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

**ROBERTO ROSIN**

# SISTEMA PARA COMPOSIÇÃO E CORREÇÃO DE LISTAS DE QUESTÕES DE MÚLTIPLA ESCOLHA

**TRABALHO DE CONCLUSÃO DE CURSO**

**PATO BRANCO**
**2011**

**ROBERTO ROSIN**

# SISTEMA PARA COMPOSIÇÃO E CORREÇÃO DE LISTAS DE QUESTÕES DE MÚLTIPLA ESCOLHA

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Campus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Beatriz Terezinha Borsoi

**PATO BRANCO**
**2011**

ATA Nº: **179**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **ROBERTO ROSIN**.

Às 08:25 hrs do dia 5 de julho de 2011, Bloco S da UTFPR, Campus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Omero Francisco Bertol (Convidado) e Rúbia E. de Oliveira Schultz Ascari (Convidada), para avaliar o Trabalho de Diplomação do aluno Roberto Rosin, matrícula 980498, sob o título **Sistema para Composição e Correção de Listas de Questões**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Coordenação de Informática. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 09:25 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Prof. Omero Francisco Bertol, M.Sc.
Convidado

Profa. Rúbia E. de Oliveira Schultz Ascari, M.Sc.
Convidada

Prof. Omero Francisco Bertol, M.Sc.
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

## RESUMO

ROSIN. Roberto. Sistema para composição e correção de listas de questões de múltipla escolha. 2011. 49 f. Monografia (graduação de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal do Paraná. Pato Branco, 2011.

O uso de sistemas computacionais em atividades de ensino e aprendizagem pode ser visto como um mecanismo de auxílio ao professor e como motivador para o aluno. Atividades como a composição de listas de questões de múltipla escolha e a correção dessas questões, podem ser realizadas por um aplicativo computacional. Responder as listas de perguntas, pelos alunos, por meio de um computador pode motivá-los ao desafio para testes cada vez mais avançados. O próprio aluno pode verificar as suas respostas, buscando aprimorar o seu conhecimento. Esse trabalho se refere ao desenvolvimento de um sistema para composição e correção de listas de questões de múltipla escolha. O fator motivador para a implementação desse sistema é a necessidade de duas alunas do curso de Química da UTFPR, mas o mesmo foi desenvolvido de forma que possa ser utilizado por outras áreas. O sistema foi implementado com a linguagem Java utilizando *Java Persistence API* (JPA) para persistência dos dados. Como forma de fundamentar conceitualmente o trabalho, o seu referencial teórico está centrado em persistência em Java, enfatizando a utilização da JPA.



**Palavras-chave:** Java para *desktop*. *Java Persistence API*. Persistência de dados em Java.

## LISTA DE FIGURAS

## LISTAGENS DE CÓDIGOS FONTE

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| API | *Application Programming Interface* |
| CRUD | *Create, Retrieve, Update, Delete* |
| DDL | *Data Definition Languag* |
| DML | *Data Manipulation Language* |
| EJB | *Enterprise Java Beans* |
| EJB QL | *Enterprise Java Beans Query Language* |
| IDE | *Integrated Development Environment* |
| IECA | Inclusão, Exclusão, Consulta e Alteração |
| JDBC | *Java Database Connectivity* |
| JDK | *Java Development Kit* |
| JDO | *Java Data Objects* |
| JEE | *Java Enterprise Edition* |
| JME | *Java Mobile Edition* |
| JPA | *Java Persistence API* |
| JRE | *Java Runtime Environment* |
| JSE | *Java Standard Edition* |
| JSR | *Java Specification Requests* |
| JVM | *Java Virtual Machine* |
| ORM | *Object Relational Mapping* |
| POJO | *Plain Old Java Object* |
| SDK | *Standard Development Kit* |
| SQL | *Structured Query Language* |
| UTFPR | Universidade Tecnológica Federal do Paraná |
| XML | *Extensible Markup Language* |
| UML | *Unified Modeling Language* |
| SGBD | Sistema Gerenciador de Banco de Dados |

**SUMÁRIO**

# 1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, com uma visão geral do assunto no qual o trabalho se insere, os objetivos, a justificativa e a organização do texto.

## 1.1 CONSIDERAÇÕES INICIAIS

O computador pode ser utilizado de diversas maneiras como auxiliar no processo de ensino e aprendizagem, seja disponibilizando informações por meio da Internet, pelo uso de objetos de aprendizagem que simulam fenômenos físicos ou pelo auxílio na elaboração e na realização de atividades como exercícios e testes.

Um sistema computacional que permita a elaboração de testes como uma lista de questões é interessante porque o professor pode cadastrar as questões, elaborar os testes e os alunos respondê-los no próprio computador. As questões que compõem um teste podem ser escolhidas pelo professor ou por um processo de escolha aleatória realizado pelo computador.

Os resultados dos testes realizados pelos alunos podem ficar armazenados em banco de dados e assim o professor pode acompanhar a evolução de cada aluno e de um grupo de alunos. Com os dados armazenados é possível, ainda, identificar conceitos ou conteúdos da disciplina nos quais os alunos apresentam mais dificuldade. Facilitando, desta forma, a aplicação de recuperação de conteúdos específicos.

Contudo, para que o próprio sistema computacional possa fazer a verificação das respostas do aluno há restrições quanto ao tipo de questão que é elaborada. Questões de múltipla escolha, de associar colunas e de completar frases com palavras, por exemplo, facilitam o processo de verificação. Já questões com respostas abertas, nas quais o raciocínio pode estar amplamente envolvido na resposta, faz com que o processo de correção automatizado seja bastante difícil ou mesmo impossível.

Como forma de facilitar a composição de testes, como listas de questões, e a sua correção, por meio deste trabalho um banco de questões de múltipla escolha é modelado e implementado. A implementação é realizada empregando a linguagem Java em um sistema para *desktop* utilizando a JPA - *Java Persistence API (Application Programming Interface*).

## 1.2 OBJETIVOS

O objetivo geral apresenta a finalidade principal da realização do trabalho realizado e os objetivos específicos complementam o resultado obtido com o objetivo geral.

### 1.2.1 Objetivo Geral

- Implementar um banco de questões com respostas de múltipla escolha.

### 1.2.2 Objetivos Específicos

- Possibilitar a composição de listas de questões escolhidas por um usuário ou sorteadas pelo sistema;
- Armazenar as questões por área, nível de dificuldade e série escolar;
- Permitir ao aluno responder no sistema uma lista de questões que serão corrigidas pelo sistema e as respostas armazenadas em banco de dados;
- Exemplificar o uso da JPA no mapeamento entre os objetos Java e um banco de dados relacional.

## 1.3 JUSTIFICATIVA

A implementação de um sistema para armazenamento de questões facilita o trabalho de professores e pode ser um fator motivador para o aluno. Isso porque o aluno pode responder questões em níveis crescentes de dificuldade e assim o professor pode avaliar o aprendizado de cada aluno e encontrar dificuldades comuns em determinados conteúdos.

O banco de questões desenvolvido será utilizado em um estudo que está sendo realizado por duas alunas do curso de Bacharelado em Química da UTFPR (Universidade Tecnológica Federal do Paraná), campus Pato Branco. Contudo, o aplicativo pode ser utilizado por outras áreas. O sistema comportará apenas questões de múltipla escolha porque essa foi à solicitação das alunas e, ainda, por ser mais simples de realizar a correção das questões.

O sistema possui uma versão *desktop*, desenvolvida pelo autor deste trabalho, e uma versão para *web*, desenvolvida pelo acadêmico Julio Cezar Riffel[1]. Esse desenvolvimento

[1]RIFFEL, J. C. **Sistema web para realização de testes de múltipla escolha.** 2010. 51f. Trabalho de conclusão de curso (graduação em Tecnologia em Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal do Paraná. Pato Branco, 2011.

paralelo de um mesmo sistema possibilitará realizar comparações, do ponto de vista do programador. Os possíveis dados relevantes identificados, de diferenças ou semelhanças da implementação *web* e *desktop* utilizando a mesma linguagem, neste caso a linguagem Java, serão documentados.

Considerando que será utilizada a linguagem Java e um banco de dados relacional é justificada a necessidade de uma forma de mapeamento entre os objetos Java e as tabelas e respectivos campos das tabelas do banco dados. Russel (2008) enfatiza que em aplicações modernas a quantidade de esforço despendido para persistência pode representar a maior parte dos custos de um projeto e usar ferramentas de ORM (*Object Relational Mapping*), que provêm o relacionamento entre objetos e entidades de bancos de dados, pode reduzir significativamente esses custos.

A necessidade de propagar o estado de um objeto da memória para o banco de dados, torna necessária a interação entre a aplicação e a camada de persistência. Com a *Java Persistence API* (JPA), essa tarefa é realizada chamando o gerenciador de persistência, conhecido também como gerenciador de entidades (*entity manager*), responsável por quase todas as operações realizadas com objetos persistentes.

A JPA é uma especificação que provê o mapeamento entre objetos e entidades relacionais de um banco de dados. Essa API será utilizada para a persistência dos dados e isso permite identificar um aspecto de contribuição do trabalho que é exemplificar a persistência em Java quando um banco de dados relacional é utilizado.

## 1.4 ORGANIZAÇÃO DO TEXTO

Este texto está organizado em capítulos, dos quais este é o primeiro e apresenta a ideia e o contexto do sistema, incluindo os objetivos e a justificativa.

O Capítulo 2 contém o referencial teórico que fundamenta a proposta conceitual do sistema desenvolvido. O referencial teórico está centrado em persistência, persistência em Java e enfatiza a JPA, por ser a especificação utilizada para realizar a persistência do sistema desenvolvido.

No Capítulo 3 estão os materiais e o método utilizados no desenvolvimento deste trabalho. São as tecnologias e as ferramentas para a modelagem e a implementação e as atividades realizadas durante o ciclo de desenvolvimento do trabalho.

O Capítulo 4 contém o sistema desenvolvido, com exemplos de documentação da modelagem e de implementação. A modelagem é exemplificada por documentos de análise e

projeto. A implementação é exemplificada pela apresentação do sistema com telas e descrição de suas funcionalidades e por partes da codificação do sistema.

No Capítulo 5 está a conclusão com as considerações finais.

# 2 PERSISTÊNCIA COM A LINGUAGEM JAVA UTILIZANDO JPA

Este capítulo apresenta o referencial teórico que fundamenta a conceituação utilizada no desenvolvimento do sistema obtido como resultado da realização deste trabalho. A JPA, que é uma API para persistência dos objetos Java em bancos de dados relacionais, é a ênfase do referencial teórico por ser utilizada para a persistência dos dados.

## 2.1 PERSISTÊNCIA

É comum que sistemas computacionais necessitem manipular dados armazenados em bancos de dados. Essa manipulação se refere a armazenar, recuperar, editar (atualizar ou alterar) e excluir dados. Para isso as linguagens de programação oferecem formas de solução e implementações dessas operações. O armazenamento dos dados é referido como persistência, no sentido que os dados podem ser mantidos, ou seu estado preservado, após o sistema que os manipula não estar mais em execução. Booch (1998) definiu persistência como os dados que vivem mais que o programa.

Muitas das aplicações atuais são construídas usando duas tecnologias distintas que são a programação orientada a objetos para a lógica de negócio e bases de dados relacionais para o armazenamento de dados (RUSSELL, 2008). Ressalta-se que essas não são as únicas tecnologias utilizadas, há as aplicações procedurais, por exemplo, e banco de dados orientados a objetos, além de outras formas de armazenamento como arquivos textos e formatados, como em XML (*Extensible Markup Language*).

Para Russel (2008) programação orientada a objetos é uma tecnologia chave para a implementação de sistemas complexos, provendo benefícios e reusabilidade, consistência e manutenibilidade. Bases de dados relacionais são repositórios para dados persistentes. E ORM é uma ponte entre essas duas tecnologias para que dados de bases de dados relacionais sejam manipulados mais facilmente por linguagens de programação orientadas a objetos.

Com ORM o mapeamento objeto relacional é automatizado e fica transparente a persistência de objetos em aplicações Java para tabelas em um banco de dados relacional. Esse mapeamento é realizado por meio de metadados que descrevem o mapeamento entre os objetos e o banco de dados.

De certa forma, as operações de consulta e exclusão que podem ser realizadas com os dados armazenados, também podem estar incluídas na denominação genérica de persistência, que parece referir-se ao armazenamento em si. Isso porque a exclusão altera o estado de dados

persistidos. A consulta ocorre em dados persistidos.

A persistência é comumente relacionada aos bancos de dados, especialmente os relacionais porque ainda são bastante utilizados. Contudo, ela também pode estar relacionada a outros formatos de bancos, como os orientados a objetos, o armazenamento em arquivo texto ou armazenamento como dados formatados, como em XML, por exemplo.

Para o armazenamento em bancos de dados relacionais é comum o uso da linguagem SQL (*Structured Query Language*). A SQL possui duas divisões principais (DALLACQUA, 2009): DDL (*Data Definition Language*, ou Linguagem de Definição de Dados) e DML (*Data Manipulation Language*, ou Linguagem de Manipulação de Dados). A DDL é utilizada para criar e alterar e estrutura dos elementos de um banco de dados, usando comandos como *create*, *alter* e *drop*. A DML é usada para manipular e recuperar dados por meio de comandos como *select, insert, update* e *delete.*

Quando um banco de dados relacional é utilizado em uma aplicação, os comandos ou sentenças SQL podem ser transmitidos para a base de dados por meio da API *Java Database Conectivity* (*JDBC).*

## 2.2 PERSISTÊNCIA EM JAVA

Até a versão 1.4 da JEE (*Java Enterprise Edition*) uma das opções para mapear objetos Java para tabelas e campos de um banco de dados era por meio da utilização de *Entity Beans*, que exigem um contêiner EJB (*Enterprise JavaBeans*). As aplicações nas quais a arquitetura não envolvia EJBs precisavam utilizar ferramentas como o Hibernate para fazer a persistência. Outra forma de fazer esse mapeamento era por meio da implementação por código escrito pelo programador para que a persistência ocorresse.

Para que objetos Java possam ser persistidos em um banco de dados relacional é necessário que eles sejam mapeados para tabelas e seus campos. Esse mapeamento é denominado objeto relacional. Isso porque, objetos Java serão persistidos como tabelas (campos de) de um banco de dados.

Para implementar persistência em Java, a camada responsável por esta atividade mapeia objetos para o banco de dados, mesmo os relacionais, de modo que eles possam ser consultados, carregados, atualizados ou removidos sem a necessidade de sintaxe SQL nativa do sistema gerenciador de banco de dados utilizado.

ORM ou mapeamento objeto relacional é uma técnica que permite realizar o mapeamento de banco de dados em objetos de entidades de persistência. A visão de aplicação

de ORM consiste de duas partes principais (RUSSEL, 2008): a API de persistência e as classes de domínio. Em Java, a API é tipicamente um dos padrões de persistência juntamente com EJB e JDO (*Java Data Objects*).

Uma API de persistência prove as operações básicas também conhecidas como CRUD (*Create, Retrieve, Update, Delete*) para criar (cadastrar), recuperar (consultar), alterar (atualizar) e excluir em objetos de classes persistentes.

Uma técnica para que a implementação ORM interaja com objetos transacionais deve prover verificações, associações e outras funções de otimização. Nesse tipo de solução, as sentenças SQL são automaticamente geradas de um mapeamento baseado em metadados. Por exemplo, classe é mapeada para tabela, objeto para linha da tabela, atributo para coluna da tabela e associação é chave estrangeira da tabela. O mapeamento pode ocorrer por meio de XML ou por anotações (*annotations*).

Uma das vantagens de usar padrões de persistência é que eles permitem adiar decisões de persistência, seja em banco de dados ou em provedores de persistência. A API de persistência permite que programadores realizem todas as operações de CRUD padrão do banco de dados (RUSSEL, 2008). Por exemplo, criar uma instância de um domínio de persistência significa criar registros em um banco de dados que correspondem exatamente à instância da classe mapeada. Outro exemplo, excluir uma instância de uma classe de domínio significa excluir a linha ou linhas (registros) de uma tabela do banco de dados que correspondem exatamente à instância da classe de domínio.

Uma API inclui métodos que geram instâncias de uma classe persistente, recuperam uma instância de uma classe por meio de sua identidade primária, localizam todas as instâncias de classes e subclasses por meio de uma consulta expressa em termos de valores de classes. E, ainda, excluem uma instância de uma classe a partir de uma base de dados.

Anterior a versão EJB 3.0 a persistência era parte exclusivamente da plataforma EJB, porém as versões mais recentes já trazem a JPA disponível em uma API própria.

## 2.3 JAVA PERSISTENCE API

JPA é a especificação padrão para o gerenciamento de persistência e mapeamento objeto relacional, surgida na plataforma *Java Enterprise Edition 5.0* na especificação do EJB 3.0 (CASTILHO, 2011). A JPA (JPA, 2011) é uma especificação da Sun Microsystems com a finalidade de simplificar o modelo de persistência de entidades mantidas em bancos de dados relacionais e a forma de manipulação dessas entidades, além de adicionar possibilidades

inexistentes no EJB 2.1.

A JPA é um conjunto de interfaces, especificadas na JSR-220 (*Java Specification Requests)*, responsável pela padronização do mapeamento objeto-relacional na plataforma Java. A API fornece uma solução completa para o mapeamento e persistência de objetos (ROCHA, KUBOTA, 2006), por meio de: a) um modo declarativo de descrever o mapeamento objeto-relacional; b) uma linguagem de consulta; e c) um conjunto de ferramentas para manipular entidades.

Desta forma, ao utilizar JPA, todas as consultas e operações de persistência tornam-se independentes do banco de dados que está sendo utilizado. Com isso, se o banco de dados for mudado o impacto causado na aplicação é menor.

Atualmente, há várias implementações de JPA no mercado. Entre elas, pode-se citar (NASCIMENTO, 2009): EclipseLink da Eclipse (ECLIPSE, 2011), TopLink da Oracle (TOPLINK, 2011), Hibernate (HIBERNATE, 2011), TopLink Essentials por meio de GlassFish (GLASSFISH, 2011), Kodo da Oracle (KODO, 2011), OpenJPA da Apache (OPENJPA, 2011) e Ebean (SourceForge) (SOURCEFORGE, 2011). Cada implementação apresenta seus pontos positivos e negativos, levando em consideração licenças, velocidade de processamento de consultas, erros, bibliotecas adicionais, entre outros fatores.

Um dos principais conceitos relacionados à API JPA é o de entidade (CASTILHO, 2011). Uma entidade corresponde a um objeto que pode ser mantido em uma base de dados a partir de um mecanismo de persistência. A classe que implementa a entidade persistente é um POJO (*Plain Old Java Object*), que contém, basicamente, um construtor padrão sem argumentos e uma chave primária e não precisa derivar de nenhuma interface ou classe, nem implementar qualquer método em especial.

A JPA define uma maneira de mapear objetos POJO e transmiti-los para um banco de dados. Esses objetos são chamados também de *bean* de entidade, que são como qualquer outra classe Java, exceto por representar tabelas e os respectivos campos do banco de dados. Cada *bean* de entidade corresponde a uma tabela em um banco de dados e cada instância de um *bean* corresponde a uma coluna desta tabela.

Um *bean* de entidade não é parte de um banco de dados relacional, entretanto contém dados que são carregados e armazenados do e no banco de dados no momento apropriado. Um *bean* de entidade pode ser compartilhado por múltiplos clientes. Os *beans* de entidade são persistentes. Eles existem mesmo após que a aplicação que o usa termine sua execução.

Um *bean* de entidade poderá, também, ter relacionamentos com outros *beans* de entidade, assim como uma tabela em um banco de dados relacional. Um *bean* de entidade

pode gerenciar sua própria persistência, ou deixar que seja realizada pelo contêiner de aplicações Java. Com o *bean* gerenciando sua própria persistência, o código do *bean* de entidade também inclui sentenças SQL. Quando o gerenciamento da persistência é realizado pelo contêiner, as chamadas de acesso ao banco de dados são geradas automaticamente pelo contêiner.

JPA também permite o uso de XML para definir estes metadados (HEIDER, GÖRLER, REISBICH, 2009). Uma vez feito o ORM, se houver uma mudança da solução que implemente JPA, não será necessário realizar alterações significativas no ORM, pois qualquer solução que implemente JPA será capaz de entender a linguagem de mapeamento. Apenas bibliotecas e arquivos de configuração deverão sofrer alterações relacionadas à nova implementação (SILVA, 2011). Outras vantagens atribuídas ao uso de JPA (O'CONNER, 2007):

a) Não é necessário criar objetos de acesso a dados complexos;

b) A API auxilia no gerenciamento de transações;

c) Possibilidade de escrever código padrão que interage com qualquer base de dados relacional, livre de código específico de fabricantes;

d) É possível descartar SQL, dando preferência a uma linguagem de consulta que usa os nomes das classes e suas propriedades; e

d) É possível usar e gerenciar POJOs.

As entidades de persistência possuem um identificador (descrito pela chave primária) e um estado, sendo seu tempo de vida independente do tempo de vida da aplicação. Assim, aplicações distintas podem compartilhar a mesma entidade, que é referenciada por meio do seu identificador.

Para propagar o estado de um objeto que está em memória para o banco de dados ou vice-versa, há a necessidade de que a aplicação interaja com uma camada de persistência. Com a API JPA, isso é feito invocando o gerenciador de persistência, também conhecido como gerenciador de entidades (*EntityManager*), responsável por quase todas as operações de persistência de objetos.

Outro conceito relacionado à especificação JPA é o de contexto persistente (*PersistenceContext*), que é uma área de memória na qual os objetos persistentes se encontram. Quando da interação com o mecanismo de persistência, é necessário que a aplicação tenha conhecimento sobre os estados do ciclo de vida da persistência.

A implementação da JPA é feita por um *persistence provider* (provedor de persistência). Cada provedor decide a maneira e o momento de carregar, atualizar e armazenar

as entidades, assim como sincronizar os dados com o banco. As configurações utilizadas pelo provedor em uma determinada aplicação (conexão com o banco de dados, entidades que serão gerenciadas, tipo de transação etc.) são descritas em uma *persistence unit*, que é configurada num arquivo especial denominado *persistence.xml*.

As classes e as interfaces da JPA estão localizadas no pacote *javax.persistence*. O mapeamento objeto relacional utilizando JPA pode ser feito a partir de anotações. As anotações podem ser definidas como metadados que aparecem no código fonte e são ignorados pelo compilador (NASCIMENTO, 2009). Qualquer símbolo em um código Java que comece com uma @ (arroba) é uma anotação. Este recurso foi introduzido na linguagem Java a partir da versão JSE 5.0 (*Java Standard Edition*). Em outras palavras, as anotações marcam partes de objetos de forma que tenham algum significado especial.

A persistência possibilita que o objeto (estado) continue a existir mesmo após a destruição do processo que o criou em uma aplicação orientada a objetos, sendo armazenado em disco, podendo ser recriado no futuro em outro objeto. Quatro estados compõem o ciclo de vida de uma entidade: novo (*new*), gerenciado (*managed*), removido (*removed*) e separado (*detached*). A Figura 1 apresenta o diagrama de estados e suas transações.

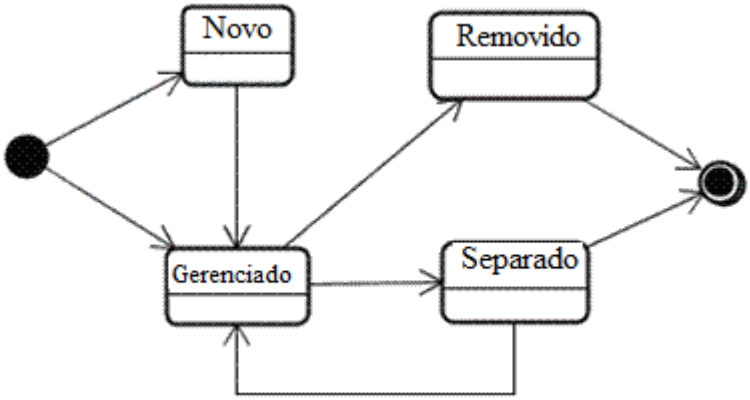


**Figura 1 – Ciclo de vida de uma entidade JPA**

De acordo com o diagrama de estados representado na Figura 1:

a) Um objeto no estado novo é definido a partir do operador *new* e não possui um valor para o seu identificador persistente.

b) Um objeto no estado gerenciado é o que possui um valor para o seu identificador persistente e está associado a um contexto de persistência. Assim, o gerenciador de entidades fará com que a base de dados seja atualizada com os novos valores dos atributos persistentes no final da transação.

c) O estado removido refere-se a objetos associados a um contexto de persistência,

mas que estão marcados para exclusão na base de dados no final da transação, permanecendo nesse estado.

d) Os objetos que se encontram no estado separado possuem um valor para o seu identificador persistente e possuem registros correspondentes na base de dados, mas sem associação com um contexto de persistência.

A JPA faz uso intensivo de anotações. Por isso não é necessário criar descritores XML para cada uma das entidades da aplicação. Uma entidade é uma classe Java comum, rotulada pela anotação *@Entity*. Não é preciso implementar interfaces ou estender outras classes para tornar uma classe persistível. Contudo, é necessário que a classe da entidade possua um construtor sem parâmetros.

Valores padrão podem ser utilizados para mapeamento. O que não é definido explicitamente assume a configuração padrão da API. Por padrão a JPA considera o nome da entidade como o mesmo nome da tabela no banco de dados e o nome da propriedade como o mesmo nome do campo (coluna) da respectiva tabela.

Os dados de uma única entidade podem estar distribuídos em mais de uma tabela, e diversos tipos de relacionamentos entre entidades são possíveis. Os mais comuns são os de agregação (anotações *@OneToOne*, *@OneToMany*, *@ManyToOne*, *@ManyToMany*) e herança (anotação *@Inheritance*).

A JPA oferece suporte a consultas estáticas e dinâmicas (JPA, 2011). A API fornece uma linguagem própria de consulta, que é uma extensão da EJB QL (*Enterprise Java Beans Query Language*). Essa linguagem pode ser usada como uma alternativa ao SQL, que também é suportado. As consultas suportam polimorfismo, o que significa que quando uma entidade é consultada, todas as entidades descendentes que atendam ao critério da consulta também são retornadas. A criação de consultas é feita por meio do *EntityManager*, que fornece métodos específicos para instanciar consultas estáticas e dinâmicas, além de permitir a execução das operações CRUD. As consultas estáticas possuem nomes e são descritas pela anotação *@NamedQuery*. Elas são definidas nas entidades correspondentes e ficam pré-compiladas.

O *EntityManager* utiliza o nome da consulta para instanciá-la por meio do método *createNamedQuery*(). Para usar uma consulta, basta informar os seus parâmetros. A execução pode ser feita pelos métodos *getSingleResult*() ou *getResultList*(), dependendo do resultado desejado. As consultas dinâmicas não possuem nome e podem ser construídas em tempo de execução. A criação desse tipo de consulta é feito pelo método *createQuery*().

# 3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados na realização deste trabalho. Os materiais se referem às tecnologias como linguagens e ferramentas para a modelagem e a implementação do sistema. O método contém as etapas com os principais procedimentos utilizados para o desenvolvimento do sistema, abrangendo o levantamento dos requisitos e a definição das tecnologias a serem utilizadas aos testes.

## 3.1 MATERIAIS

Para a modelagem e a implementação do banco de questões que é resultado do desenvolvimento deste trabalho foram utilizadas as seguintes ferramentas e tecnologias:

a) Astah Community para modelagem;
b) MySQL para o banco de dados;
c) MySQL WorkBench para definição da base de dados;
d) Linguagem Java para implementação do sistema;
e) NetBeans como IDE (*Integrated Development Environment*) de desenvolvimento;
f) JPA para mapeamento objeto relacional entre Java e as entidades do banco de dados relacional MySQL. O referencial teórico sobre JPA está na Seção 2.2 Java Persistence API

### 3.1.1 Astah Community

Astah Community (ASTAH, 2010) é uma ferramenta de modelagem gratuita para projeto de sistemas orientados a objeto. É baseada nos diagramas e na notação da UML 2.0 (*Unified Modeling Language*) e gera código em Java. A Figura 2 apresenta a interface principal dessa ferramenta. Ela é basicamente composta por uma área de edição, à direita, na qual são colocados os componentes gráficos para a representação dos modelos. Esses componentes são pré-definidos e estão representados na barra imediatamente superior à área de edição. Os componentes são apresentados de acordo com o tipo de modelo de diagrama que pode ser escolhido por meio do menu “Diagram”.

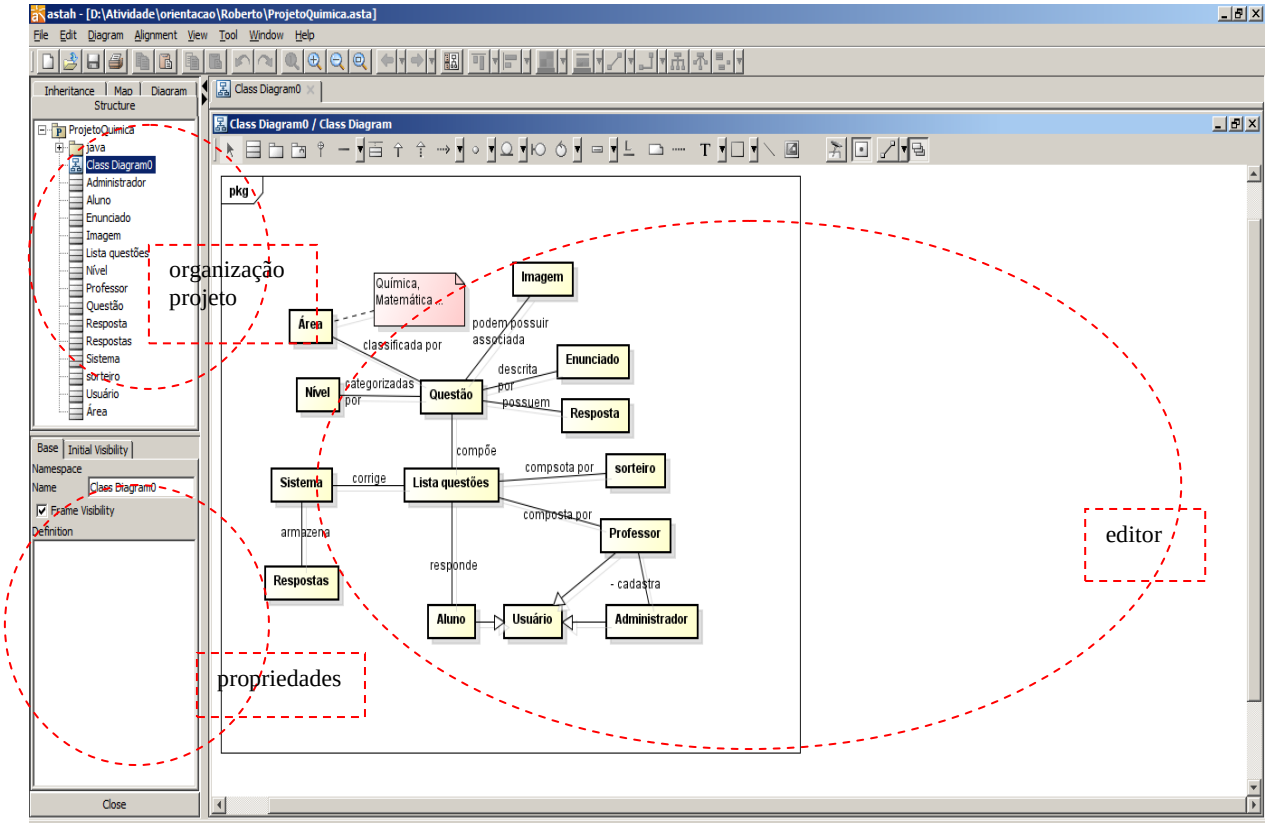


**Figura 2 – Ferramenta de modelagem Astah**

A ferramenta Astah Community está dividida em três partes principais, como pode ser visualizado na Figura 2:

a) Organização do projeto (área superior à esquerda da Figura 2) - é a área na qual estão as diferentes visões do projeto, são elas: *Structure* que é a árvore de estrutura do projeto, *Inheritance* que exibe as heranças identificadas, *Map* para mostrar todo o editor de diagrama e *Diagram* que mostra a lista de diagramas do projeto. Nessa área está um menu em forma de árvore que exibe os diagramas e componentes gerados e permite o acesso para edição dos mesmos. Esses componentes são objetos definidos para compor os modelos a partir das classes que estão na barra de componentes.

b) Visão das propriedades (área inferior à esquerda da Figura 2) - é a área na qual podem ser alteradas as propriedades dos elementos do diagrama. As propriedades de um item selecionado são exibidas e podem ser editadas.

c) Editor do diagrama (área à direita da Figura 2) - é a área na qual são exibidos e compostos os diagramas. Ao ser selecionado um determinado diagrama, dos constantes na lista, o mesmo é carregado e todos os seus elementos gráficos são mostrados nesta área.

A ferramenta Astah Community gera código na linguagem Java, este código é apenas para a definição da classe e dos seus atributos e métodos. Para que essa geração de código

possa ser feita, o diagrama de classes deve estar pronto, as classes bem definidas com o tipo e o tamanho de todos os atributos, e os métodos precisam ter os seus parâmetros definidos.

### 3.1.2 MySQL

O MySQL é um servidor e gerenciador de banco de dados relacional, que utiliza a linguagem SQL (MYSQL, 2011). Ele é de licença dupla (*software* livre e paga). A seguir, algumas das principais características incorporadas na versão 5 do MySQL (MILANI, 2007):

a) Visões – são consultas pré-programadas no banco de dados que permitem unir duas ou mais tabelas e retornar uma única tabela como resultado. Além disso, podem ser utilizadas para filtrar informações, exibindo somente os dados específicos de uma determinada categoria de uma ou mais colunas da tabela. Com o uso de visões, operações frequentes com uniões de tabelas podem ser centralizadas. É possível também utilizá-las para controle de acesso, permitindo que determinados usuários acessem dados de uma visão, mas não as tabelas utilizadas por esta, restringindo acesso a informações.

b) Cursores - cursores possibilitam a navegação em conjuntos de resultados. De forma simples, é possível navegar pelos registros de uma tabela a partir de laços de repetição, permitindo realizar operações necessárias e transações à parte para cada linha da tabela.

c) *Information Schema* - tabelas responsáveis apenas pela organização dos recursos do banco de dados, conhecidos como dicionário de dados, ou metadados. Desta forma, é possível realizar consultas sobre a estrutura do banco de dados por meio dessas tabelas.

d) Transações distribuídas XA - transações distribuídas XA, uma espécie de extensão da ACID (Atomicidade, Consistência, Isolamento, Durabilidade), fornecem a possibilidade de gerenciamento dessas transações realizadas com a união de múltiplos bancos de dados (transações distribuídas) para a execução de uma mesma transação.

e) Clusterização - a clusterização é baseada na integração e sincronismo de dois ou mais servidores para dividirem a demanda de execuções entre si. Além da sincronização de um *cluster*, é possível especificar um balanceador de cargas. Desta forma, esse recurso não gerenciará apenas o redirecionamento de servidores secundários no caso de o primário sair do ar, mas sim balanceará as consultas recebidas pelo *cluster* e irá distribuí-las pelos servidores de acordo com sua disponibilidade.

### 3.1.3 MySQL Workbench

MySQL Workbench (WORKBENCH, 2011) é uma ferramenta gráfica para modelagem de dados. A ferramenta possibilita trabalhar diretamente com objetos *schema*, além de fazer a separação do modelo lógico do catálogo de banco de dados.

Toda a criação dos relacionamentos entre as tabelas pode ser baseada em chaves estrangeiras. Outro recurso que a ferramenta possibilita é realizar a engenharia reversa de esquemas do banco de dados, bem como gerar todos os scripts em SQL.

Com essa ferramenta, a modelagem do banco de dados pode assumir níveis conceituais, lógicos e físicos. MySQL Workbench apresenta uma arquitetura extensível, sendo possível visualizar a representação de tabelas, funções, entre outros.

A Figura 3 é um *print screen* da tela inicial da ferramenta MySQL Workbench e apresenta os três serviços disponíveis pelo MySQL Workbench.

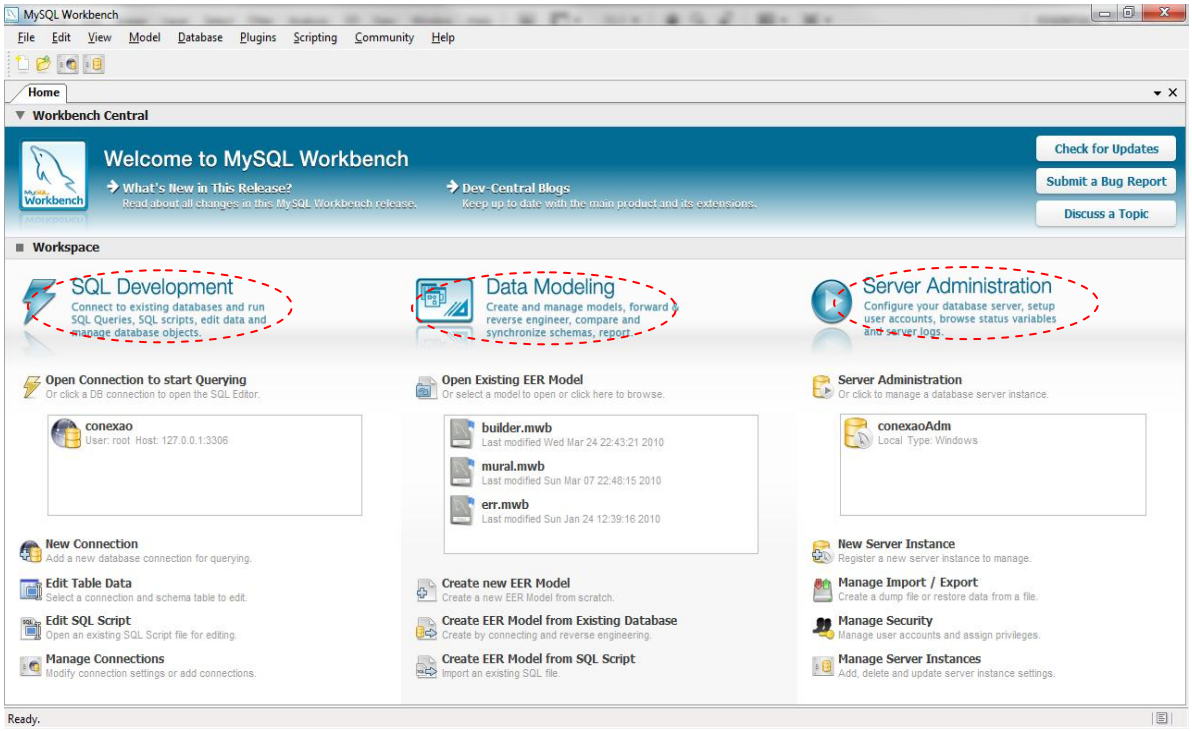


**Figura 3 – Tela dos serviços do MySQL Workbench**

As partes marcadas da Figura 3 indicam:

a) *SQL Development* é uma área para conexão às bases de dados registradas no servidor possibilitando a execução de *query*, *scripts* SQL e editar dados destas bases. O editor SQL informa o usuário caso haja erros de sintaxe, sem necessidade de executar comandos.

b) *Data Modeling* possibilita ao usuário criar e gerenciar modelos de dados e executar engenharia reversa utilizando uma base de dados registrada para gerar um novo diagrama.

Também permite gerar *scripts* completos ou apenas os que o usuário escolher para executar no *SQL Development*.

Com essa ferramenta criar um modelo de dados é simples bastando arrastar e soltar as tabelas. Ao clicar duas vezes em uma tabela, as propriedades da mesma são apresentadas na parte inferior da tela, possibilitando para adicionar colunas, índices e demais componentes de uma tabela.

c) *Server Administration* contém as configurações gerais do servidor, manipulação de usuários, controle de portas e conexões do sistema e do servidor, *backups* e restaurações de bases de dados.

### 3.1.4 Linguagem Java

Java é uma linguagem orientada a objetos e a maior parte dos elementos de um programa Java são objetos. Como exceção cita-se os tipos básicos, como o *int* e o *float*. O código Java é organizado em classes, que podem estabelecer relacionamentos de herança simples entre si.

Um objeto é uma entidade que possui uma identidade que permite individualizá-lo dos demais objetos de maneira inequívoca. Objeto é, assim, um elemento real (cliente, produto) ou conceitual (estratégia de definição de preço, estratégia de jogo), mas deve possuir significado bem definido para o contexto da aplicação. Cada objeto ter sua identidade significa que dois objetos são distintos mesmo que eles apresentem exatamente as mesmas características. Para Booch (1998) um objeto possui estado, exibe algum comportamento bem definido e possui uma identidade única. O estado é definido pelos atributos ou propriedades do objeto, o comportamento são as operações que o objeto realiza e a identidade é a forma de identificação única de um objeto, como por exemplo, o código acadêmico de um aluno.

Uma classe de objetos descreve um grupo de objetos com propriedades (atributos) similares, comportamento (operações) similares, relacionamentos comuns com outros objetos e uma semântica comum.

Java provê o gerenciamento de memória por meio de *garbage collection* (coleta de lixo). Sua função é a de verificar a memória de tempos em tempos, liberando automaticamente os blocos que não estão sendo utilizados. Esse procedimento pode deixar o sistema de *software* com execução mais demorada por manter uma *thread* paralela à execução do programa. Porém, esse procedimento otimiza o uso da memória, evitando problemas como referências perdidas e avisos de falta de memória quando ainda há memória disponível na

máquina.

Quando um programa Java é compilado um código intermediário é gerado, chamado de *bytecode*. Esse *bytecode* é interpretado pelas máquinas virtuais Java (*Java Virtual Machine* - JVM) existentes para a maioria dos sistemas operacionais. A máquina virtual é a responsável por criar um ambiente multiplataforma.

Entre outras funções, a máquina virtual Java também é responsável por carregar de forma segura todas as classes do programa, verificar se os *bytecodes* aderem à especificação JVM e se eles não violam a integridade e a segurança do sistema.

Java é dividida em três grandes edições:

a) *Java Standard Edition* (JSE) - é a tecnologia Java para computadores pessoais, computadores portáteis e arquiteturas com capacidade de processamento e memória consideráveis. Várias APIs acompanham essa versão e tantas outras podem ser obtidas no site da Sun (http://www.sun.com). É com elas que a maioria das aplicações são construídas e executadas. O JSE possui duas divisões: *Java Development Kit* (JDK) ou *Standard Development Kit* (SDK) - um conjunto para desenvolvimento em Java; e *Java Runtime Edition* (JRE) - uma versão da JDK preparada para o ambiente de execução, ou seja, é esta versão que executará os sistemas construídos com a SDK.

b) *Java Mobile Edition* (JME) – é a tecnologia Java para dispositivos móveis com limitações de memória e/ou processamento. Possui APIs que visam economia de espaço, de memória e de processamento. Os aplicativos desenvolvidos com JME são utilizados em sistemas como os de aparelhos celulares, *palm tops, pocket pcs, smartphones* e *javacards*.

c) *Java Enterprise Edition* (JEE) - é a tecnologia Java para aplicações corporativas que podem estar na Internet ou não. Possui um grande número de APIs em que a segurança é a principal preocupação. É ideal para a construção de servidores de aplicação, integração de sistemas ou distribuição de serviços para terceiros.

### 3.1.5 NetBeans

A IDE Netbeans (NETBEANS, 2009) é um ambiente de desenvolvimento utilizado para produzir código fonte de acordo com a sintaxe e a semântica da linguagem Java. NetBeans é um ambiente de desenvolvimento multiplataforma, uma ferramenta que auxilia programadores a escrever, compilar, depurar (*debug*) e instalar aplicações. NetBeans auxilia o trabalho de desenvolvimento de aplicativos por possuir um grande conjunto de bibliotecas, módulos e APIs que são basicamente um conjunto de rotinas, protocolos e ferramentas para a

construção de software. A Figura 4 apresenta a interface da IDE NetBeans, um *print screen* da tela principal dessa ferramenta.

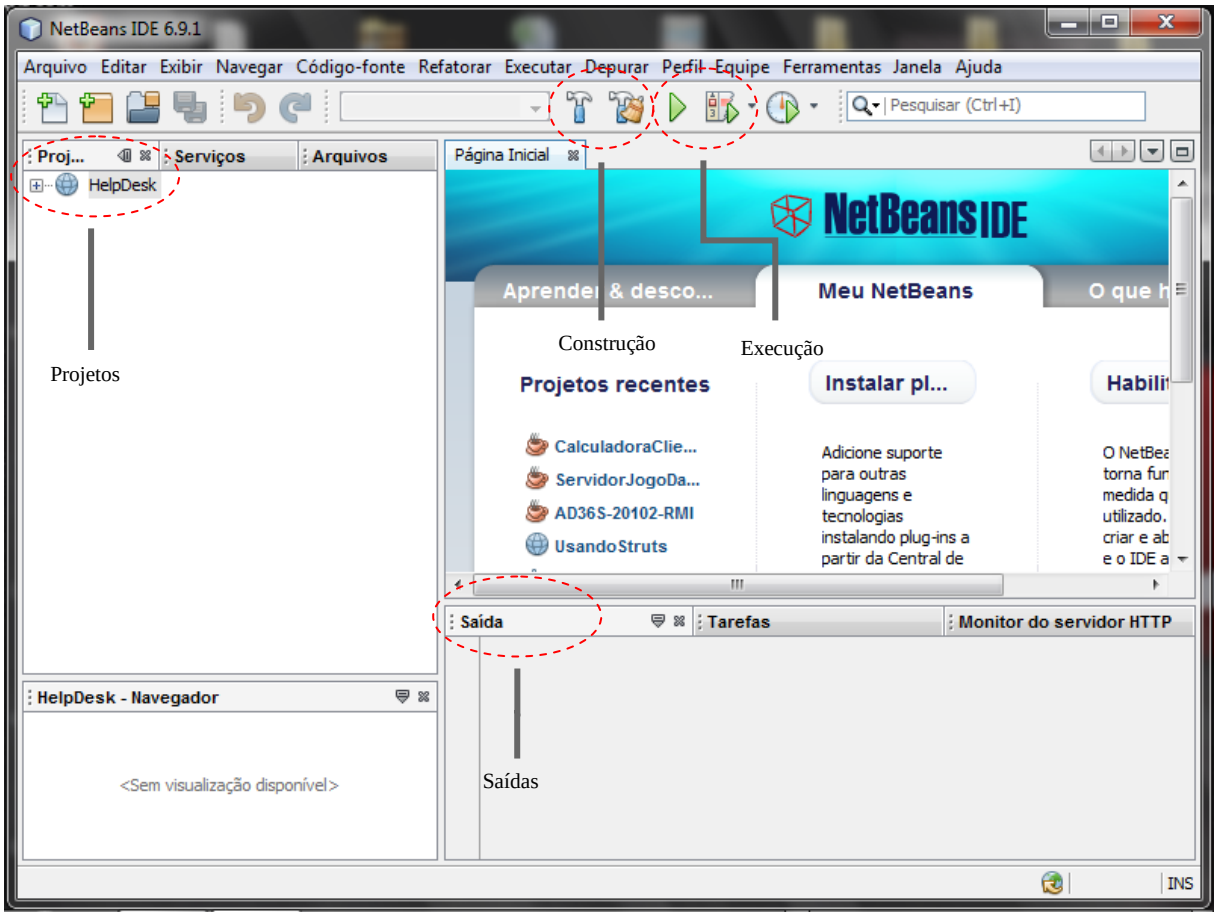


**Figura 4 – Tela inicial do NetBeans**

As áreas circuladas na Figura 4 representam:

a) Projetos – exibe os projetos abertos e seus arquivos.

b) Construção – para fazer a construção ou o *deploy* da aplicação no servidor de aplicações.

c) Execução – o botão mais a esquerda é responsável pela execução do projeto, já o botão mais a direita executa o projeto em modo de *debug* obedecendo aos pontos de parada que foram inseridos no código.

d) Saídas – exibe as mensagens do console.

Como o NetBeans é escrito em Java, ele é independente de plataforma, executa em qualquer sistema operacional que suporte a máquina virtual Java. O Netbeans está disponível para vários sistemas operacionais, como Linux, Windows e Mac OS.

## 3.2 MÉTODO

O método utilizado para a realização do trabalho está baseado nas fases de análise, projeto, codificação e testes do modelo sequencial linear exposto em Pressman (2005). Essas fases foram utilizadas como base, mas sofreram inclusões e alterações para adaptar-se aos interesses deste projeto. Ressalta-se que as atividades realizadas não foram estritamente sequenciais. Ciclos iterativos e incrementais ocorriam, à medida que os requisitos do sistema eram definidos e complementados.

As principais fases ou etapas definidas para o ciclo de vida do sistema são:

a) Planejamento – planejamento do projeto, definindo as principais etapas e atividades. Esse planejamento foi feito com a orientadora visando desenvolver os sistemas e a monografia em paralelo. Os sistemas porque o acadêmico Julio Cezar Riffel implementou uma versão *web* desse mesmo sistema. Reuniões periódicas permitiam avaliar e ajustar as atividades planejadas e estabelecer prazo para o término de cada atividade que seria realizada. Reuniões também ocorreram entre o autor deste trabalho e Julio Cezar Riffel, isso porque os requisitos para o sistema deveriam ser os mesmos. E os dados obtidos com as respostas dos testes deveriam ser integrados.

Uma reunião foi realizada com o professor de Química da Universidade e suas duas alunas orientadas que usariam o sistema. Essa reunião serviu para que o autor deste trabalho e o acadêmico Julio Cezar Riffel pudessem expor o entendimento do sistema a partir dos requisitos que haviam sido inicialmente solicitados. Nessa reunião também foram resolvidas algumas dúvidas dos acadêmicos e formado um entendimento geral da forma de apresentação dos dados obtidos com as respostas pelos alunos às listas de exercícios.

b) Requisitos – a definição dos requisitos do software foi realizada com a ajuda da orientadora, do professor de Química que solicitou a implementação dos sistemas e das duas alunas orientadas desse professor que utilizarão o sistema.

c) Análise – inicialmente foi definido um diagrama que representasse a visão geral do sistema e em seguida foram definidos os seus requisitos, modelados sob a forma de caso de uso e descrições suplementares. Esses requisitos foram complementados com aspectos de qualidade, definindo os requisitos não funcionais. Esses requisitos definiram o problema, ou seja, o que seria implementado pelo sistema.

d) Projeto – tendo como base os requisitos definidos na fase de análise foram definidos os modelos para solucionar o problema, ou seja, definir como o sistema atenderia aos requisitos (funcionalidades e requisitos não funcionais) definidos na análise. Essa solução

foi modelada basicamente por meio de diagramas de classes e entidade-relacionamento da base de dados. Para a modelagem dos dados foi utilizada a ferramenta MySQL Workbench.

e) Implementação – na implementação foi realizada a codificação do sistema e realização de testes pelo programador. A linguagem Java foi utilizada para implementar o sistema com JPA fazendo o mapeamento entre as classes Java e as tabelas do banco de dados relacional, MySQL.

f) Testes – os testes para verificação erros de codificação foram informais e realizados pelo próprio programador, o autor deste trabalho. Os testes para verificar requisitos e funcionalidades do sistema foram realizados pelas alunas do curso de Química para quem o sistema foi desenvolvido.

# 4 RESULTADOS E DISCUSSÃO

Este capítulo apresenta o sistema que foi desenvolvido como resultado deste trabalho. Inicialmente é apresentada a descrição do mesmo. Em seguida está a sua modelagem. E, por fim, é exemplificada a forma de uso das tecnologias empregadas no desenvolvimento do trabalho, enfatizando a JPA.

## 4.1 APRESENTAÇÃO DO SISTEMA

O aplicativo desenvolvido é um sistema computacional para o armazenamento de questões de múltipla escolha e suas respostas. Esse sistema permite a composição de testes a partir dessas questões. As questões armazenadas possuem grau de dificuldade e nível associado. O nível se refere a uma das séries do ensino médio ou pré-vestibular.

Os testes, como uma composição de questões, ficam armazenados no sistema e podem ser respondidos de forma *online* pelos alunos. Os testes são corrigidos pelo sistema e a resposta da correção é disponibilizada para o aluno que o respondeu e para o professor responsável. Os dados das respostas e a quantidade de acertos são armazenados e informados por meio de relatórios.

## 4.2 MODELAGEM DO SISTEMA

A Figura 5 apresenta uma visão geral do sistema como um conjunto de conceitos relacionados entre si.

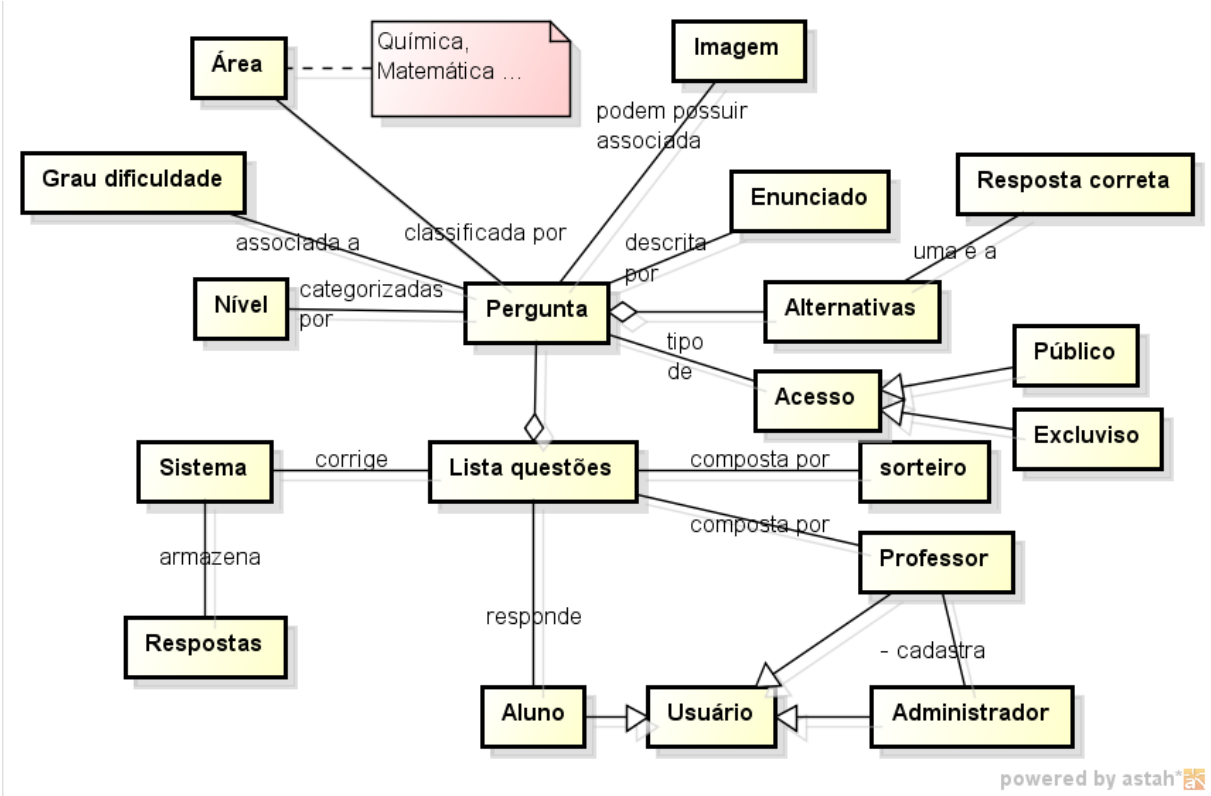


**Figura 5 – Visão geral do sistema**

Pela representação da Figura 5, uma questão pertence a uma área, está categorizada por um nível e associada a um grau de dificuldade. A categorização da área da questão facilita a composição de testes quando o sistema é utilizado por áreas distintas. A categorização por nível (por exemplo, 1º ano do ensino médio, 2º ano do ensino médio, 3º ano do ensino médio e pré-vestibular) e por grau de dificuldade (como fácil, médio e difícil) auxiliam na composição das listas.

Uma pergunta é descrita por um enunciado e é composta por um conjunto de alternativas. Sendo que uma dessas alternativas é a resposta correta à questão. O enunciado da questão é um texto que pode ter uma imagem associada. Cada pergunta pode ter seu acesso público ou exclusivo. O acesso é público quando ocorre para todos os usuários do sistema do tipo professor ou administrador. O acesso é exclusivo quando a referida pergunta somente pode ser acessada pelo usuário que a inseriu.

Uma lista de questões é uma composição de questões e o respectivo enunciado, com imagem associada, se houver, e um conjunto de alternativas. A alternativa que determina a resposta correta é utilizada na correção do teste. Uma lista de questões é composta por sorteio pelo computador ou pelo usuário professor.

O sistema possui como usuários, o administrador que cadastra professores. O

professor pode realizar cadastro (inclusão, exclusão, consulta e alteração) de questões, compor listas e verificar respostas a testes. O usuário aluno somente responde testes e tem acesso à respectiva correção.

A Figura 6 apresenta o diagrama de casos de uso com os requisitos funcionais.

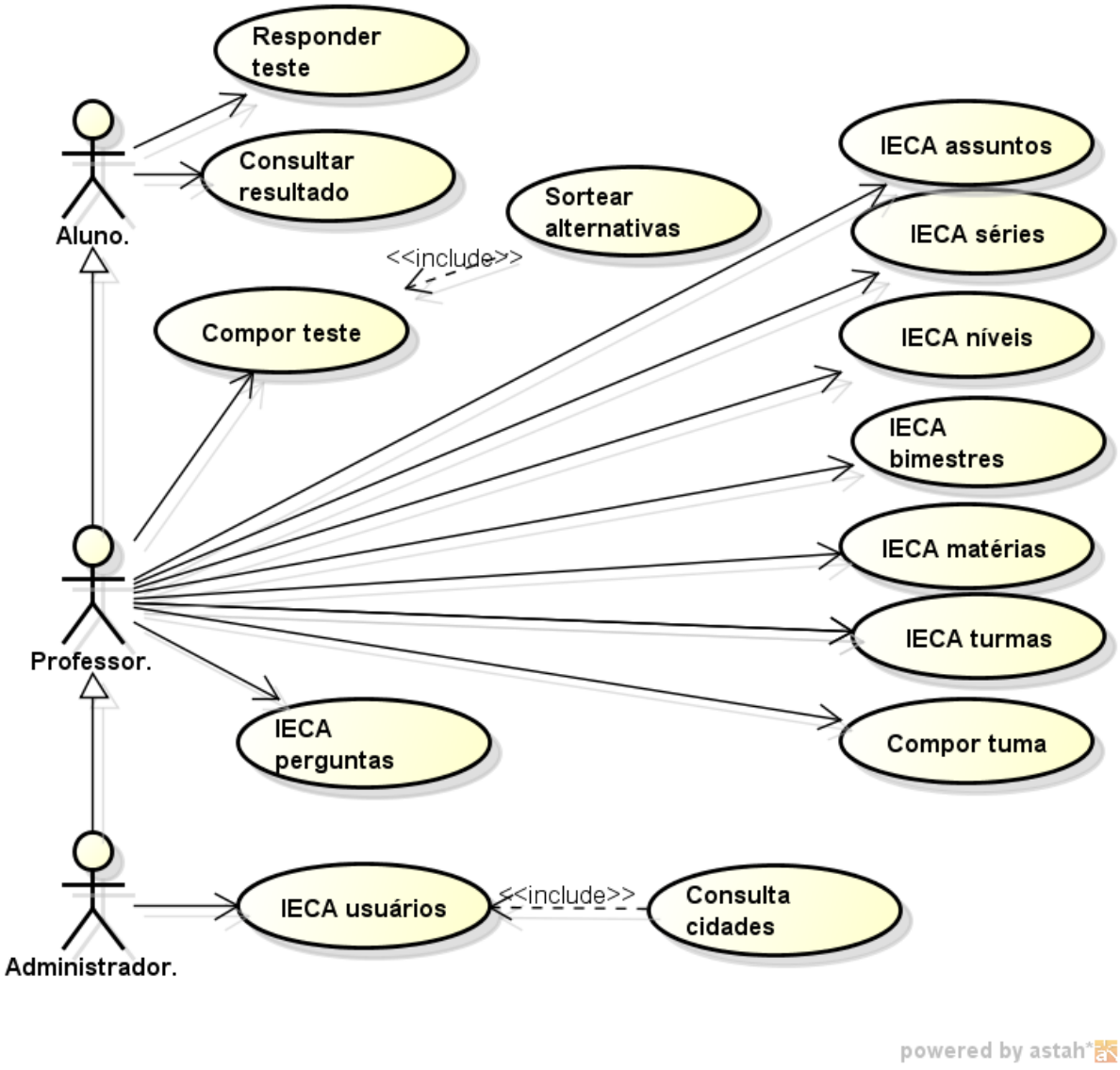


**Figura 6 – Diagrama de casos de uso**

Na Figura 6, a sigla IECA significa Inclusão, Exclusão, Consulta e Alteração. O Quadro 1 apresenta uma descrição sucinta dos casos de uso.

| Caso de uso | Descrição |
|---|---|
| Responder teste | O aluno acessa um teste para ser respondido. E o sistema apresenta um teste para ser respondido. Quando o aluno submete o formulário com o teste o sistema faz a correção, informa o aluno do resultado da correção e armazena esses dados para relatórios posteriores. |
| Consultar resultados | O aluno pode consultar resultados de testes que ele respondeu. |

| | |
|---|---|
| Compor teste | O professor compõe um teste como um conjunto de questões. A composição é realizada a partir de perguntas cadastradas. O sistema faz o sorteio da ordem das alternativas. |
| IECA assuntos | Esse caso de uso se refere à inclusão de um assunto novo, exclusão de assunto cadastrado, consulta de assuntos cadastrados e alteração de um assunto cadastrado. |
| IECA séries | Uma série se refere a um período escolar (como, por exemplo, 1º ano do Ensino Médio ou pré-vestibular).<br>Esse caso de uso se refere à inclusão de uma série, exclusão de séries já cadastradas, consulta de séries cadastradas e alteração de dados de séries cadastradas. |
| IECA níveis | Níveis servem para categorizar as perguntas quanto ao grau de dificuldade, como fácil, médio e difícil.<br>Esse caso de uso se refere à inclusão de um nível, exclusão de níveis já cadastrados, consulta de níveis cadastrados e alteração de dados de níveis cadastrados. |
| IECA bimestres | Um bimestre está relacionado ao período escolar, como, por exemplo, 1º, 2º, 3º e 4º bimestre.<br>Esse caso de uso se refere à inclusão, exclusão, consulta e alteração de bimestres. |
| IECA matérias | Uma matéria se refere a uma disciplina escolar. Por exemplo, Física, Química e Matemática.<br>Esse caso de uso se refere à inclusão de matérias, exclusão e alteração de matérias já cadastradas e consulta de matérias. |
| IECA turmas | Uma turma é um agrupamento de alunos. Assim, um aluno sempre pertence a uma turma.<br>Esse caso de uso se refere à inclusão de turmas, exclusão de turmas já cadastradas, consulta de turmas cadastradas e alteração de dados de turmas cadastradas. |
| Consulta cidades | As cidades foram incluídas diretamente no banco de acordo com um catálogo disponibilizado pelo IBGE. O usuário pode consultar uma cidade e vinculá-la ao seu cadastro. Manutenção de cidades existentes, inclusão de novas e exclusão são realizadas diretamente no banco de dados. |
| Compor turma | A partir de uma turma cadastrada e de alunos, o professor compõe uma turma, vinculando alunos à mesma. Esse caso de uso se refere a associar alunos a uma turma. |
| IECA perguntas | Uma lista é um teste composto por um conjunto de perguntas.<br>Esse caso de uso se refere à inclusão de perguntas, exclusão de perguntas cadastradas, consulta de perguntas cadastradas e alteração de dados de perguntas cadastradas. Na inclusão de uma pergunta é possível vincular uma imagem e deve ser indicado se a mesma é de acesso público ou exclusivamente ao usuário que a incluiu. |
| IECA usuários | Um usuário tem acesso ao sistema para responder um teste e acessar o seu resultado, se aluno. Para todas as funcionalidades de negócio do sistema, se professor. E alteração de nível de acesso, se administrador. |
| Alterar tipo de usuário | O usuário administrador altera o nível de acesso do usuário aluno ou professor cadastrados. |

**Quadro 1 – Listagem dos casos de uso**

Além dos requisitos funcionais definidos como casos de uso, foram identificados os seguintes requisitos não funcionais:

a) Cada pergunta deve ter uma e somente uma alternativa correta;

b) Toda pergunta deve possuir cinco alternativas;
c) O sistema sorteará randomicamente as alternativas de cada pergunta no momento da composição de um teste;
d) Cada pergunta deverá ter definido se o seu acesso é público ou restrito ao usuário que a inseriu;
e) O usuário aluno tem acesso somente às respostas (correção) dos testes que ele realizou.

Foram definidos os seguintes relatórios para o sistema:

a) Para o aluno: lista de perguntas e quantidade de acertos no teste respondido
b) Para o professor: relatório por turma, alunos e nota de cada aluno.

A Figura 7 apresenta o diagrama de classes do sistema.

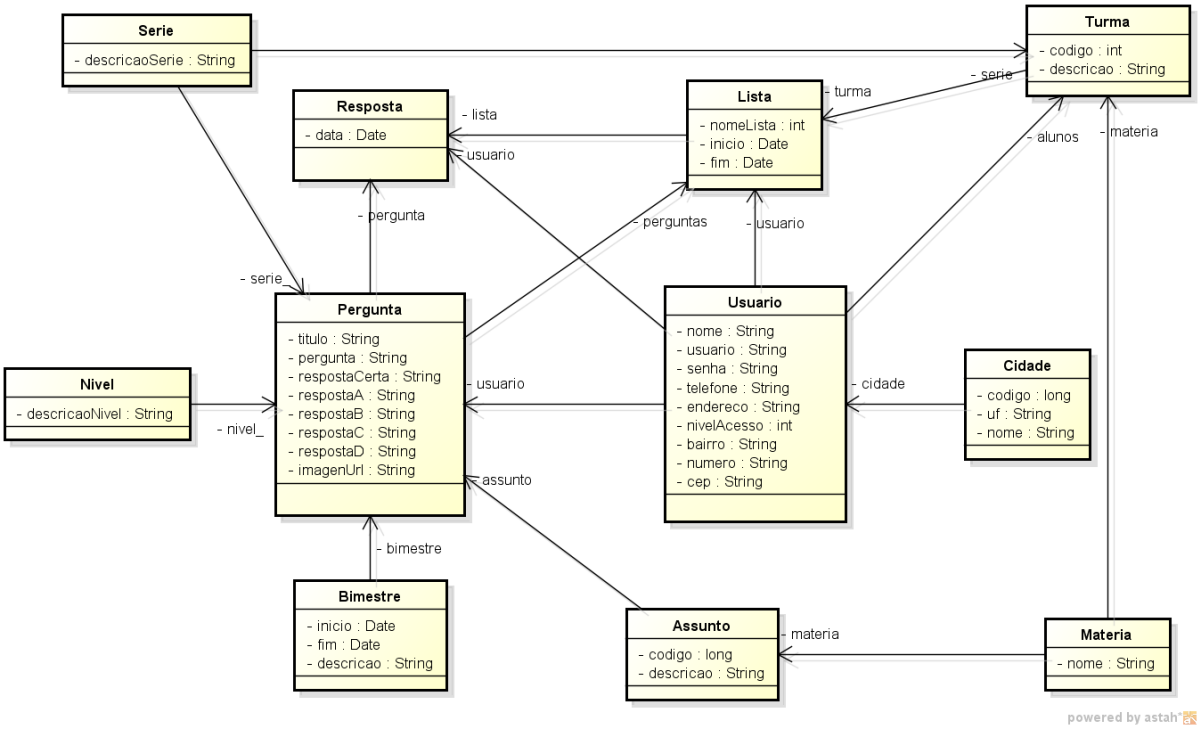


**Figura 7 – Diagrama de classes**

De acordo com a Figura 7 é possível identificar que as classes **Pergunta** e **Lista** são as principais, se considerado como parâmetro a quantidade de ligações que elas possuem. A classe **Pergunta** contém o título da pergunta, a descrição do enunciado da pergunta, as alternativas e a resposta correta. Essa classe está associada com as classes **Bimestre**, **Nível** (que estabelece o grau de dificuldade), **Serie**, **Assunto** e **Resposta** correta. O assunto está associado a uma **Materia**. Isso permite que o sistema seja utilizado simultaneamente por disciplinas escolares distintas, como Física, Química e Biologia.

A classe **Lista** está associada à **Pergunta, Resposta, Usuario** (um usuário responde uma lista) e **Turma**. A classe **Usuario** também está associada à **Resposta** para que seja possível o usuário visualizar o resultado da correção do teste que ele realizou. E, ainda, a **Turma**, porque um usuário aluno pertence a uma turma. Assim, é possível realizar análises de resultados por turma. A classe **Cidade** é utilizada para a inclusão do endereço do usuário.

O diagrama de entidades e relacionamento do banco de dados está apresentado na Figura 8.

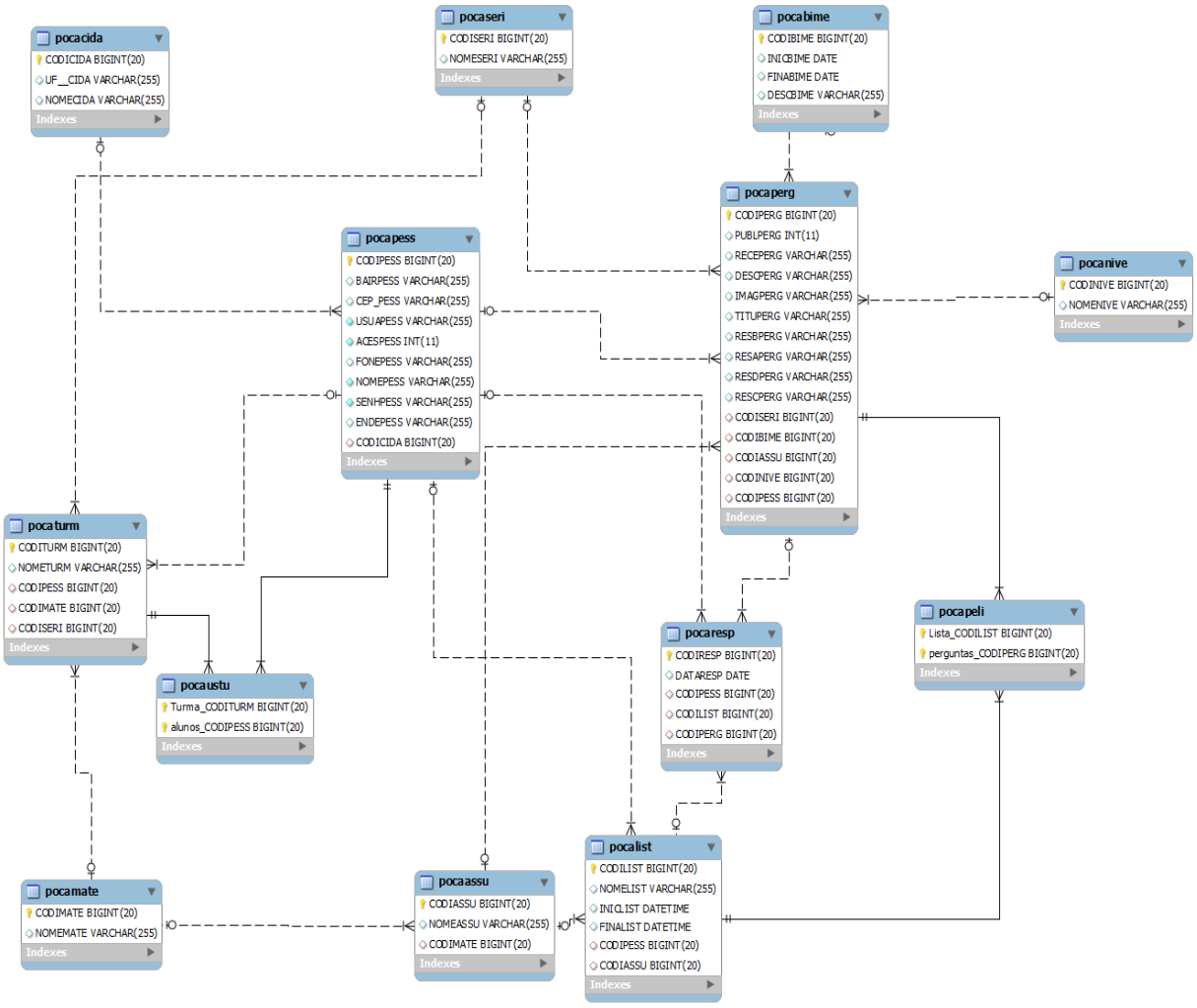


**Figura 8 – Diagrama de entidades e relacionamentos do banco de dados**

## 4.3 DESCRIÇÃO DO SISTEMA

O sistema compreende um banco de questões, sendo que estas questões serão informadas por usuários com nível de acesso de professor ou de administrador. As questões são separadas por Série, Nível, Disciplina, Assunto e Bimestre.

As questões cadastradas são relacionadas a questionários por professores. Esses questionários serão respondidos por alunos relacionados a uma determinada turma, que é cadastrada por um professor e vinculada ao mesmo. No momento de compor uma lista de questões (questionário) o computador faz o sorteio da ordem das alternativas. Assim, uma mesma questão pode ser utilizada em testes distintos, mas a alternativa correta não está na mesma ordem.

O sistema possui uma tela de *login* para controle dos usuários logados no sistema. Somente é possível acessar o sistema, os usuários que já tenham sido previamente cadastrados. A tela de *login* (Figura 9) é composta pelos campos para informar o nome de usuário, a respectiva senha e dois botões, um para entrar e outro para sair do sistema.

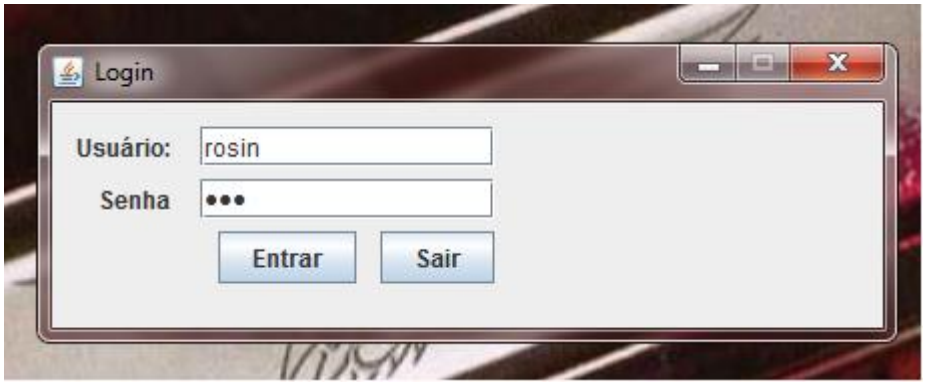


**Figura 9 – Tela de login**

Ao logar no sistema, o usuário tem acesso à tela principal (Figura 10). Essa tela contém os menus para acessar as outras telas do sistema. No menu Cadastro, estão as opções para cadastrar usuários, questões, matérias, assuntos, conteúdos, níveis, séries e turmas. O menu Questionários oferece as opções relacionadas aos questionários, como gerar questionários, montar questionários e responder questionários. O menu Resultados apresenta os resultados dos questionários aplicados aos alunos. Ainda na tela principal há uma barra de ferramentas, com ícones para acesso rápido às principais funcionalidades do sistema.

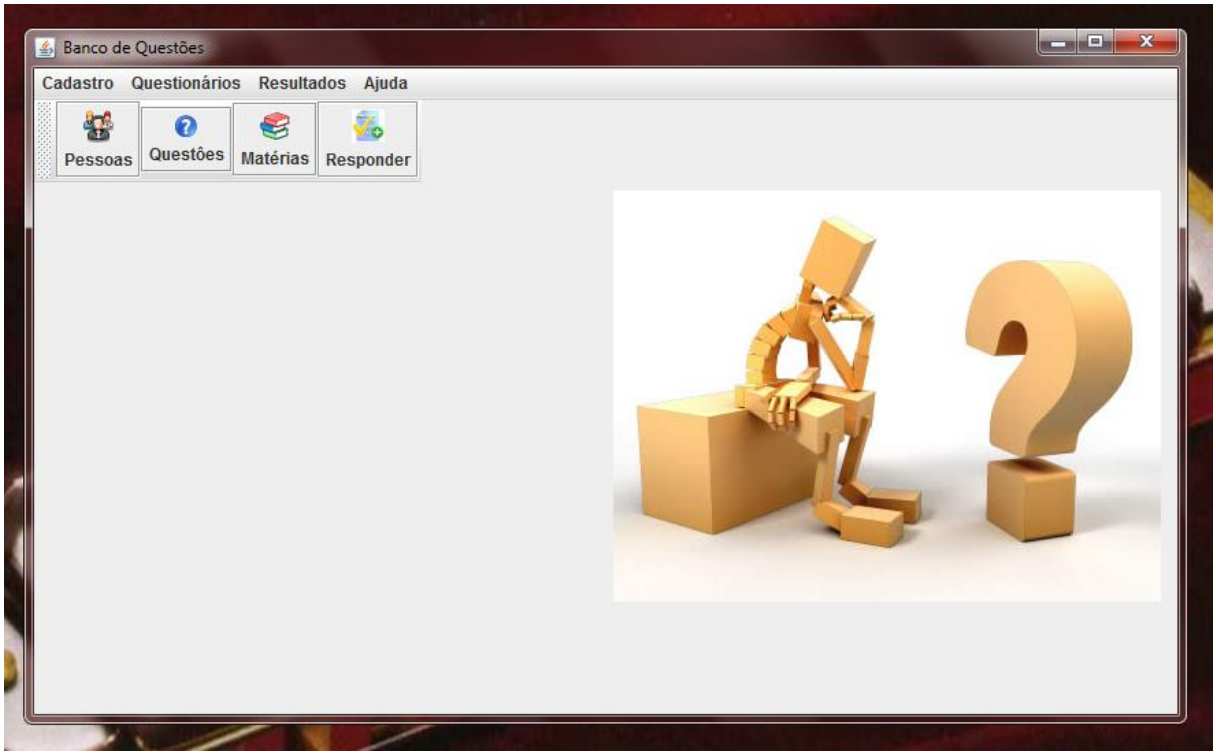


**Figura 10 – Tela principal do sistema**

Ao selecionar uma das opções do sistema, o usuário tem acesso à tela de manutenção. É a tela na qual pode-se excluir, selecionar registros para alteração ou incluir novos registros no banco de dados. Essa tela é composta de um campo do tipo *combo box* para escolher o campo pelo qual será feita a busca, um campo de texto no qual é digitado algum filtro para pesquisa, um botão para pesquisar, um *grid* no qual são exibidas as informações persistidas no banco, um botão para adicionar, um para alterar, um para excluir e outro para fechar a tela.

A Figura 11 apresenta a tela de manutenção de turmas. Nessa tela é possível identificar o padrão de tratamento das operações de inclusão, consulta, exclusão e alteração explicitados no parágrafo anterior. Esse padrão é seguido por todas as telas de manutenção do sistema.

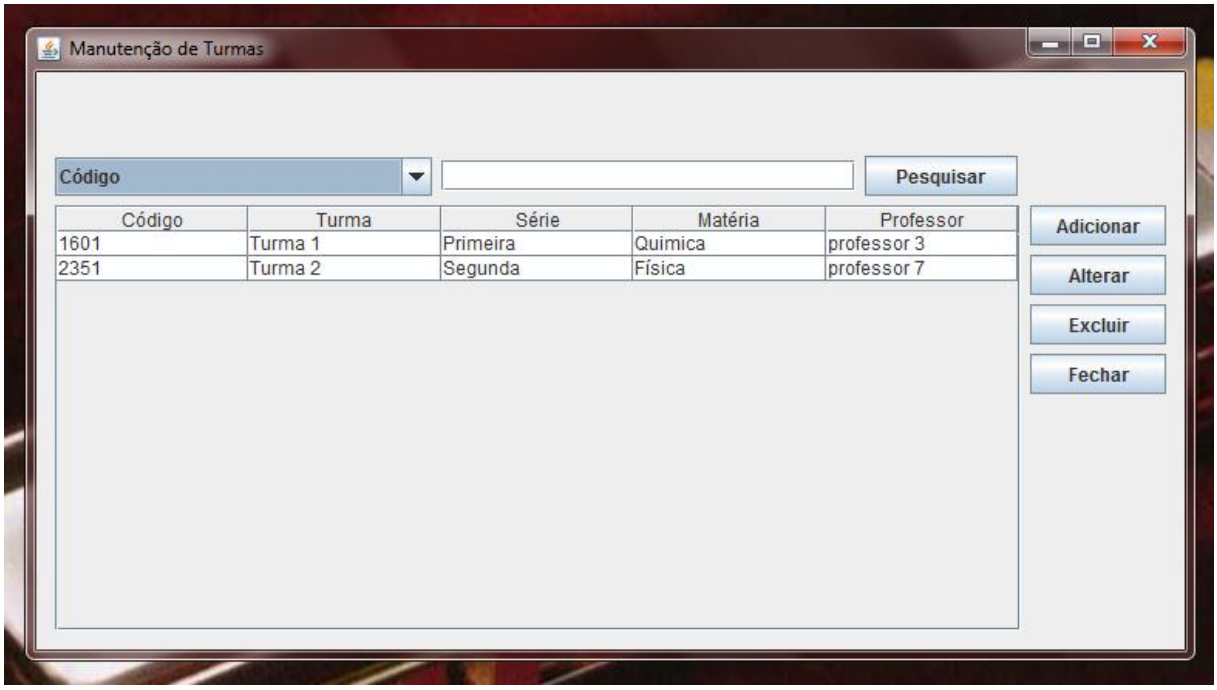


**Figura 11 – Tela de manutenção de turmas**

As telas de cadastro são as mesmas para a inclusão e para a alteração de um mesmo tipo de registro. A tela da Figura 12 é para o cadastro de questionários. Nessa tela há um campo para o nome do formulário, um para a data início da liberação desse formulário para ser respondido e um para a data final. Além de caixas de combinação (*combo box*) para escolher o nível, a série, a turma e o assunto das questões. E, ainda, um botão para listar as questões e um *grid* no qual são listadas as questões referentes aos filtros selecionados. Abaixo do *grid*, botões para adicionar e remover questões de um questionário, um *grid* com as questões referentes ao questionário e os botões de salvar, cancelar e voltar para a tela de manutenção.

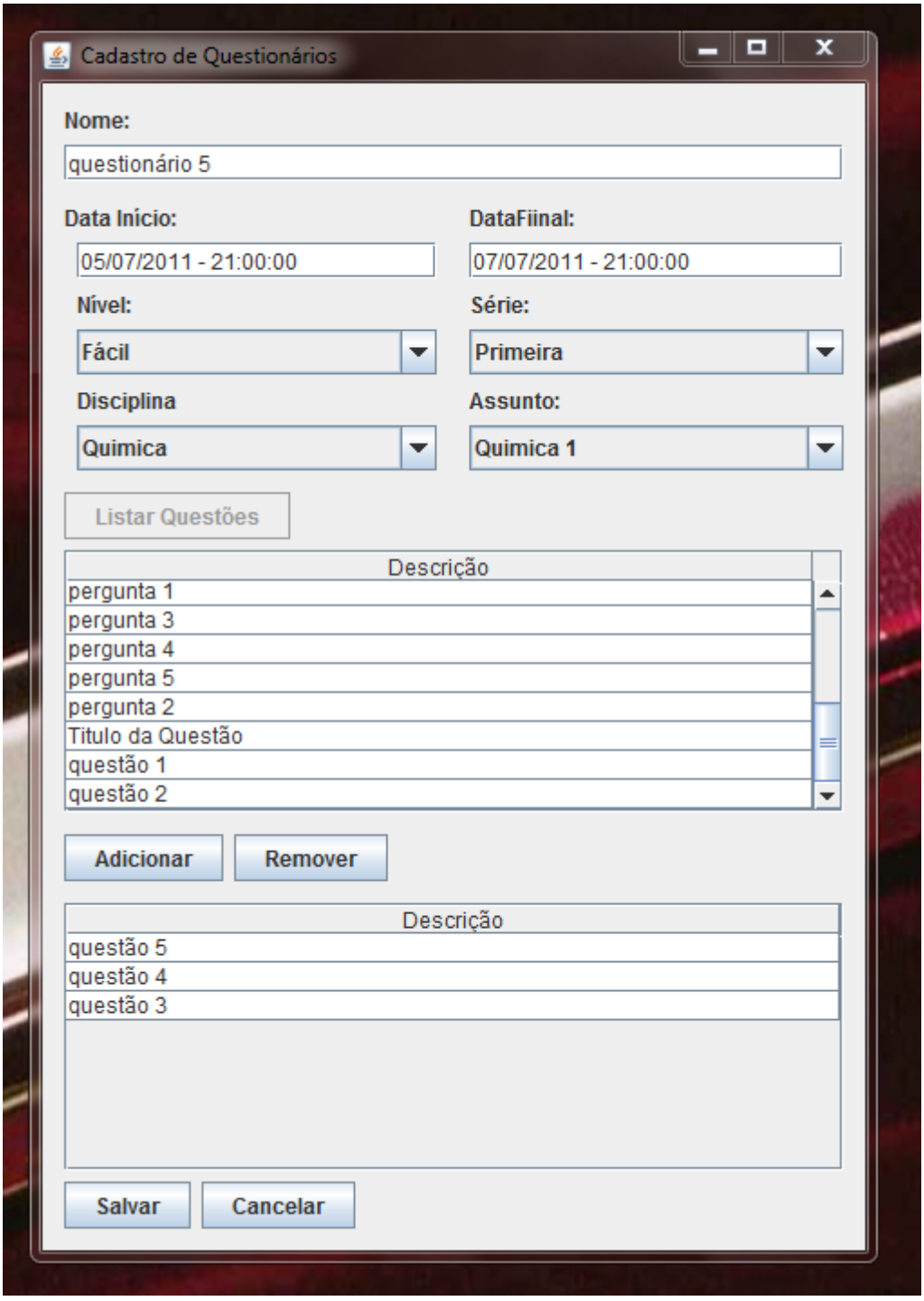


**Figura 12 – Tela de cadastro de questionário**

Mais um exemplo de tela de cadastro é a tela de Cadastro de Questões (Figura 13). Nessa tela há um campo de texto para que seja informado um título rápido para exibição quando da composição de questionários, uma área de texto para a descrição da pergunta, um *check box* para marcar a pergunta como pública ou não. Uma pergunta pública pode ser acessada por todos os professores e uma pergunta privada só pode ser acessada pelo professor que criou a pergunta. Há, ainda, um botão para adicionar uma figura para a pergunta, as

caixas de seleção para selecionar o bimestre, a série, o nível, a disciplina e o assunto da pergunta. Mais abaixo está uma área de texto para preencher as respostas. São enviadas 5 respostas, sendo que uma delas deve, obrigatoriamente, estar marcada como resposta certa. Os botões enviar e remover são para enviar ou remover perguntas do *grid* de perguntas, que está no final da tela. E, por fim, estão os botões salvar, cancelar e voltar à tela de consulta.

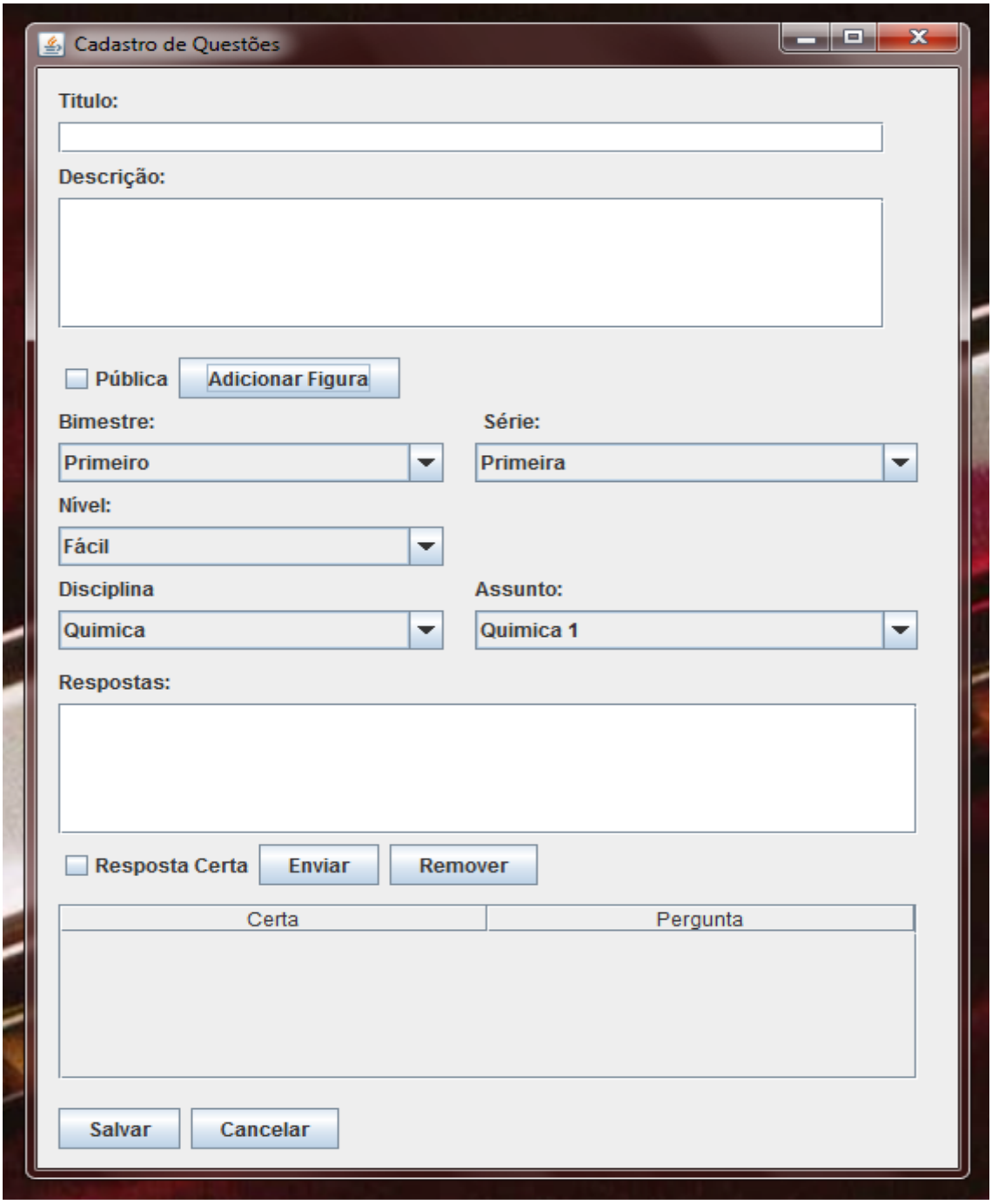


**Figura 13 – Tela para cadastro de questões**

A tela de Responder Questionário (Figura 14) é composta de uma área de texto desabilitada na qual estão listadas as questões e suas respostas. Na parte inferior da tela estão

cinco caixas de verificação (*check box*) para selecionar a resposta que o aluno julgar correta, e os botões para voltar para a pergunta anterior, finalizar o questionário, visualizar a imagem relacionada a pergunta (caso exista) e finalizar o questionário.

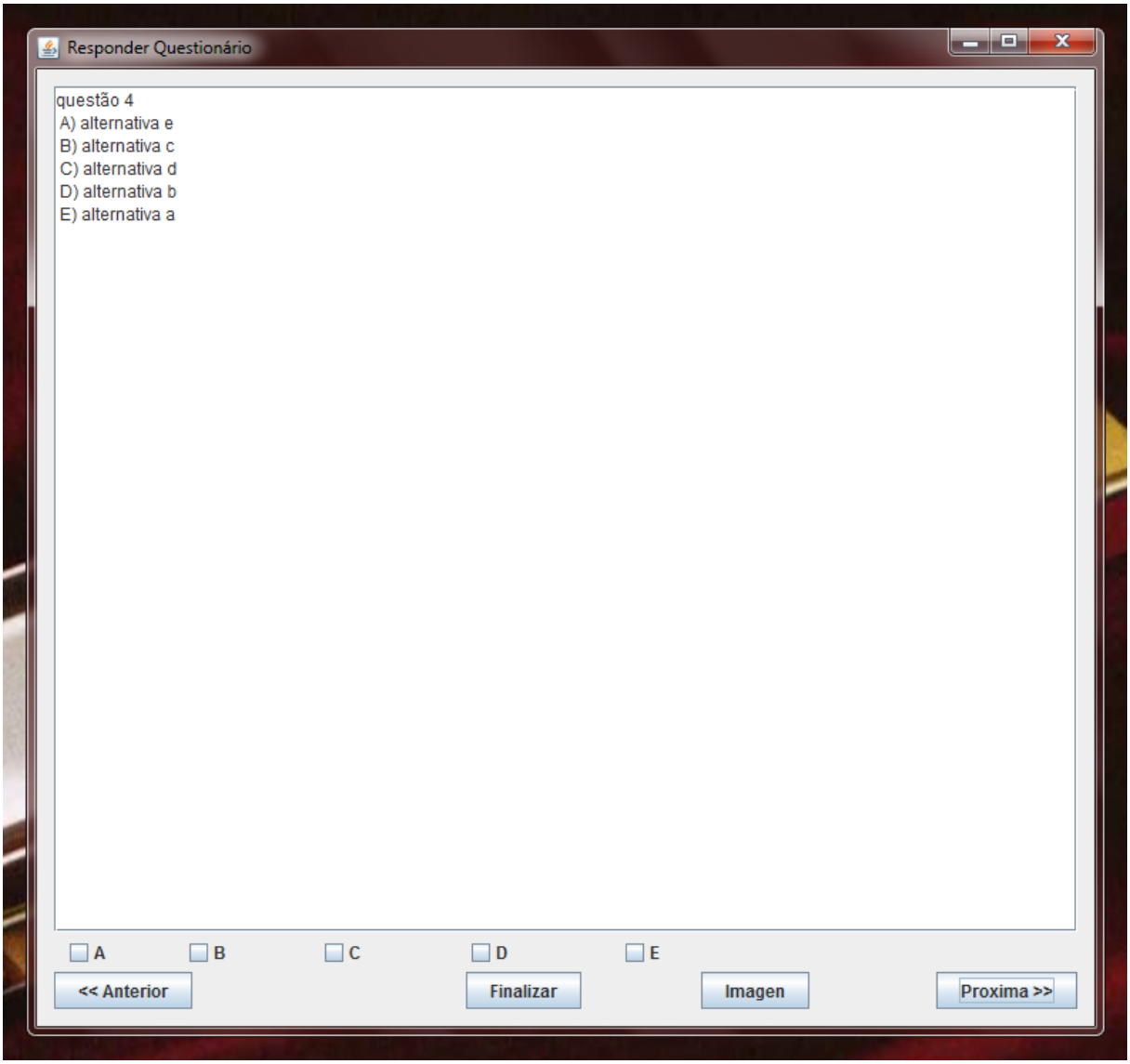


**Figura 14 – Tela para responder questionário**

Na tela de apresentação de resultados (Figura 15) é apresentado o desempenho dos alunos em cada questionário. No exemplo mostrado nessa Figura 15 são apresentados o nome do aluno, o número de questões que ele acertou e a nota do aluno no questionário.

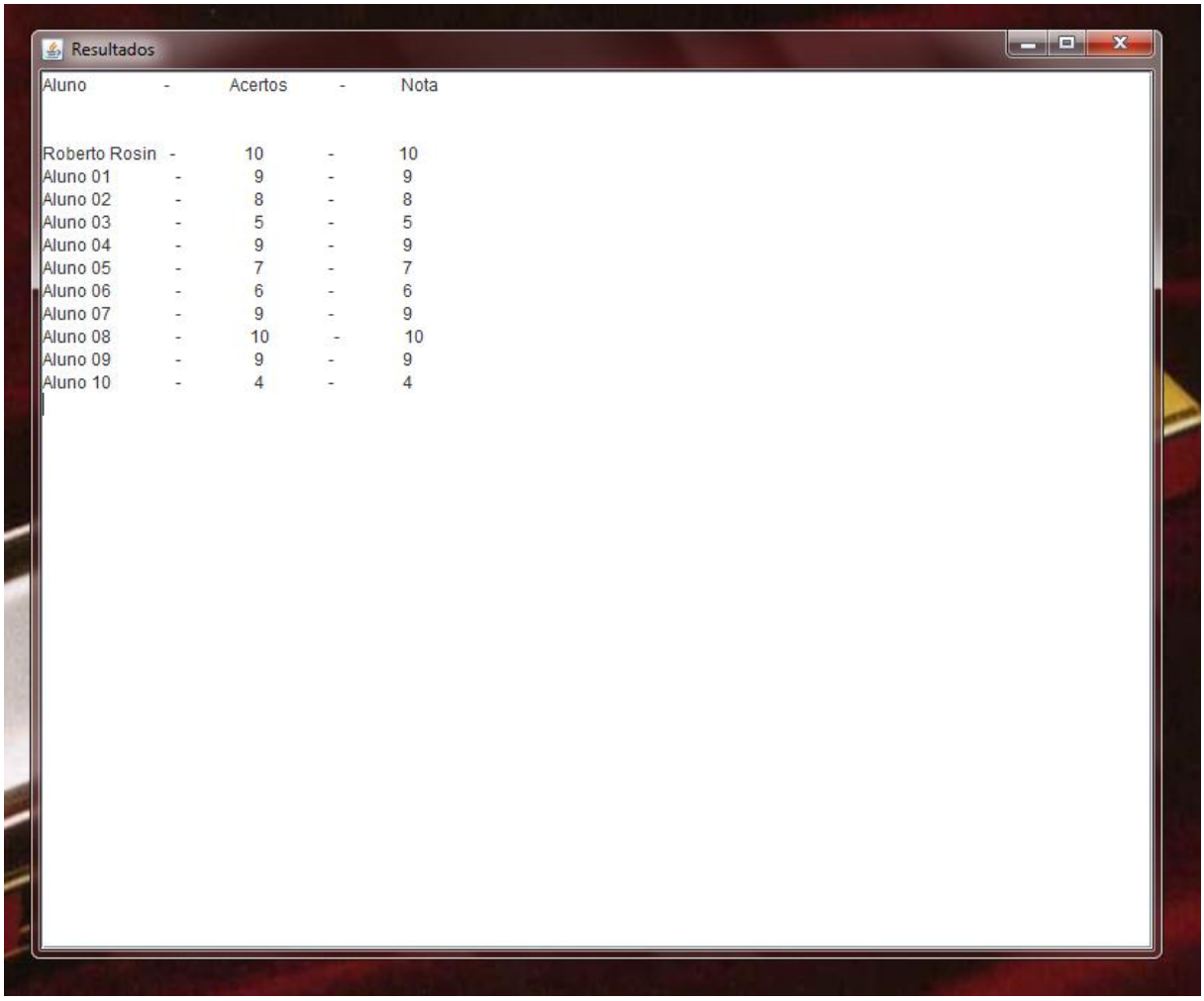


**Figura 15 - Apresentação de resultados**

## 4.4 IMPLEMENTAÇÃO DO SISTEMA

O método **login** que é chamado pela tela de acesso ao sistema e é implementado na classe **Controller** e está exemplificado na Listagem 2.

```
115
116     private void btnEntrarActionPerformed(java.awt.event.ActionEvent evt) {
117         Usuario usu = new Usuario();
118         usu.setUsuario(txtUsuario.getText());
119         usu.setSenha(txtSenha.getText());
120         usu.setNome("Roberto Rosin");
121         usu.setNivelAcesso(3);
122         if (controller.login(usu)) {
123             new Principal().setVisible(true);
124             this.dispose();
125         } else {
126             javax.swing.JOptionPane.showMessageDialog(this, "Usuário ou senha inválidos");
127         }
128     }
129
130     private void btnSairActionPerformed(java.awt.event.ActionEvent evt) {
131         this.dispose();
132     }
133
```

**Listagem 1 – Código para login ao sistema**

O método **login** (Listagem 2) é *booleano* e instancia a classe **UsuarioJpaController**, que é a classe que  é responsável pelo acesso ao banco de dados quando o tipo de dados a ser persistido ou consultado for um **Usuario**. É chamado o método **getUsuarioToLogin(Usuario usuário)**, que recebe como parâmetro um **Usuario** e retorna um valor *booleano*. Se o usuário não foi encontrado no banco de dados é retornado *false* para a tela de *login*. Caso contrário, se encontrado um usuário com aquele nome de usuário e senha no cadastro de pessoas, então, retorna-se *true* para a tela de *login* e carrega-se a variável **usuarioLogado** da classe **Variaveis** de controle com o usuário que foi retornado pelo método.

```
66
67      public boolean login(Usuario usuario) {
68          usuarioJPA = new UsuarioJpaController();
69          usuarioLogado = usuarioJPA.getUsuarioToLogin(usuario);
70          if (usuarioLogado == null) {
71              return false;
72          } else {
73              VariaveisControle.usuarioLogado = usuarioLogado;
74              return true;
75          }
76      }
```

**Listagem 2 – Método da classe para login de usuário**

Na classe **UsuarioJpaController** (Listagem 3) é criado um **EntityManagerFactory** a partir de uma unidade de persistência, que é um arquivo XML que contém as configurações de persistência da API JPA. A partir desse **EntityManagerFactory** é retornado um **EntityManager**, que é o objeto usado para gerenciar as entidades no banco de dados.

```
27
28      public UsuarioJpaController() {
29          emf = Persistence.createEntityManagerFactory("TCCPU");
30      }
31      private EntityManagerFactory emf = null;
32
33      public EntityManager getEntityManager() {
34          return emf.createEntityManager();
35      }
36
```

**Listagem 3 – Código da classe UsuarioJpaController**

Um **EntityManager** é usado para criar, alterar, excluir ou consultar os registros do banco de dados. Na Listagem 4 está o código gerado pela ferramenta NetBeans para criar um novo usuário no banco de dados. Neste método é criada uma nova instância de um **EntityManager**, após esse **EntityManager** recebe o retorno do método **getEntityManager**() e gera um novo **EntityManager** a partir de um **EntityManagerFactory**. Com o **EntityManager** criado é aberta uma transação com o banco de dados, persistido o usuário e feito um *commit* (confirmação) das alterações. Por fim o **EntityManager** é liberado da memória.

```
36
37      public void create(Usuario usuario) {
38          EntityManager em = null;
39          try {
40              em = getEntityManager();
41              em.getTransaction().begin();
42              em.persist(usuario);
43              em.getTransaction().commit();
44          } finally {
45              if (em != null) {
46                  em.close();
47              }
48          }
49      }
50
```

**Listagem 4 – Criação de um novo usuário no banco de dados**

O método para verificar se o usuário e a senha são válidos cria um **EntityManager** e instancia o mesmo (Listagem 5). Depois de criado o **EntityManager**, é criado um **CriteriaBuilder**, que é uma interface da API *Criteria*, que é usada em conjunto com a JPA e a o Hibernate para fazer acesso a dados. A **CriteriaQuery** é uma interface que cria as consultas personalizadas. **Predicate** é a classe que passa os filtros para uma **criteriaQuery**.

```
140
141     public Usuario getUsuarioToLogin(Usuario usuario) {
142         EntityManager em = getEntityManager();
143         try {
144             CriteriaBuilder cb = em.getCriteriaBuilder();
145             CriteriaQuery<Usuario> cq = cb.createQuery(Usuario.class);
146             Root<Usuario> rt = cq.from(Usuario.class);
147             cq.select(rt);
148             Predicate predicate = cb.and(cb.equal(rt.get("usuario"), usuario.getUsuario()),
149                                          cb.equal(rt.get("senha"), usuario.getSenha()));
150             cq.where(predicate);
151             TypedQuery<Usuario> tq = em.createQuery(cq);
152             List<Usuario> usuarios = tq.getResultList();
153             try {
154                 return usuarios.get(0);
155             } catch (Exception e) {
156                 return null;
157             }
158
159         } finally {
160             em.close();
161         }
162     }
163
```

**Listagem 5 – Código do método gerUsuarioToLogin**

Na Listagem 5 está a parte do código da classe **Usuario**. Nesse código, são declaradas as variáveis da classe e definidos alguns parâmetros para a gravação dos campos no banco de dados. Na primeira linha exemplificada está a anotação **@Entity**, que define que aquela classe será uma classe que será persistida no banco de dados. Logo abaixo está outra anotação**, @Table(name=”POCAPESS”).** Essa anotação definirá o nome que a tabela terá no banco de dados, caso não esteja definido, será criada uma tabela com o nome da classe.

A anotação **@Id** informa para a JPA que o atributo a seguir será a chave da tabela, a Identidade. A anotação **@GeneratedValue**, é usada para definir a estratégia que será utilizada para a geração do auto incremento da tabela.

A anotação **@Column** define algumas propriedades da coluna que será gerada no banco de dados, como, no caso o nome da coluna, que será **CODIPESS**. Após essas anotações está o campo **Long id**. Esse campo é a referenciado no banco de dados como ***CODIPESS***,(como definido na anotação **@Column**) é o campo chave da tabela que é gerada pela JPA, e é usado para comparar os registros.

A anotação **@OneToOne** é a anotação que informa à JPA qual será a ligação entre duas tabelas. A anotação **@JoinColumn** informa o nome da coluna que será gerada na classe que servirá de ligação entre as duas tabelas. O código que vem a seguir e que não é mostrado na Listagem 6 é apenas o código de métodos **Getters** e **Setters** dos atributos da classe. Cada atributo dessa classe é uma coluna no banco de dados.

```
22  @Entity
23  @Table(name="POCAPESS")
24  public class Usuario implements Serializable {
25      private static final long serialVersionUID = 1L;
26      @Id
27      @GeneratedValue(strategy = GenerationType.AUTO)
28      @Column(name="CODIPESS")
29      private Long id;
30      @Column(name="NOMEPESS", nullable=false)
31      private String nome;
32      @Column(name="USUAPESS", nullable=false)
33      private String usuario;
34      @Column(name="SENHPESS", nullable=false)
35      private String senha;
36      @Column(name="FONEPESS")
37      private String telefone;
38      @Column(name="ENDEPESS")
39      private String endereco;
40      @Column(name="NUMEPESS")
41      private String numero;
42      @Column(name="BAIRPESS")
43      private String bairro;
44      @Column(name="CEP_PESS")
45      private String cep;
46      @Column(name="ACESPESS", nullable=false)
47      private int nivelAcesso;
48      @OneToOne
49      @JoinColumn(name="CODICIDA")
50      private Cidade cidade;
51
```

**Listagem 6 – Código de mapeamento entre classe e tabela de usuário**

O **persistence.xml**, é um arquivo de configuração que contém as conexões com o banco de dados e as classes que serão persistidas em banco. Nesse arquivo são encontradas informações como a biblioteca de Persistência que será utilizada no projeto, a URL (*Uniform Resource Locator*) de conexão com o banco de dados e a estratégia de geração de tabelas. Essa estratégia informa se as tabelas serão geradas a cada execução, se serão apagadas e criadas a cada execução ou ainda se não serão geradas tabelas, que é a configuração utilizada em tempo de produção do sistema. Define a estratégia de validação a ser utilizada e o modo de compartilhamento de *cache*. A Figura 15 apresenta a configuração das Unidades de persistência

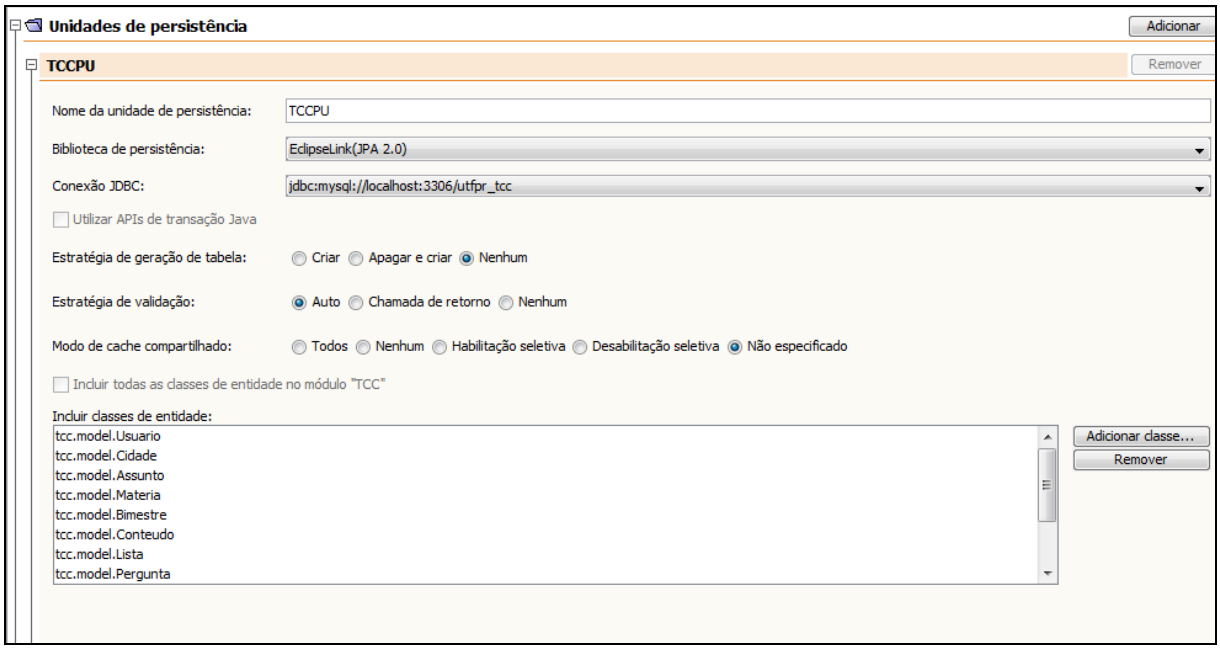


**Figura 15 – Tela para cadastro de questões**

Na Figura 15 também podem ser visualizadas as classes que serão persistidas, informadas na parte inferior dessa tela. Essas classes são apenas aquelas que são anotadas com @**Entity**, que informa que elas são classes de entidade e, portanto, devem ser persistidas. O **persistence.xml** pode também ser visualizado em forma de editor de códigos como mostra a Listagem 7.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/
3    <persistence-unit name="TCCPU" transaction-type="RESOURCE_LOCAL">
4      <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
5      <class>tcc.model.Usuario</class>
6      <class>tcc.model.Cidade</class>
7      <class>tcc.model.Assunto</class>
8      <class>tcc.model.Materia</class>
9      <class>tcc.model.Bimestre</class>
10     <class>tcc.model.Conteudo</class>
11     <class>tcc.model.Lista</class>
12     <class>tcc.model.Pergunta</class>
13     <class>tcc.model.Nivel</class>
14     <class>tcc.model.Serie</class>
15     <class>tcc.model.Resposta</class>
16     <class>tcc.model.Turma</class>
17     <properties>
18       <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/utfpr_tcc"/>
19       <property name="javax.persistence.jdbc.password" value="***************"/>
20       <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
21       <property name="javax.persistence.jdbc.user" value="root"/>
22     </properties>
23   </persistence-unit>
24 </persistence>
25
```

**Listagem 7 – Código para persistence.xml**

## 4.5 DISCUSSÃO: COMPARANDO OS SISTEMAS *WEB* E *DESKTOP*

A discussão aqui apresentada é decorrente de o mesmo sistema ter sido implementado em uma versão para *web* e em uma versão para ambiente *desktop*. O que está colocado nesta seção tem como base o ponto de vista do programador. Assim, são considerados aspectos relacionados à implementação decorrentes de uma análise informal, que tem o objetivo de colocar o ponto de vista de uso de tecnologias.

A versão *desktop* do sistema foi desenvolvida em Java e da versão o desenvolvimento *web* foi realizado utilizando duas linguagens: Flex e PHP.

Há uma grande diferença no método de utilização de banco de dados. Com a linguagem Java é possível conectar diretamente ao banco. E enquanto que com no Flex isso não é possível. É necessário ter uma linguagem *uma backend* para a conexão.

O Flex possui a vantagem de possuir mais componentes prontos que a linguagem Java, que podem ser utilizados ou editados conforme a necessidade.

Para facilitar e agilizar o desenvolvimento em Flex existem várias bibliotecas em *open source* com componentes já desenvolvidos que são, muito semelhantes às APIs disponíveis para Java. Tanto Java quanto Flex são orientados a objetos, facilitando a manutenção e permitindo reaproveitamento de componentes e código.

A plataforma de execução deve apenas possuir uma JVM (*Java Virtual Machine*) no caso do Java ou Flash Player no caso do Flex. Ambas executam da mesma maneira em qualquer plataforma e facilitando o desenvolvimento de aplicações *desktop*, *web* e *mobile*.

# 5 CONCLUSÃO

O objetivo desse trabalho foi alcançado, o sistema foi implementado e disponibilizado para uso. O desenvolvimento desse sistema foi, também, uma oportunidade de aprendizado de tecnologias por meio do uso de Java e JPA no desenvolvimento de um aplicativo para ambiente *desktop*.

A JPA 2.0 agiliza o desenvolvimento das classes de persistência do sistema. O programador não precisa criar a lógica dos métodos inclusão, exclusão, consulta e alteração, basta apenas utilizar os métodos que a JPA dispõe para persistência de dados. Esse *framework* também cria todas as tabelas do sistema no banco de dados, com todos os atributos das classes do modelo. Porém é um *framework* de uso não muito intuitivo, mas à medida que a curva de aprendizado do seu uso aumenta, o desenvolvimento vai se tornando mais fácil e produtivo.

Quanto ao uso da linguagem Java para desenvolvimento para uma plataforma *desktop* essa não se mostrou muito produtiva, pois não há muitos componentes prontos oferecidos com a linguagem para essa plataforma.

O sistema desenvolvido pode ser utilizado como uma ferramenta auxiliar no processo de ensino e de aprendizagem. O professor pode utilizar essa ferramenta como maneira de verificar conteúdos que não foram adequadamente assimilados pelos alunos. O aluno pode utilizar o sistema como forma de verificar o seu próprio aprendizado, desafiando-se a responder testes cada vez mais avançados. E, com base nos seus erros o próprio aluno pode buscar reforço para aprendizado.

Contudo, para que esse sistema se torne uma ferramenta mais efetiva de auxílio ao ensino, o mesmo precisa ser incrementado. Esses incrementos consistem basicamente em categorizar as perguntas por assuntos e identificar os erros e acertos por assunto. Assim, o professor pode identificar conteúdos que precisem ser revistos ou reforçados. De qualquer forma, as alunas que farão uso do sistema o consideram adequado para a finalidade que elas destinaram ao mesmo.

**REFERÊNCIAS**

ASTAH. **Astah community**. Disponível em <http://astah.change-vision.com/en/product/astah-community.html>. Acesso em: 08 mar. 2011.

BOOCH G. **Object-oriented analysis and design. With applications**, 2a. ed. Addison-Wesley, 1998.

CASTILHO, A. S. **Introduzindo java persistence API.** Universidade Federal de Santa Catarina. Centro Tecnológico. Curso de Sistemas de Informação. Disponível em: <http://projetos.inf.ufsc.br/arquivos_projetos/projeto_676/artigo2.doc>. Acesso em: 18 abr. 2011.

DALLACQUA, V. T. **Persistência de dados em java com JPA e toplink**. In: XI Encontro de Estudantes de Informática do Tocantins, 2009, Palmas. Anais do XI Encontro de Estudantes de Informática do Tocantins. Palmas: Centro Universitário Luterano de Palmas, 2009. p. 67-74. Disponível em <http://tinyurl.com/yk7veet>. Acesso em: 20 mar. 2011.

ECLIPSE. **EclipseLink**. Disponível em <http://www.eclipse.org/eclipselink/>. Acesso em: 11 mar. 2011.

GLASSFISH. **Glassfish - toplink essentials**. Disponível em <https://glassfish.dev.java.net/javaee5/persistence/>. Acesso em: 5 fev. 2011.

HEIDER, S., GÖRLER, A. REISBICH, J. **Getting started with java persistence API and SAP JPA 1.0**, SAP AG, 2009.

HIBERNATE. **Java persistence with hibernate.** Disponível em <https://www.hibernate.org/397.html>. Acesso em: 5 mar. 2011.

JPA. **Tópicos avançados de programação.** Disponível em: <http://xa.yimg.com/kq/groups/24791235/236744173/name/Introducao_JPA.pdf>. Acesso em: 22 abr. 2011.

KODO. **Kodo**. Disponível em: <http://download-llnw.oracle.com/docs/cd/E13189_01/kodo/docs41/>. Acesso em: 07 mar. 2011.

MILANI, A. **MySQL – guia do programador**. São Paulo: Novatec, 2007.

MYSQL. **MySQL.** Disponível em: <http://www.mysql.com>. Acesso em: 29 mar. 2011.

NASCIMENTO, N. de L. do. **Persistência em banco de dados: um estudo prático sobre as API JPA e JDO**. Trabalho de graduação. Universidade Federal de Pernambuco. Graduação em Ciência da Computação. Centro de Informática, 2009. Disponível em: <http://www.cin.ufpe.br/~tg/2009-2/nln.pdf>. Acesso em: 18 mar. 2011.

NETBEANS. **NetBeans IDE**. Disponível em: <http://www.netbeans.org>. Acesso em: 12 fev. 2011.

O'CONNER. J. **Using java persistence API in desktop applications**. Sun Developer Network, 2007. Disponível em: <http://java.sun.com/developer/technicalArticles /J2SE/Desktop/persistenceapi/>. Acesso em: 2 abr. 2011.

OPENJPA. **OpenJPA.** Disponível em: <http://openjpa.apache.org/>. Acesso em: 11 mar. 2011.

PRESSMAN, R. **Engenharia de software.** Rio de Janeiro: McGraw-Hill, 2005.

ROCHA, A.D.; KUBOTA, S.O. **Persistência no Java EE**. Java Magazine, Rio de Janeiro, ano 5, n. 39, p.28-37, 2006.

RUSSELL, C. **Bridging the object-relational**. May/June 2008 ACM Queue, p. 16-28.

SILVA, D.S. **Camada de persistência de dados: DAO e activerecord**. Disponível em <http://manifesto.blog.br/1.5/Blog/Programacao/dao-active-record.html>. Acesso em: 9 abr. 2011.

SOURCEFORGE. **Ebeam**. Disponível em <http://sourceforge.net/projects/ebeanorm/>. Acesso em: 5 mar. 2011.

TOPLINK. **Toplink**. Disponível em: <http://www.oracle.com/technology/products/ias /toplink/index.html>. Acesso em: 5 mar. 2011.

WORKBENCH. **MySQL workbench**. Disponível em: <http://wb.mysql.com/>. Acesso em: 2 abr. 2011.