especificações que seriam a JPA e JDO. Em geral, para realização de persistência com as especificações basta seguir o que já foi dito nos tópicos 3.2.1, JDO e 3.2.2, JPA respectivamente.

Mas antes foi disposto o que difere de outros *frameworks* no seu processo de utilização no desenvolvimento de aplicativos. Os seguintes itens podem ser vistos como um diferencial na hora de se utilizar esse *framework*:

- Realização do processo de *Enhanced* (melhoramento) das classes utilizadas para representar as entidades do banco de dados. Algo que é considerado obrigatório e não pode evitar;
- Possibilidade de construção do projeto com *Maven*, isto é um opcional, os desenvolvedores do *framework* até trazem algumas facilidades quando o DataNucleus é utilizado com projetos em *Maven*, facilidades dentre as quais pode-se citar um maior suporte às especificações e um maior número de instruções que podem ser encontrados de como se criar um projeto em *Maven*.

Devido às instruções do *framework* serem voltadas mais a projetos com *Maven*, ele foi escolhido até por que foi tentado fazer o projeto *desktop* normal, porém, o processo de *Enhanced* é mais complicado para ser realizado desta forma.

4.5.1 O que Diferenciou dos Outros?

Para a geração de incremento para a chave primária quando está se utilizando do mesmo banco, os *frameworks* acompanham o mesmo padrão, sendo necessário escolher qual banco vai usar e escolher entre os tópicos 4.1.2 para MySQL ou 4.1.3 para Firebird quando está se utilizando de JPA.

Entre outras coisas também foi encontrado a possibilidade de se fazer projetos em *Maven*, claro que isto em si é possível para qualquer um dos

frameworks, porém, neste, a sua documentação contribui consideravelmente para que os possíveis novos utilizadores dele venham a utilizá-lo com *Maven*.

Devido ao Windows só suportar passagens pequenas de caminhos com parâmetros, foi colocado dentro da tag *configuration* do *Maven* (*maven-datanucleus-plugin*) a *tag<fork>false</fork>* que possibilita contornar o impedimento do Windows em assumir caminhos grandes.

4.5.2 Utilizando Maven em um Projeto

Para iniciar um projeto com *Maven* dentro do Net*Beans*, deve ir em Arquivo, escolher a opção Novo projeto, dentro deste aparecerão as opções referentes aos projetos disponíveis, então escolha a categoria *Maven*, e após Aplicativo Java como pode ser visto.

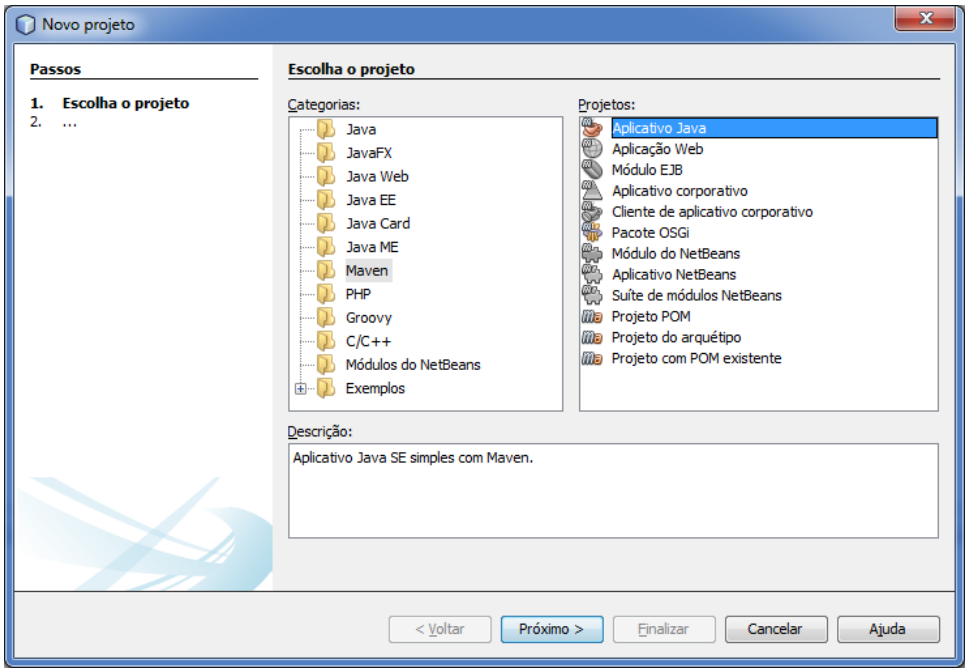

Figura 25 - Novo projeto com *Maven*
Fonte – Autoria própria

Ao clicar em próximo na figura 25, a próxima janela que seria a figura 26 mostrará opções referentes:

- Nome do projeto;
- Localização do projeto;
- Diretório do projeto;
- Id de artefato, Id de grupo, Versão (propriedades para projetos *Maven*);

- Pacote, que será algum possível pacote padrão criado.

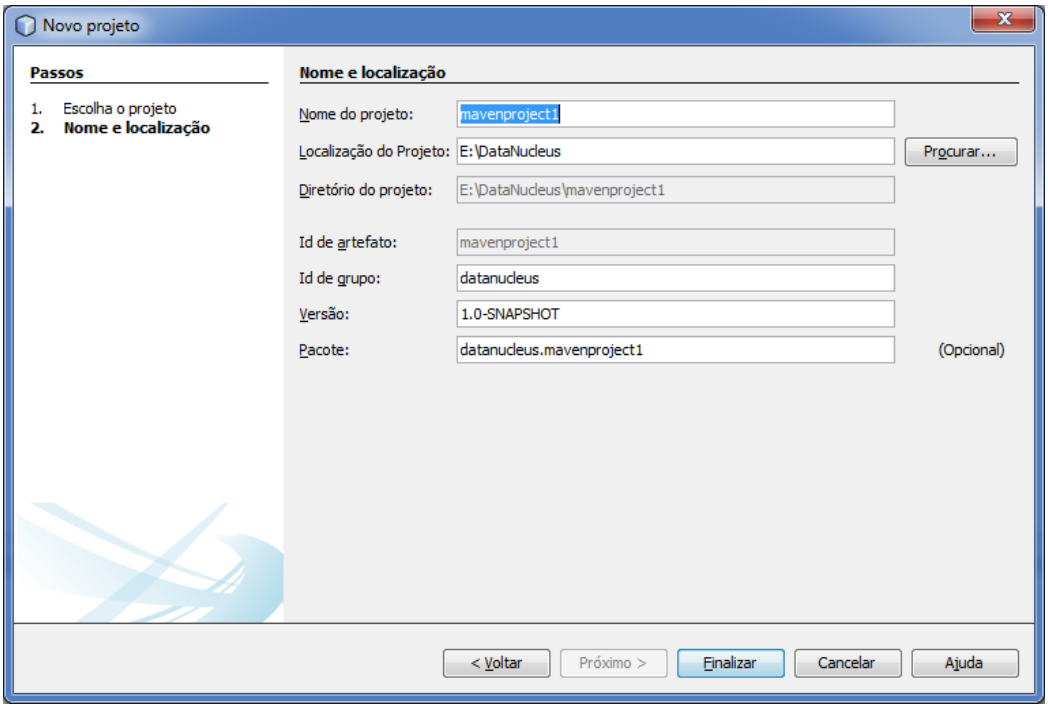

Figura 26 - Nome e localização projeto *Maven*
Fonte – Autoria própria

A estrutura referente a esse tipo de projeto fica diferente do que é feito para projeto *desktop* Java. A figura 27 mostra algumas pastas que antes não continham no projeto *desktop*, que são elas:

- Dependências que é a mesma coisa do que o pacote de bibliotecas, mas aqui as dependências podem ser baixadas da internet;
- Dependência Java, é a JDK que antes ficava dentro do pacote de bibliotecas;
- Arquivos de projeto contém o feito em XML chamado pom, que é por onde se adiciona novas dependências, ou seja, novas bibliotecas no projeto, mas claro, não pode se esquecer de colocar seus devidos repositórios.

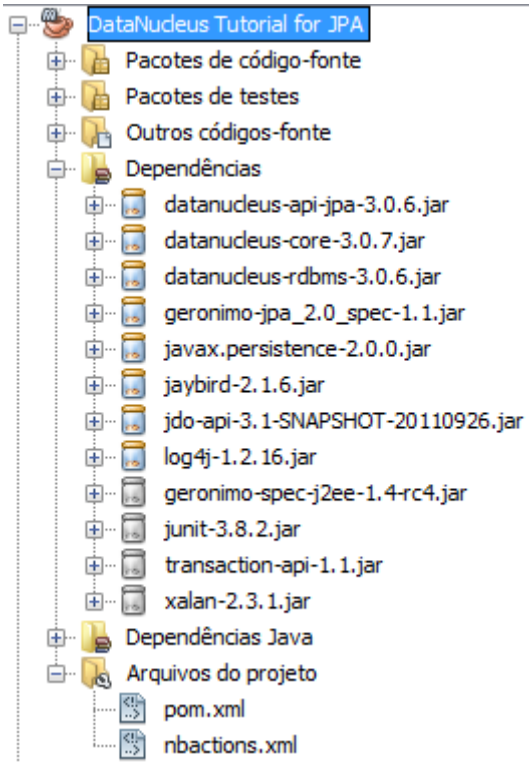

Figura 27 - Estrutura projeto *Maven*
Fonte – Autoria própria

Quanto a adicionar novas dependências dentro de um POM, recomenda-se seguir a documentação do *framework* e sempre observando se é a mesma especificação utilizada que se está vendo no tópico. De maneira geral, o NetBeans tem uma facilidade quanto a adição de novas dependências, mas nesse caso, você deve ter certeza que ela será utilizada, pois o quanto mais dependências você adicionar, consequentemente maior ficará seu projeto final.

Então para adicionar uma nova dependência, você deve ir com o botão direito em Dependências e logo após aparecerá à figura 28, onde você irá consultar qual o nome da dependência e se ela estiver disponível dentro dos repositórios disponíveis, então poderá ser baixada.

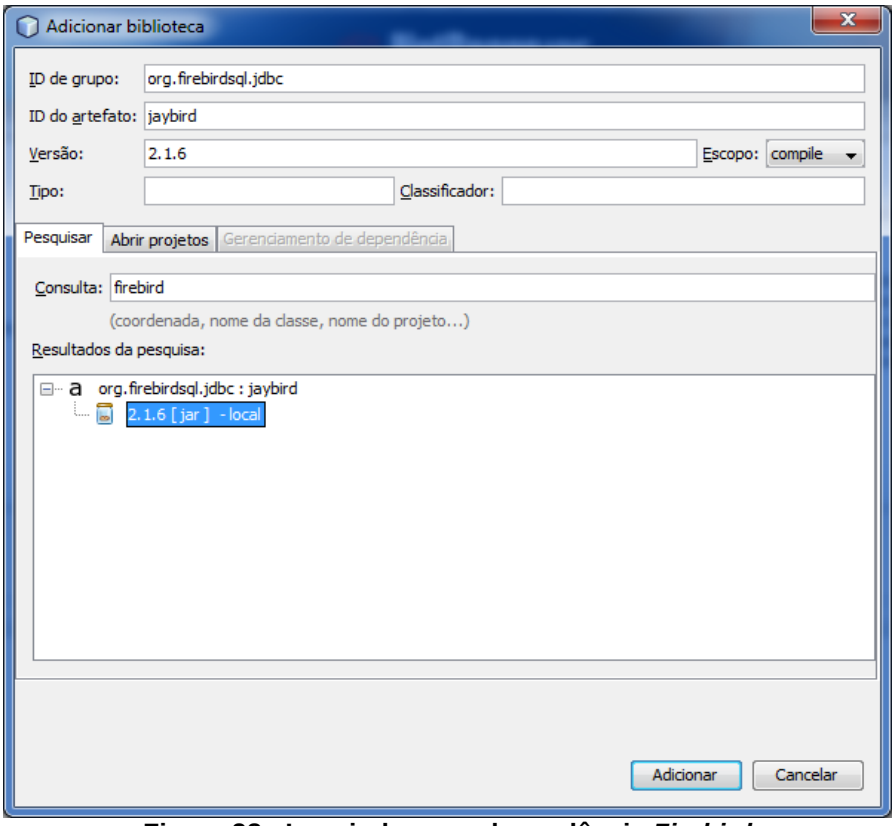

Figura 28 - Inserindo nova dependência *Firebird*
Fonte - Autoria própria

4.5.3 Especificação JPA

Para realização de persistência de dados utilizando essa especificação, basta seguir o que foi dito no tópico 4.2.2 onde poderá haver algumas mudanças, porém, essas são de um *framework* para outro. Para saber o que muda no desenvolvimento de aplicativos com esse *framework*, fizemos o tópico “O que Diferenciou dos Outros?”. E não se esquecendo também, que precisamos de bibliotecas específicas para o desenvolvimento com essa especificação.

Como o projeto DataNucleus é feito com *Maven*, então basta criar o arquivo pom.xml com suas devidas dependências que elas serão baixadas da internet.

4.5.4 Especificação JDO

Para realização de mapeamentos temos duas opções que são: Anotações feitas diretamente dentro da classe Java. utilizada para representar a entidade de banco de dados e a outra seria via XML. Os tipos de anotações que são possíveis

usando essa especificação, são diferentes do que foi utilizado nos outros projetos de *frameworks*.

4.5.4.1 Conexão

Para realizar a conexão com essa especificação, foi usado um arquivo de propriedades o qual será chamado para obter os dados nele especificados e se conectar ao banco de dados. As propriedades definidas dentro do arquivo de propriedade foram estipuladas de acordo com o Quadro 5.

Propriedade	Valor
javax.jdo.option.Connection*Driver*Name	org.firebirdsql.jdbc.FB*Driver*
javax.jdo.option.ConnectionURL	jdbc:firebirdsql://localhost:3050/c:/Bancos/PECUARISTA.FDB
datanucleus.ConnectionPassword	Masterkey
datanucleus.ConnectionUserName	Sysdba

Quadro 45 - Arquivo de propriedades DataNucleus JDO
Fonte - Autoria própria

4.5.4.2 Criação de POJO

Para criação de novas classes de entidades Java que representam tabelas do banco de dados, foi utilizado o mesmo processo feito para o tópico de “Criação de POJO – JPA”, exceto na parte onde teremos as anotações devido à especificação JDO não realizar anotações da mesma maneira que a JPA.

Na realização de anotações dentro da classe Java gerada, será feito através de tipos de anotações específicas para JDO:

- *@PersistenceCapable*

  Anotação responsável por dizer que a classe vai representar uma entidade no banco de dados.

- @PrimaryKey

  Anotação responsável por dizer que o atributo em questão a que ele estiver representando é uma chave primária dentro do banco.

- @Persistent(valueStrategy = IdGeneratorStrategy.SEQUENCE, sequence="GEN_ALIMENTO_ID")

Anotação responsável pela geração de incremento na chave primária da tabela a que representa a classe Java.

4.5.4.3 Métodos de persistência

Os métodos para persistir dados foram divididos entre alteração de dados que compõe (inserir, atualizar, e excluir) o dado no banco. E para consulta foi especificado cada método que é possível ser utilizado para isso.

Antes de se realizar a persistência de dados deve-se declarar as classes que nos auxiliaram nesse trabalho, para isso separamos em dois métodos que são um para conectar, e outro para desconectar que como próprio nome diz é voltado para conectar e o outro em desconectar o programa do banco.

- Método conectar

Método

public void conectar(){
pmf = JDOHelper.getPersistenceManagerFactory("datanucleus.properties");
pm = pmf.getPersistenceManager();
tx = pm.currentTransaction();
tx.begin();
}

Quadro 46 - Método conectar DataNucleus JDO
Fonte - Autoria própria

- Método desconectar

Método

public void desconectar(){
pm.flush();
tx.commit();
pm.close();
pmf.close();
}

Quadro 47 - Método desconectar DataNucleus JDO
Fonte - Autoria própria

4.5.4.3.1 Métodos de alteração de dados

Aqui estão envolvidos os métodos que são feitos para alterar os dados do banco.

- Inserir

Método

public String inserirObjeto(Alimento a){
try{
conectar();
pm.makePersistent(a);
desconectar();
resultado = "Objeto inserido com sucesso";
}catch(Exception e){
resultado = "Erro ao inserir objeto:\n" + e;
}
return resultado;
}

Quadro 48 - Inserir com DataNucleus JDO
Fonte - Autoria própria

- Atualizar

O método utilizado para atualizar os dados dentro do banco é quase o mesmo de se inserir, ou seja, para atualizar basta chamar novamente o método de inserir um dado, só que claro definindo o código (chave primária) para aquele que quer atualizar. O método de atualizar pode ser visto no Quadro 49.

Método

public String atualizarObjeto(Alimento a){
try{
conectar();
Alimento a2 = (Alimento)pm.getObjectById(Alimento.class, a.getCodalimento());
a2.setCusto(a.getCusto());
a2.setTipoalimento(a.getTipoalimento());
pm.makePersistent(a2);
desconectar();
resultado = "Objeto atualizado com sucesso";
}catch(Exception e){
resultado = "Erro ao atualizar objeto:\n" + e;
}
return resultado;
}

Quadro 49 - Atualizar com DataNucleus JDO
Fonte - Autoria própria

- Excluir: exige é claro que o objeto a ser excluído esteja com todos os seus atributos não nulos, ou seja, ele esteja devidamente preenchido.

Método

public String excluirObjeto(Alimento a){
VacaPers vp = new VacaPers();
List<Vaca> obj = vp.procuraVacaAlimento(a.getCodalimento());
if(obj != null && !obj.isEmpty()){
resultado = "Alimento sendo usado para alguma vaca";
}else{
try{
conectar();
a = (Alimento)pm.getObjectById(Alimento.class, a.getCodalimento());
pm.deletePersistent(a);
desconectar();
a.setCodalimento(null);
resultado = "Objeto excluido com sucesso";
}catch(Exception e){
resultado = "Erro ao excluir objeto:\n" + e;
}
}
return resultado;
}

Quadro 50 - Método excluir DataNucleus JDO
Fonte - Autoria própria

4.5.4.3.2 Métodos de consulta de dados

Dentre as pesquisas que existem por código pode realizar as seguintes:

- Pesquisa por chave primária

Para realizar esse tipo de pesquisa deve-se passar por parâmetro dentro de *getObjectById* a classe a qual representa a tabela que está sendo pesquisada, e o campo que deve ser obrigatoriamente chave primária da tabela.

Método

public Alimento procuraCodigo(int codigo){
conectar();
Alimento a = (Alimento)pm.getObjectById(Alimento.class, codigo);
return a;
}

Quadro 51 - Pesquisa por chave primária JDO
Fonte - Autoria própria

- Pesquisa utilizando SQL

Método

public List<Alimento> procuraNomeParcial(String nome){
String sql = "select * from alimento where tipoalimento like '%" + nome + "%'";
conectar();
Query q = pm.newQuery("javax.jdo.query.SQL",sql);
q.setClass(Alimento.class);
List<Alimento> obj = (List<Alimento>)q.execute();
return obj;
}

Quadro 52 - Pesquisa com SQL no JDO
Fonte - Autoria própria

- Pesquisa utilizando JDOQL

Método

public Alimento procuraNomeExata(String tipo){
Alimento a= new Alimento();
conectar();
Query q = pm.newQuery(Alimento.class);
q.setFilter("tipoalimento == '" + tipo + "' ");
List<Alimento> obj = (List<Alimento>)q.execute();
Iterator it= obj.iterator();
while(it.hasNext()){
a = (Alimento)it.next();
}
return a;
}

Quadro 53 - Pesquisa com JDOQL no JDO
Fonte - Autoria própria

- Pesquisa utilizando JPQL

Método

public Bezerro procuraNomeExata(String nome){
Bezerro b= new Bezerro();
conectar();
Query q = pm.newQuery("javax.jdo.query.JPQL", "SELECT b FROM Bezerro b WHERE " +
"nome = '" + nome + "'");
q.setClass(Bezerro.class);
List<Bezerro> obj = (List<Bezerro>)q.execute();
Iterator it = obj.iterator();
while(it.hasNext()){
b = (Bezerro)it.next();
}
return b;
}

Quadro 54 - Pesquisa com JPQL no JDO
Fonte - Autoria própria

5 RESULTADOS DOS *FRAMEWORKS*

Este capítulo apresenta os critérios de avaliação, caracterizando os *frameworks* em pontos fortes e fracos, para facilitar a escolha de um determinado *framework* de persistência considerando suas características.

5.1 AVALIAÇÃO *FRAMEWORK* HIBERNATE

5.1.1 Pontos Fortes

- Possuir API própria, o que possibilita ao *framework* ter uma adição de recursos, fornecendo um ambiente mais adequado de desenvolvimento para o programador;
- Utilização da especificação JPA, sendo esta genérica para a comunidade de programadores, o quê significa que se muitos usuários a utilizam torna a adaptação do *framework* a ela mais fácil;
- Criação das classes de POJO tendo como base o banco de dados, faz com que o programador não precise fazer a criação manual de classes POJO;
- Desenvolvimento em várias plataformas que permite ao programador que o utilizar para o desenvolvimento de um sistema que manipulem banco de dados, o possa utilizar para desenvolvimento web, *desktop*, entre outros;
- Código aberto, o que traz a possibilidade de adaptá-lo, acrescentando novos módulos para atender as necessidades especificadas de um determinado usuário, sem precisar de outras ferramentas;
- Fácil adaptação a novas conexões, pois este *framework* possibilita a instalação de novos *drivers* relacionados ao acesso de diferentes bancos de dados e, em não existindo um *driver* para conexão com o banco de dados o usuário poderá criá-lo.

5.1.2 Pontos Fracos

- Suporte a só uma especificação de persistência que é a JPA.

5.2 AVALIAÇÃO *FRAMEWORK*TOPLINK

5.2.1 Pontos Fortes

- Suporte a especificação JPA;
- Suporte ao banco de dados MySQL;
- Possui linguagem de manipulação OO.

5.2.2 Pontos Fracos

- Só aceita a especificação JPA para realização de suas persistências;
- Não tem suporte a um dos bancos estudados que seria o Firebird.

5.3 AVALIAÇÃO *FRAMEWORK* SPRING

5.3.1 Pontos Fortes

O Spring tem suporte às especificações JPA e JDO: este tipo de suporte é feito de maneira a poder utilizar o Spring juntamente com uma ou outra.

5.3.2 Pontos Fracos

O *framework* Spring não ter um provedor de persistência, elemento necessário para que seja feita a persistência dos objetos. Sendo assim, ele somente pode utilizar as especificações JPA e JDO através de outro *framework*.

5.4 AVALIAÇÃO *FRAMEWORK* DATANUCLEUS

5.4.1 Pontos Fortes

- Utiliza as especificações JPA e JDO para persistir seus dados. Isso é uma característica relevante do *framework*, pois as especificações de persistência são padrões utilizados para montagem da estrutura da persistência de cada *framework*, com isso pode-se ter diferentes *frameworks* mas que utilizam o mesmo padrão para persistência e o que traz facilidade na adaptação aos programadores.

5.4.2 Pontos Fracos

- Exigência de *Enhanced* nas classes entidade que representam as tabelas do banco de dados, o que torna o projeto mais trabalhoso. Esse procedimento para quem desconhece pode tornar a realização de um novo aplicativo bem difícil, pois exige um conhecimento em como configurar o DataNucleus de maneira específica para a especificação que esteja usando.

5.5 AVALIAÇÃO DAS ESPECIFICAÇÕES DE PERSISTÊNCIA

5.5.1 Pontos Fortes

- JDO tem suporte a vários tipos de bancos;
- Ambas têm facilidades com comandos de consulta e comandos de persistência.

5.5.2 Pontos Fracos

JPA tem suporte a um número menor de banco de dados quando comparado a JDO.

5.6 CONSIDERAÇÕES SOBRE O CAPÍTULO

O quadro 6 resume as principais características dos *frameworks* analisados no desenvolvimento de um aplicativo exemplo, tais características são: se o *framework* segue uma especificação ou não, se há integração com os SGBD' s MySQL e FireBird, a possível geração das classes de entidade (POJO) através das tabelas mapeadas do banco, e por último a possibilidade de o *framework* ser integrador a IDE de desenvolvimento.

	DataNucleus	**Hibernate**	**Spring**	**TopLink**
Especificação JPA	SIM	SIM	SIM(*)	SIM
Especificação JDO	SIM	NÃO	SIM(*)	NÃO
MySQL	SIM	SIM	SIM	SIM
FireBird	SIM	SIM	SIM	NÃO
Geração do POJO	NÃO	SIM	NÃO	NÃO
Geração de tabelas	SIM	SIM	NÃO	SIM
Integração a IDE	NÃO	SIM	SIM	SIM

Quadro 55 - Divergência entre os frameworks
Fonte - Autoria própria

Vale lembrar que o Spring somente consegue utilizar as especificações JPA e JDO através de outro *framework* que será o seu provedor de persistência. Quanto à realização de mapeamentos utilizando somente o Spring, isso só é possível, por meio de sua integração com outro *framework*, o qual se encarregará do papel de mapear as tabelas do banco de dados.

A geração de POJO por meio das tabelas é realizada somente pelo Hibernate, os demais *frameworks* analisados o fazem através do módulo de persistência do NetBeans quando está usando a especificação JPA.

Quando se fala em Módulo do NetBeans, refere-se aos arquivos da IDE que podem ser escolhidos através da opção novo arquivo e na categoria persistência, que possibilitam a realização da persistência de dados, esse módulo é voltado para especificação JPA, pois uma de suas bibliotecas a ejb-persistence.jar é o primeiro arquivo para utilizar quando se insere uma unidade de persistência. Constatou-se que as bibliotecas que têm essa identificação são necessárias para persistir dados com a utilização da especificação JPA.

Ainda ao invés de gerar o POJO, tem a possibilidade de que os *frameworks* gerem as tabelas dentro do banco de dados através de classes POJO devidamente definidas e mapeadas. A essa possibilidade de gerar tabela através de classe de entidade POJO, se encontram os seguintes *frameworks*: DataNucleus, Hibernate e TopLink.

O *framework* Spring não aparece na relação devido ao mesmo não gerar sequer o mapeamento das tabelas, ou seja, sem o auxílio de outro *framework*, o que seria necessário para geração dessas tabelas. Em função do Spring ser voltado para o desenvolvimento Web, ele tem uma implementação com *beans* que são classes que se utilizam da persistência de dados e essas são mapeadas obrigatoriamente dentro do arquivo de configuração do *framework*, sendo que se essas classes não estiverem mapeadas, a persistência de dados não funcionará efetivamente.

No caso de usar o Spring dentro de um desenvolvimento em três camadas (Visão, Negócio e Persistência), as classes de persistência devem ser declaradas dentro do arquivo de configuração sendo chamadas de *bean*.

Quanto à integração dos *frameworks* a IDE NetBeans o Hibernate, Spring e TopLink já vêm integrado na primeira instalação da IDE, sendo que somente o DataNucleus não tem nenhum tipo de integração com esta IDE, mas pode ser facilmente integrado com o Eclipse, tendo no *site* da organização que o desenvolve um *plugin* criado para essa integração.

No suporte a banco de dados, o DataNucleus se mostra superior devido a oferecer um suporte a um grande número de bancos quando está se usando a especificação JDO. Quando se precisam manipular dados, o Hibernate se mostra superior na recuperação de dados com associações, pois segundo Campos (2010), ele tem um maior número de opções para otimização de carregamento.

Já nas consultas, o *framework* Hibernate tem a facilidade de usar a API para evitar ter que escrever um comando inteiro SQL e, há a possibilidade de utilizar uma linguagem de manipulação chamada HQL que é voltada a orientação a objetos. Quando está utilizando o *framework* DataNucleus, ele pode fazer consultas através de linguagens com suporte a OO, ou utilizando SQL.

O Spring utiliza comandos SQL puros, que pode ser considerado por alguns programadores acostumados com comandos SQL mais fácil de ser aprendida, portanto o *framework* Spring não tem nenhuma linguagem de manipulação com suporte a OO.

Ao utilizar o TopLink, as consultas podem ser feitas através da especificação JPA, onde esta permite a utilização da linguagem SQL, linguagem orientada a objetos e com a sua API que permite algumas facilidades com as consultas, o que também é possível quando o *framework* tem suporte a especificação JDO, contudo por serem especificações diferentes, as facilidades também serão diferenciadas.

A tabela 1 de análise de tempo leva em conta em milissegundo dos métodos (inserir, atualizar, excluir, procurar e a conexão ao banco). Os tempos foram conseguidos realizando a análise da implementação de cada um dos *frameworks* através do *profiler* que contém dentro da IDE NetBeans. Para a busca foi utilizado o método procurar por código. Os tempos foram contados a partir da camada de persistência.

O ambiente utilizado para a realização dos testes de desempenho dos *frameworks* foi um Pentium i5 2,67 GHz de segunda geração, sendo a memória de 4GB e o sistema operacional o Windows 7- *Ultimate* de 64 bits. O SGBD encontrava-se instalado localmente na máquina *desktop,* onde tanto o programa quanto o banco estavam na mesma máquina. Os testes foram realizados dez vezes em cada *framework,* sendo feito posteriormente a média na tentativa de eliminar qualquer interferência no cálculo do desempenho dos *frameworks* analisados neste trabalho.

Tabela 1 - Análise de desempenho dos *frameworks*

Ferramenta	Inserir	Atualizar	Excluir	Procurar	Conectar
Hibernate	6,9202	3,9657	16,5426	39,0198	5062,204
HibernateJPA	40,0897	12,4146	13,36	102,8479	10379,1
DataNucleusJDO	239,4634	44,2583	75,602	205,416	4533,158
DataNucleusJPA	231,9455	57,6658	43,3433	121,6155	5007,538
SpringJdbc	109,6537	126,7622	199,2361	542,6224	1377,913
TopLinkJPA	0,8603	3,803	6,2044	43,7936	3018,057

Fonte - Autoria própria

Ao fazer a análise da Tabela 1, Hibernate teve o menor tempo para procurar, e levou tempos razoáveis para inserir, atualizar, e excluir. Já o que teve menor tempo para realizar a conexão com o banco de dados foi o *framework* Spring e para inserir e excluir obteve tempos expressivos. Para o método procurar ele obteve o maior tempo e também o maior tempo com relação ao método atualizar.

No *framework* TopLink com JPA, obteve os menores tempo com relação aos métodos de persistência inserir, atualizar e excluir, sendo o segundo menor tempo

com relação a conexão ao banco. Quando se fala do *framework* DataNucleus, as duas especificações obtiverem um tempo expressivo quanto a inserir novos dados e um tempo médio quanto a conexão ao banco.

Em geral, pode-se dizer que com relação aos tempos Hibernate utilizando a API e TopLink com JPA obtiveram os melhores tempos, sendo o TopLink melhor nos tempos com relação aos métodos de inserir, atualizar, excluir e ainda com a conexão ao banco de dados em comparação a conexão do Hibernate sendo ela com a API ou com a especificação JPA.

Para ter uma comparação à velocidade de um *framework* e um projeto sem a sua utilização, ou seja, utilizando comandos em linguagem SQL, ou ainda, com *procedures* no banco.

Para esta comparação foi realizado um segundo projeto, com a utilização da mesma tabela que foi usada para primeira comparação de tempos dos *frameworks,* assim não tendo qualquer divergência de tempo com relação à estrutura de tabelas diferentes. A tabela 2 demonstra a relação de tempos de análise de tempos.

Tabela 2 - Tempo para Projeto sem *Framework*

Método	Tempo
Inserir com SQL na aplicação	13,3
Inserir com Procedure	19,3
Atualizar com SQL na aplicação	10,8
Atualizar com Procedure	10,7
Excluir com SQL na aplicação	10,3
Excluir com Procedure	10,6
Procurar	136
Conectar	476

Fonte - Autoria própria

Nota-se na tabela 2 que para persistir dados sem a utilização de um *framework* os tempos são consideravelmente menores, mais se colocando em consideração a tabela 1, o *framework* Hibernate com a utilização da API e o TopLink tem ainda assim tempos menores, e devem ser levados em consideração no desenvolvimento de novos programas.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este capítulo apresenta as conclusões obtidas em relação aos diferentes *frameworks* estudados para realizar uma análise comparativa. É apresentada também a seção 6.2 de futuros trabalhos acadêmicos que venham a complementar esse estudo.

6.1 CONCLUSÃO

Ao analisar os *frameworks:* Hibernate, DataNucleus, TopLink e Spring, foi possível verificar e compreender como cada um pode ser implementado, apresentando as diferenças na utilização de cada um.

Na manipulação de dados, ou seja, (incluir, alterar e excluir) o Hibernate tem a API que colabora no processo de persistir dados, mas não é possível deixar de mencionar os outros *frameworks* que envolvem as especificações JPA e JDO, pois as mesmas quando utilizadas, embora sejam diferentes da API do Hibernate, também facilitam muito a persistência.

Quando se trata de realizar consultas a utilização de algum tipo de API é considerado um benefício, pois as mesmas facilitam a construção dessas pesquisas ao invés de se realizar tediosos comandos SQL que podem se tornar muito complexos. As API's que são encontradas no trabalho, são do Hibernate e das especificações JPA e JDO.

O desenvolvedor que utiliza ambas as especificações JPA e JDO, poderia utilizar os quatro *frameworks,* pois eles têm suporte a pelo menos uma das especificações.

Os conceitos de herança e polimorfismo oriundos da Orientação a Objetos, aliados aos *frameworks* como Hibernate, Spring, TopLink e DataNucleus, propiciaram uma camada de persistência com código fonte conciso, claro e testado. Dessa forma, ocorre redução nos custos envolvidos na manutenção do código e para adição de novas funcionalidades, além de atingir altos níveis de reusabilidade.

6.2 TRABALHOS FUTUROS

Quanto aos trabalhos futuros, a proposta é considerar todas as características estudadas neste trabalho dos *frameworks* de persistência, para construir um novo *framework*, o qual trate as restrições apresentadas nos *frameworks*, objetivos dessa pesquisa.

REFERÊNCIAS

ADAMATTI, Marcelo Pittigliani. **FUMIGANT: Gerador de Código Java a Partir de Base de Dados.** Gravataí, 2006. Disponível em:<http://fumigant.googlecode.com/svn/trunk/docs/monografia/trabalho.pdf> Acesso em: 20 out. 2011.

BARRETO, C.G., FUKS, H.; LUCENA, C.J.P. **Agregando *Frameworks* em uma Arquitetura Baseada em Componentes: Um Estudo de Caso no Ambiente AulaNet**, Anais do 5º Workshop de Desenvolvimento Baseado em Componentes - WDBC 2005, 07-09 de Novembro, Juiz de Fora-MG, ISBN 85-88279-47-9, pp. 25-32. 2005. Disponível em:<http://groupware.les.inf.puc-rio.br/public/papers/2005.WDBC.Barreto.*Frameworks*.pdf > Acesso em: 20 out. 2011.

BARROS, Bruno Alberth Silva; BARROS. Raul Silva; CORTES, Omar AndresArmona. **UM ESTUDO COMPARATIVO ENTRE APIS DE PERSISTÊNCIA UTILIZANDO GRANDES VOLUMES DE DADOS.** CONNEPI, 2009. Disponível em:<connepi2009.ifpa.edu.br/connepi-anais/artigos/101_2304_1927.pdf> Acesso em: 24 out. 2011.

BAUER, Christian; KING, Gavin. **Hibernate em ação**. Rio de Janeiro: Ciência Moderna, 2005. 532 p. ISBN 8573934042

BOOCH, Grady. **UML Guia do Usuário.** O mais avançado tutorial sobre Unified Modeling Language (UML), Rio de Janeiro: Campus, 2000.

CAMPOS, Raphael Palhares de. **Análise Comparativa de *Frameworks* de Persistência:** UFLA, LAVRAS-MG 2010. Disponível em:< http://www.bcc.ufla.br/monografias/2010/RAPHAEL_BARRETO_PALHARES_DE_CAMPOS.pdf> Acesso em: 24 out. 2011.

DATANUCLEUS**. DATANUCLEUS AccessPlatform 3.0**. Disponível em:<http://www.DATANUCLEUS.org/products/accessplatform_3_0/index.html> Acessado em: 06 nov. 2011.

ELMASRI, Ramez; NAVATHE, Shamkaut B. **Sistemas de Bancos de Dados.** São Paulo, SP: Person, 2005.

FERNANDES, Raphaela Galhardo; LIMA, Gleydson de A. Ferreira. **Hibernate com Anotações.** Natal, 2007.

GALANTE, Alan Carvalho; MOREIRA, Elvis Leonardo Rangel; BRANDÃO, Flávio Camilo. **BANCO DE DADOS ORIENTADO A OBJETOS: UMA REALIDADE**: FSMA. Disponível em <http://www.fsma.edu.br/si/edicao3/banco_de_dados_orientado_a_objetos.pdf> Acesso em 08 out. 2011.

GAMMA, Erich et al. Padrões de projeto: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2002.

GUIMARÃES, Célio Cardoso. **Fundamentos de bancos de dados.** Campinas, SP: UNICAMP, 2003.

HIBERNATE.**Capitulo 2. Arquitetura.** Disponível em:< http://docs.jboss.org/hibernate/core/3.5/reference/pt-BR/html/architecture.html> Acessado em: 16 de dez de 2011.

HIBERNATE**. Hibernate Core Reference Manual**. Disponível em:<http://docs.jboss.org/hibernate/core/3.6/reference/pt-BR/pdf/hibernate_reference.pdf> Acessado em: 01 de novembro de 2011.

KING, Gavin; BAUER, Christian; ANDERSEN, Max Rydahl; BERNARD, Emmanuel; EBERSOLE, Steve; FERENTSCHIK, Hardy. **Documentação de Referência Hibernate,** 2011.

KORTH, Henry F.; SILBERSCHATZ,Abraham.**Sistemas de Banco de Dados**.2ª. ed., São Paulo, MAKRON Books, 1995.

KRAEMER, Fabiano; VOGT, Jerônimo Jardel. Trabalho de conclusão da disciplina Programação com Objetos Distribuídos. **Hibernate, um Robusto *Framework* de Persistência Objeto-Relacional.** Porto Alegre, 2005.

LIMA, Adilson da Silva. **UML 2.2: Do Requisito à Solução**. 4ª Edição Revisada e Atualizada. São Paulo, 2009.

LIMA, Rodolfo Schulz. **CPPObjects: Biblioteca de Mapeamento Objeto-Relacional em C++.** Rio de Janeiro, RJ:UFRJ, 2008. Disponível em:< http://monografias.poli.ufrj.br/monografias/monopoli10003014.pdf> Acesso em: 08 out. 2011.

MATOS, Simone Nasser. **Um Método Dirigido por Responsabilidades para Obtenção Antecipada de Pontos de Estabilidade e de Flexibilidade no Envolvimento de *Frameworks* de Domínio**. São José dos Campos, SP, 2008.

MELLO, Mauricio. **O Modelo Entidade-Relacionamento (MER):** PUCPR. Disponível em:< http://www.las.pucpr.br/mcfmello/BD/BD-Aula02-MER.pdf> Acesso em: 08 out 2011.

ORACLE. Disponível em:< http://www.oracle.com/technetwork/java/jndi/index.html> Acesso em: 14 dez. 2011.

ORACLE. **Toplink Concepts.** Disponível em:< http://download.oracle.com/docs/cd/A97688_16/toplink.903/b10061/concepts.htm> Acesso em: 24 out. 2011.

PEREIRA, Adrovando Inocêncio Universidade Tecnológica Federal do Paraná (Câmpus Cornélio Procópio). **Principais padrões de persistência do Java: um estudo comparativo.** 2009. 50 f.: Monografia (Especialização) - Universidade Tecnológica Federal do Paraná. Curso de Especialização em Tecnologia Java, Cornélio Procópio 2009.

PINHEIRO, J. F. V.; **Um Framework para Persistência de objetos em Banco de Dados Relacionais**; Msc. Thesis; Universidade Federal Fluminense, 2005.

PIRES, Ivan. **Modelagem de Dados Usando o Modelo Entidade-Relacionamento**: UNEMAT. Disponível em <http://www2.unemat.br/~ivanpires/files/dwl/bd/slides/bd_4.pdf> Acesso em 08 out 2011.

SANTOS, Thiago Roberto dos**. Análise e Comparação de *Frameworks* para Desenvolvimento Web em Java,** Florianópolis – SC, 2007. Disponível em:<http://projetos.inf.ufsc.br/arquivos_projetos/projeto_669/TCC-ThiagoRobertoSantos-final.pdf> Acesso em: 20 out. 2011.

SHAEFER, Henrique; ISAIA, Ferreira Victor. **Desenvolvimento das Camadas Lógicas e Persistência do Projeto SIAP-F:** UFSANTA CATARINA. Disponível em:< http://projetos.inf.ufsc.br/arquivos_projetos/projeto_316/artigo_tcc.pdf> Acesso em: 13 dez. 2011.

SPRING. **Reference Documentation**. Disponível em:<http://static.springsource.org/spring/docs/3.1.x/spring-*framework*-reference/pdf/spring-*framework*-reference.pdf> Acesso em: 26 de dez. de 2011.

TOPLINK. **Introduction to TopLink** . Disponível em:<http://docs.oracle.com/cd/E14571_01/web.1111/b32441/undtl.htm#CHDEABHC> Acessado em: 26 de Jan. de 2012.

TRANCOSO, Leonardo Rocha; PEREIRA, Luiz Felipe Garcia.**Nellorin: Aplicação Web para Entretenimento em Grupo.** Curitiba, 2009**.** Disponível em:<http://www.dainf.cefetpr.br/~cristina/Nellorin.pdf> Acesso em: 24 outubro 2011.

DALLACQUA, Vinicius Teixeira.**Persistência de Dados em Java com JPA e Toplink.**Instituto Federal de Educação, Ciência e Tecnologia - IFTO, 2009**.** Disponível em:<http://www.clebertoledo.com.br/blogs/tecnologia/administracao/files/files/Persistencia_de_Dados_em_Java_com_JPA_e_Toplink.pdf> Acesso em: 24 outubro 2011.

WALLS, Craig; BREIDENBACH, Ryan. **Spring Em Ação.** Rio de Janeiro: Editora Ciência Moderna Ltda, 2006.

WILLEMANN, David Pedro; IBARRA, Gustavo Bestetti. **Trabaho de Conclusão de Curso *Framework* Java de Apoio ao Desenvolvimento de Aplicações Web com Banco de Dados, utilizando Struts, Tiles e Hibernate**: Universidade Federal de Santa Catarina, 2007.

APÊNDICE A – *SCRIPT*S DE GERAÇÃO DOS BANCOS

_Script_s para geração do banco de dados Firebird

O script foi feito de acordo com a ferramenta BrModelo que, de acordo com o modelo relacional que já tinha sido convertido com ajuda da ferramenta para scripts que podem ser usados no banco. Ainda para o Firebird foi criado *generator*, que se consegue através do comando:

CREATE generator gen_alimento_id;

O script para geração:

```
-- Geração de Modelo físico
-- Sql ANSI 2003 - brModelo.

CREATE TABLE Vaca (
CodVaca Integer PRIMARY KEY,
CodAlimento Integer,
CodFrigorifico Integer,
DatAbate Varchar(40)
);
CREATE TABLE Vacinacao (
DatVacinacao Varchar(40),
CodVaca Integer,
CodVacina Integer,
PRIMARY KEY (DatVacinacao,CodVaca,CodVacina),
FOREIGN KEY (CodVaca) REFERENCES Vaca (CodVaca)
);
CREATE TABLE Bezerro (
CodBezerro Integer,
CodVaca Integer,
DtNascimento Varchar(40),
Nome Varchar(50),
PRIMARY KEY(CodBezerro,CodVaca),
FOREIGN KEY(CodVaca) REFERENCES Vaca (CodVaca)
);
CREATE TABLE Vacina (
CodVacina Integer PRIMARY KEY,
Nome Varchar(50),
Validade Varchar(40),
Custo Varchar(20),
Qtde Integer
);
CREATE TABLE Frigorifico (
CodFrigorifico Integer PRIMARY KEY,
nome Varchar(50),
Telefone Varchar(20)
);
```

```
CREATE TABLE Alimento (
CodAlimento Integer PRIMARY KEY,
TipoAlimento Varchar(20),
Custo Varchar(20)
);
CREATE TABLE Producao (
DataProducao Varchar(40),
HoraProducao Varchar(20),
CodLeite Integer,
CodVaca Integer,
QtdeProducao Integer,
PRIMARY KEY (DataProducao,HoraProducao,CodLeite,CodVaca),
FOREIGN KEY (CodVaca) REFERENCES Vaca (CodVaca)
);
CREATE TABLE Leite (
CodLeite Integer PRIMARY KEY,
TeorGordura Varchar (20)
);
ALTER TABLE Vaca ADD FOREIGN KEY (CodAlimento) REFERENCES Alimento
(CodAlimento);
ALTER TABLE Vaca ADD FOREIGN KEY (CodFrigorifico) REFERENCES Frigorifico
(CodFrigorifico);
ALTER TABLE Vacinacao ADD FOREIGN KEY (CodVacina) REFERENCES Vacina
(CodVacina);
ALTER TABLE Producao ADD FOREIGN KEY (CodLeite) REFERENCES Leite
(CodLeite);
```

***Script*s para geração do banco de dados MySQL**

Para se gerar o banco com MySQL, segue-se quase que o mesmo feito para o banco Firebird, com algumas alterações que seriam:

- Seção comentada como *Roles* foi excluída devido a isso ser algo específico para geração de bancos com IBExpert.
- Primeira seção excluída devido a fazer referência a banco de dados com Firebird.
- Segunda seção excluída devido a manipular *generator*, algo que não é possível ser feito com o banco MySQL. Em troca a essa opção se tem a propriedade de chave primária do MySQL referente a auto incremento, que se

ela constar na construção das tabelas então aquela chave receberá o auto incremento. A tabela com a propriedade de auto incremento fica:

```
CREATE TABLE ALIMENTO (
    CODALIMENTO   INTEGER AUTO_INCREMENT,
    TIPOALIMENTO  VARCHAR(20),
    CUSTO         VARCHAR(20),
    PRIMARY KEY (CODALIMENTO));
```

APÊNDICE B – DOWNLOAD DE RECURSOS USADOS

Principais *links* das ferramentas utilizadas nesse trabalho, são:

- Servidor de Banco de Dados

Quanto à construção do trabalho os testes foram realizados em dois tipos de servidores, sendo eles:

a) Firebird: para conseguir a instalação de um servidor de banco de dados Firebird, deve-se visitar o site www.firebirdsql.org, onde é possível encontrar um material extenso sobre ele, e também *links* para fazer *download* de uma versão que mais se adapte ao computador que estiver utilizando.

b) MySql: para conseguir a instalação desse tipo de servidor é necessária à visita ao *site* www.mysql.com, onde conseguirá material para estudar e também encontrar *link* para *download* de uma versão, mas com a condição que exige que faça cadastro na página, porém, sem ter que pagar por isso.

- IDE para Programação

Foi escolhida a IDE Net*Beans* para desenvolvimento do trabalho em questão, por ela possibilitar um ambiente de desenvolvimento melhor, devido a condição de se poder montar uma janela através da seleção e arrastar os componentes até ela. Uma breve introdução feita por seus desenvolvedores de como essa IDE pode ser conseguida através do *link* net*beans*.org/index_pt_BR.html. Para baixar uma versão atualizada deve se visitar o link net*beans*.org/downloads/index.html.

A escolha de qual IDE de desenvolvimento usar fica por conta do desenvolvedor, mas deve ficar claro que o trabalho em si foi feito totalmente com a IDE Net*Beans* e para o desenvolvimento com outra versão, teria outros problemas a serem resolvidos.

- *Frameworks*

Alguns *frameworks* precisaram ser baixados, devido não estarem integrados a IDE de desenvolvimento. Quanto aos *frameworks* procurou manter somente aqueles que possibilitaram o desenvolvimento de um aplicativo utilizando sua persistência nas condições exigidas, sendo que os *frameworks* que não contemplaram essa condição, ainda serão listados aqui. Mas como forma de dar um caminho a quem venha utilizar essa material.

Framework integrado a ferramenta de desenvolvimento, são aqueles que devido a sua grande utilização já vem com suas bibliotecas integradas a IDE Net*Beans* e, sendo assim, não é necessário baixar novas bibliotecas para sua utilização. Os

frameworks deste estudo que são integrados a ferramenta são: Hibernate, TopLink, e Spring.

Caso o usuário venha a utilizar outra IDE e queira fazer o *download* de bibliotecas ou queira ver a documentação deles, pode ser conseguido através dos seguintes sites:

- Hibernate: www.hibernate.org
- TopLink: www.oracle.com/technetwork/middleware/toplink/downloads/index.html
- Spring: www.springsource.org

Framework não integrado a ferramenta de desenvolvimento, são aqueles que não são muito utilizados e devido a isso não tem suas bibliotecas dentro da IDE de desenvolvimento e, sendo assim, precisam ser baixadas e integradas manualmente ao seu aplicativo, neste trabalho foi utilizado o DataNucleus

Este *framework* com IDE Eclipse, tem um *plugin* para integração o que possibilita um entendimento mais facilitado de como utilizá-lo. No site www.datanucleus.org podem-se encontrar informações adicionais relacionadas ao *framework*, que vai de como usar, que é possível ver através da documentação, até *downloads* de bibliotecas para ele.