UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

COORDENAÇÃO DE LICENCIATURA EM INFORMÁTICA

DESENVOLVIMENTO DE SISTEMAS PARA INTERNET E DISPOSITIVOS MÓVEIS

RAFAEL CENTENARO

JPDROID: JAVA PERSISTENCE FOR ANDROID

MONOGRAFIA DE ESPECIALIZAÇÃO

FRANCISCO BELTRÃO

2014

RAFAEL CENTENARO

JPDROID:JAVA PERSISTENCE FOR ANDROID

Monografia de Especialização apresentada a Coordenação de Licenciatura em Informática, da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de “Especialista em Desenvolvimento de Sistema para Internet e Dispositivos Móveis”.

Orientador: Prof. Robison Cris Brito.

FRANCISCO BELTRÃO

2014

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Francisco Beltrão
Diretoria de Pesquisa e Pós-Graduação
Curso de Especialização em Desenvolvimento de Sistemas para Internet e Dispositivos Móveis

TERMO DE APROVAÇÃO

Dia 02 do mês de outubro de 2014 às: 19:30 horas, na sala Q-208 - Anfiteatro do Campus Francisco Beltrão, realizou-se a apresentação pública da monografia pelo estudante Rafael Centenaro, intitulada "**JPDROID: JAVA PERSISTENCE FOR ANDROID**". Finalizada a apresentação e arguição, a Banca Examinadora declarou **aprovada** a monografia do estudante, como requisito parcial para obtenção do título de Especialização em Desenvolvimento e Sistemas para Internet e Dispositivo Móveis.

_____
Professor Robison Cris Brito - UTFPR
(Orientador)

_____
Professor Andrei Carniel - UTFPR
(Convidado)

_____
Professor Vinícius Pegorini - UTFPR
(Convidado)

_____
Professor Dr. Ademir Roberto Freddo - UTFPR
(Coordenação)

A folha de aprovação com as rubricas encontram-se disponíveis na Diretoria de Pesquisa e Pós-Graduação, UTFPR, Francisco Beltrão.

Dedico este trabalho à minha família, pelo apoio e compreensão nos momentos de ausência.

AGRADECIMENTOS

Primeiramente gostaria de deixar registrado, o meu reconhecimento á minha família, especialmente aos meus Pais por terem me dado educação e oportunidade de estudar.

Agradeço também ao meu orientador Prof. Robison Cris Brito, pela sabedoria com que me guiou nesta trajetória.

Aos colegas de sala, que colaboraram com sugestões e críticas.

A minha namorada, Ana Paula Wolfart, que me apoiou nos momentos difíceis.

Enfim, a todos que de alguma forma contribuíram para a realização deste trabalho.

Muito Obrigado.

"Os computadores são incrivelmente rápidos, precisos e burros; os homens são incrivelmente lentos, imprecisos e brilhantes; juntos, seu poder ultrapassa os limites da imaginação."

Albert Einstein

RESUMO

A evolução dos dispositivos móveis e seus recursos computacionais permitem o desenvolvimento de aplicações complexas que frequentemente necessitam de um mecanismo para persistência de dados. Por ser uma área emergente há poucas opções de *frameworks* para persistência de dados aplicáveis para plataforma Android. O objetivo deste trabalho é desenvolver um *framework* de mapeamento objeto relacional que seja capaz de realizar operações de persistência de dados, permitindo assim que o desenvolvedor se atenha mais ao desenvolvimento da aplicação. Como resultado foi desenvolvido um *framework* que realiza operações de persistência no banco SQLite. Diferenciando-se dos demais *frameworks* existentes pela simplicidade na utilização e funcionalidades de exportação e importação de dados.

Palavras-chave: Persistência. Android. Aplicação. Móvel. Framework.

ABSTRACT

The evolution of mobile devices and their computing capabilities allow the development of complex applications that often require a mechanism for data persistence. Being an emerging area there are few options for data persistence frameworks applicable to Android platform. The objective of this work is to develop an object relational mapping framework that is capable of performing data persistence operations, thus allowing the developer to stick more to the development of the application. As a result we developed a framework that performs persistence operations on SQLite database. Differentiating itself from other existing frameworks for simplicity in use and export and import data functionality.

Keywords: Persistence. Android. Application Mobile. Framework.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE SIGLAS

ADT	Android Development Tools
API	Application Programming Interface
CSV	Comma Separated Values
DML	Data Manipulation Language
IDE	Integrated Development Environment
JPA	Java Persistence API
SDK	Source Development Kit
SGDB	Sistema de Gerenciamento de Banco de Dados
SMS	Short Message Service
XML	Extensible Markup Language

LISTA DE ACRÔNIMOS

ANSI	American National Standards Institute
CRUD	Create, Read, Update and Delete
ISO	International Organization for Standardization
JAR	Java Archive
JPDROID	Java Persistence For Android
JSON	Java Script Object Notation
REST	Representational State Transfer

SUMÁRIO

1 INTRODUÇÃO

A evolução dos dispositivos móveis e seus recursos computacionais permitem o desenvolvimento de aplicações complexas que frequentemente necessitam um mecanismo de persistência de dados.

A persistência de dados consiste no armazenamento confiável e coerente das informações em um banco de dados que atualmente pode ser relacional ou orientado a objetos.

De acordo com Ramakrishnan e Gehrke (2008, p.3), o conceito de banco de dados pode ser entendido como uma coleção de dados, e para auxiliar na manutenção e utilização destes dados utiliza-se um SGDB (Sistema de gerenciamento de banco de dados).

Ainda segundo Ramakrishnan e Gehrke (2008, p.3), na década de 1980, o modelo relacional foi consolidado como paradigma dominante de SGDB. A linguagem de consulta SQL para banco de dados relacional, desenvolvida como parte do projeto System R da IBM, passou então a ser a linguagem padrão para consultas. A linguagem SQL foi padronizada no final dos anos 80 pela American National Standards Institute (ANSI) e pela International Organization for Standardization (ISO).

O surgimento da programação orientada a objetos na década de 60 possibilitou o desenvolvimento de um novo formato de programação que permite modelar uma estrutura de dados e um comportamento como um objeto.

Este novo formato de programação também influenciou o desenvolvimento de um novo modelo de persistência, sendo que em meados dos anos 80 surge uma nova categoria de banco de dados, conhecido como banco orientado a objetos. No entanto os bancos mais conhecidos no mercado continuam sendo os relacionais.

Embora existam muitos bancos relacionais, muitos programadores preferem manipular os dados usando a Orientação a Objeto, assim surge uma técnica chamada de mapeamento objeto relacional, que consiste em representar os dados para o programador como se fossem orientados a objeto, embora, na prática, toda a persistência e manipulação com o banco ainda ocorra no formato relacional.

Segundo Bauer e King (2004, p.5) persistência de objeto permite o armazenamento do estado do objeto que poderá ser recriado no futuro.

Assim, este projeto apresenta o desenvolvimento de um *framework* em JAVA para persistência utilizando o banco de dados SQLite, este presente na plataforma Android.

1.1 PROBLEMA DE PESQUISA

Existem poucos *frameworks* de persistência para Android. No geral, as opções de *frameworks* disponíveis, visam encapsular o SQLite e oferecer funções simples para edição, exclusão e consulta dos dados.

Parte da implementação de aplicativos sem a utilização de *frameworks*, em especial, no que se refere a banco de dados torna o desenvolvimento da aplicação muito oneroso. Diante deste cenário, existe a necessidade de criar um novo *framework* que possibilite desenvolver aplicações onde o foco se concentre nas regras de negócio, a persistência é uma responsabilidade que será atribuída ao *framework*.

1.2 HIPOTESE

Desenvolver um *framework* de mapeamento objeto relacional que permita realizar operações de persistência utilizando o banco de dados SQLite.

1.3 JUSTIFICATIVA

O Android oferece suporte nativo ao banco SQLite, sendo este um banco leve e poderoso, entretanto, mesmo se tratando de dispositivos móveis os programadores precisam se preocupar com todas as funções de um acesso a banco de dados relacional como: inserção, recuperação, atualização e remoção de registro.

Segundo Lisboa e Mota (2010, p.15) existe a necessidade de construir um software de forma cada vez mais ágil e a exigência da criação de produtos com mais qualidade fazem com que o processo de desenvolvimento de software seja apoiado pelo reuso de estruturas pré-existentes como os frameworks.

Desta forma, o presente trabalho propõe eliminar as dificuldades impostas pelo modelo de persistência existente e tornar transparente a manipulação dos dados do banco. Assim é possível para o desenvolvedor o acesso ao dado, sem que seja necessário conhecimento aprofundado do SGDB (Sistema Gerenciador de Banco de Dados) utilizado.

1.4 OBJETIVOS

Nesta seção serão apresentados os objetivos geral e específico.

1.5 OBJETIVO GERAL

Propor o desenvolvimento de um *framework* para persistência de dados objeto relacional, testando-o em uma aplicação modelo.

1.5.1 Objetivos Específicos

Implementar um novo *framework* de persistência que ofereça funcionalidades básicas de persistência (Inserção, recuperação, atualização, exclusão), conjunto de anotações para mapeamento objeto relacional, recuperação de objetos e seus dependentes, persistência de objetos em cascata, importação de arquivo e exportação de dados nos formatos: Xml, Csv e Json.

Criar uma aplicação de teste utilizando os recursos do *framework*, o aplicativo proposto irá automatizar a emissão de pedidos de venda contemplando cadastro de pessoas, produtos, pedidos de venda, importação e exportação de arquivos.

1.6 METODOLOGIA

Do ponto de vista da sua natureza, este trabalho pode ser classificado como uma pesquisa aplicada, exploratória e estudo de caso.

Segundo Pradanov e Freitas (2013, p.51) uma pesquisa aplicada objetiva gerar conhecimento para aplicação pratica dirigida a solução de problemas específicos.

Ainda segundo Pradanov e Freitas (2013, p.51) uma pesquisa exploratória tem a finalidade de proporcionar informações sobre o tema da pesquisa, possibilitando sua definição e seu delineamento.

Quanto aos procedimentos técnicos, o este trabalho é um estudo de caso, de acordo com Pradanov e Freitas (2013, p.60) este tipo de pesquisa ocorre quando envolve um estudo aprofundado sobre determinado assunto, de forma que permite seu amplo detalhamento e conhecimento.

1.7 ESTRUTURA DO TRABALHO

O presente trabalho está dividido em 5 capítulos. Sendo este o primeiro, que apresenta a introdução, os objetivos e a justificativa do trabalho.

No Capítulo 2 apresenta a fundamentação teórica sobre os conceitos utilizados para construção do *framework* e características do sistema operacional Android.

O Capítulo 3 apresenta a metodologia escolhida para desenvolvimento do trabalho, e os materiais utilizados.

O Capítulo 4 apresenta os resultados obtidos, que se resume ao *framework* e sua utilização em uma aplicação de teste.

Finalizando, o Capítulo 5 apresenta as conclusões, dificuldades encontradas e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica do trabalho, exibe uma visão geral da plataforma Android e também conceitos básicos sobre o banco SQLite, este utilizado no trabalho, além de recursos como *Reflection*, *Annotation* e *Generics* que serão a base para construção do *framework*.

2.1 ANDROID

A Google e a Open Headset Alliliance (OHA) se uniram para revolucionar o mercado mundial de dispositivos móveis. Através desta aliança surgiu a primeira plataforma *Open-Source* para dispositivos móveis chamada Android, esta plataforma oferece para seus usuários uma SDK (Kit para desenvolvimento de *software*) que permite o desenvolvimento de aplicativos com a linguagem Java.

A plataforma Android é um conjunto de *software* que oferece um sistema operacional baseado no Kernel Linux, um *middleware* que possibilita a comunicação entre diferentes aplicativos de baixo nível existentes no sistema operacional, com as aplicações de nível mais alto.

A arquitetura da plataforma Android pode ser visualizada na Figura 1:

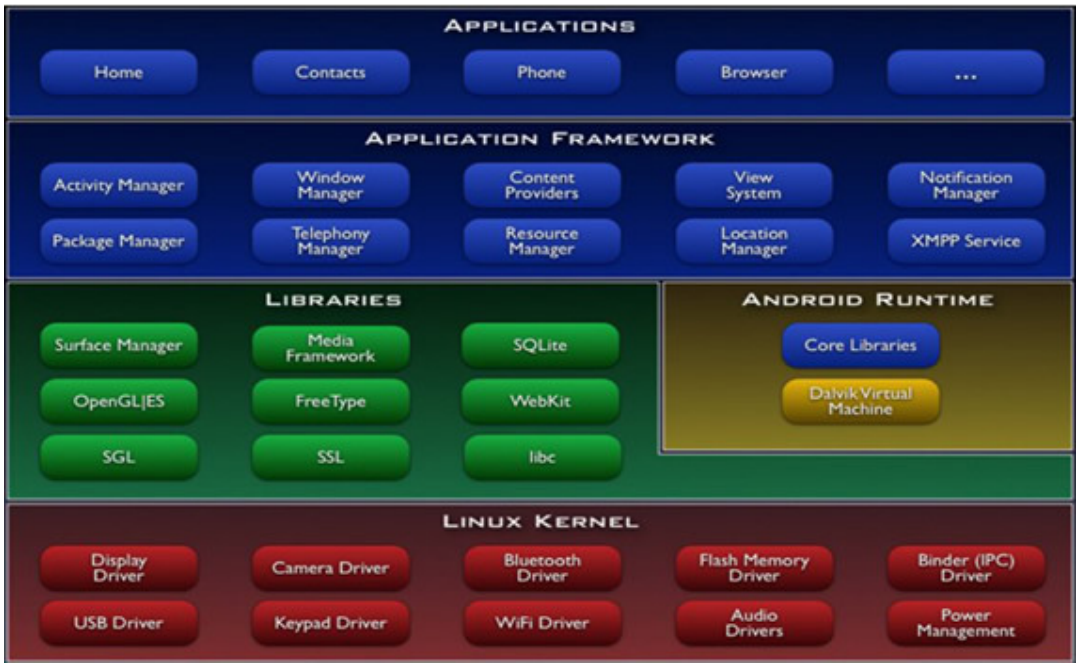

Figura 1 - Estrutura do Sistema Android.
Fonte: ANDROID, 2014

A seguir as camadas da Figura 1 serão detalhadas:

2.1.1 Linux Kernel

Camada composta pelo Kernel Linux que foi modificado para se adaptar ao dispositivo móvel, esta camada é responsável pela abstração de *hardware*, além de tarefas do sistema como segurança, gerenciamento de processos, memória e sistema de arquivo. (ABLESON, 2012, p.10).

2.1.2 Ambiente de Execução (Android Runtime)

A Dalvik VM é uma máquina virtual que emprega serviços do Kernel Linux para oferecer um ambiente otimizado onde serão executados os aplicativos Java. (ABLESON, 2012, p.11).

2.1.3 Bibliotecas (Libraries)

Esta camada possui um conjunto de bibliotecas C/C++ utilizadas pelo sistema e também bibliotecas multimídia, visualização de camadas 2D e 3D, funções para navegadores, gráficos, aceleração de *hardware*, renderização 3D, fontes *bitmap* e vetorizadas e funções de acesso ao banco SQLite. Estes recursos estão disponíveis no *framework* para desenvolvimento de aplicativos. (PEREIRA; SILVA, 2009, p.9)

2.1.4 Framework (Application Framework)

Nesta camada são encontradas as API's (Interface de Programação de Aplicativos) e os recursos utilizados pelos aplicativos, como classes visuais (botões, caixa de texto, *views* e etc), gerenciadores de localização, gerenciador de recursos de notificação e de pacotes de atividade, que controla todo clico de vida da aplicação e o acesso e navegação entre as aplicações.

(PEREIRA; SILVA, 2009, p.6).

2.1.5 Aplicativos (Application)

Nesta camada encontram se todos os aplicativos fundamentais do Android, como cliente de *e-mail*, mapas, navegadores, calendários, programas de SMS, gerenciador de contatos, agendas, entre outros que poderão ser desenvolvidos pelos usuários da plataforma. (PEREIRA; SILVA, 2009, p.6)

2.2 SQLITE

Para persistência de dados, o Android possui o SQLite como recurso nativo, sendo assim não é necessário instalação ou configuração. No Android os bancos

criados pelo SQLite são acessados pelo nome e o armazenamento dos dados ocorre através de um único arquivo, o qual pode ser configurado para permitir acesso apenas à aplicação origem ou compartilhada com outras aplicações.

O SQLite apesar de ser pequeno e leve (250 KB) implementa o padrão SQL92, porém, possui algumas restrições:

- Instruções delete em múltiplas tabelas não são suportados;
- Suporte parcial ao ALTER TABLE: Somente RENAME TABLE e ADD COLUMN do comando ALTER TABLE são suportados. Outras operações como DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT são omitidos;
- Ausência de RIGHT E FULL OUTER JOIN;
- Grant e Revoke: Como o SQLite lê e grava registros em um arquivo as únicas permissões de acesso que podem ser aplicadas são as permissões normais de acesso ao arquivo do sistema operacional. Os comandos Grant e Revoke não são implementados, pois não teria sentido para um banco de dados que pode ser embarcado em uma aplicação.

Estas são algumas das restrições existentes, a lista completa pode ser visualizada em SQLite (2014).

O projeto SQLite existe deste maio de 2000, seu principal autor chama-se D.Richard Hipp. A biblioteca escrita em C possui seu código fonte aberto e de domínio público, seu desenvolvimento contínuo é mantido pelos patrocinadores que fazem parte do SQLite Consortium (SQLite, 2014):

- Oracle;
- Bloomberg;
- Mozilla;
- Bentley;
- Adoble;
- Nokia.

O uso do SQLite é recomendado em aplicações com baixa concorrência de dados, pouco acesso e quantidade de dados limitada, máximo 2 terrabytes, portanto é perfeito para aplicações em dispositivos móveis.

2.3 FRAMEWORK

Segundo Gamma at. al. (2000, p.41) um *framework* é uma estrutura de classes que colaboram para atender a um conjunto de responsabilidades de uma aplicação ou a um domínio de aplicação.

Para construção de um *framework* a orientação a objetos fornece funcionalidades que tornam possível seu desenvolvimento.

Herança é um mecanismo de reutilização de código, que permite criar uma nova classe absorvendo características existentes e estendendo para novas funcionalidades. (SERSON, 2007, p.213)

Interfaces são classes que contém apenas a assinatura dos métodos que serão codificados pela classe que a implementa.

Classes Abstratas são classes que podem conter assinatura de métodos abstratos que serão codificados pela classe que a estende, e também pode conter métodos concretos e atributos. (GUERRA, 2013, p.6)

Polimorfismo é a capacidade de se referir a diferentes derivações de classes da mesma maneira, mas obtendo o comportamento apropriado para a classe derivada que está sendo referenciada. (SHALLOWAY; TROTT, 2001, p.21).

Além dos recursos específicos da orientação a objeto, como as apresentadas anteriormente, é possível utilizar recursos da plataforma de desenvolvimento Java para o desenvolvimento de *frameworks*, conforme segue.

2.3.1 Reflection

Segundo Guerra (2013, p.9) reflexão pode ser definida como um mecanismo pelo qual um programa pode observar e modificar sua própria estrutura e comportamento.

Ainda segundo Guerra (2013, p.9), a introspecção é um subconjunto da reflexão que permite um programa obter informações a respeito de si próprio. A partir das informações obtidas é possível manipular instancias acessando seus atributos e invocando seus métodos.

Em Java, o pacote “java.lang.reflection” contém as classes que implementam funcionalidades relacionadas à reflexão.

Desta forma é possível criar um código que trabalhe com classes cuja estrutura não conhece, porém quando necessário manipular elementos de uma classe de forma diferenciada precisamos identificar estes elementos, para estas situações existe as anotações. Esta API permite marcar elementos de forma que um algoritmo que utilize reflexão possa identificar e manipular de forma diferente estes elementos.

2.3.2 Annotation

Annotation é um recurso do Java que possibilita a declaração de metadados nos objetos, a presença de um metadado não influi no comportamento da classe, são apenas informações complementares que fazem parte da classe. Estes metadados podem ser identificados e processados por um algoritmo que utilize *reflection*.

Todas as anotações estendem a superinterface *Annotation*, que está declarada no pacote “java.lang.annotation”. (BORAKS, 2013, p.489).

As anotações foram incorporadas como padrão de linguagem a partir da versão 1.5 do Java, para versões anteriores era necessário recorrer a arquivos de configuração XML ou a API’s externas como o Xdoclet que fornecia uma funcionalidade parecida e que foi amplamente utilizada. (JCP, 2014).

Uma anotação é facilmente reconhecida através do seu nome precedido pelo caractere “@”, entre as anotações mais famosas estão as utilizadas pelo *framework* de persistência Hibernate (2014), segue um exemplo básico da utilização de anotações como metadados de configuração.

```
1.  @Entity
2.  @Table(name = "tb_pessoa")
3.  public class Pessoa {
4.  @Id
5.  @GeneratedValue
6.  private int id;
7.  @Column(name = "nome ")
8.  private String nome;
9.  @Column(name = "idade")
10. private int idade;
11. }
```

Quadro 1 - Exemplo de anotações do Hibernate.
Fonte: Autoria própria

Detalhamento da Listagem do Quadro 1:

1. Anotação responsável por mapear a classe como tabela;
2. Definição do nome da tabela correspondente ao mapeamento;
3. Cabeçalho da classe Pessoa;
4. Identifica atributo como chave-primária;
5. Define atributo como sequencial;
6. Atributo correspondente à coluna no banco;
7. Identifica atributo como coluna;
8. Atributo correspondente à coluna no banco;
9. Identifica atributo como coluna;
10. Atributo correspondente à coluna no banco;
11. Final Classe.

2.3.3 Generics

Segundo Gamma at. al. (2000, p.37) outra técnica de reutilização de código é o uso de tipos parametrizados, também conhecidos como *generics*. Essa técnica permite definir um tipo sem especificar todos os outros tipos que a classe usa. Os tipos não especificados são fornecidos como parâmetro no ponto de utilização.

Para criar uma classe genérica, basta colocar logo após o nome da classe o indicador genérico representado por uma letra entre sinas de maior e menor. Esta letra será substituída dentro da classe no momento da execução.

Métodos também podem ser genéricos, basta utilizar tipos genéricos nos parâmetros do método e no retorno.

```
1.  public class ClasseGenerica <T> {
2.      T objeto;
3.      public ClasseGenerica (T objeto) {
4.          this.objeto = objeto;
5.      }
6.      public T getObjeto() {
7.          return objeto;
8.     }
9.     public void setObjeto(T objeto) {
10.         this.objeto = objeto;
11.    }
12. }
```

Quadro 2 - Exemplo com Generics.
Fonte: Autoria Própria

Detalhamento da Listagem do Quadro 2:

1. Cabeçalho da classe “ClasseGenerica”, esta classe aguarda o parâmetro T que pode ser um objeto qualquer;
2. Declaração da variável objeto sendo este do tipo T;
3. Construtor da classe;
4. Atribuiu parâmetro recebido pelo construtor;
5. Final método;
6. Método responsável por retornar o valor da variável objeto;
7. Retorno do método getObjeto();
8. Final método;
9. Método responsável por atribuir parâmetro a variável objeto;
10. Atribui parâmetro a variável;
11. Final método;
12. Final Classe.

Por convenção, os tipos genéricos são representados por uma única letra maiúscula, desta forma fica nítida a diferença entre tipos genéricos para classes ou interfaces.

- E – Elemento (*Element*);
- K – Chave (*Key*);
- N – Número (*Number*);
- T – Tipo (*Type*);
- V – Valor (*Value*).

Em um código genérico, a representação através da interrogação (?) chama-se *wildcard*, do inglês que quer dizer coringa, esta representação é utilizada para tipos desconhecidos. (ORACLE, 2014).

```
1. public void processaItens(List<?>  itens){
2.     for(Object o : itens){
3.         System.out.println(o);
4.     }
5. }
```

Quadro 3 - Método Genérico
Fonte: Autoria Própria

Detalhamento da Listagem do Quadro 3:

1. Método responsável por processar os itens enviados por parâmetro;

2. Estrutura de repetição que irá percorrer os itens da lista;
3. Imprime na tela o objeto como string;
4. Final da estrutura de repetição;
5. Final método;

O exemplo acima é capaz de imprimir qualquer tipo de lista de objeto.

3 METODOLOGIA

Neste capítulo serão apresentadas as metodologias utilizadas para desenvolvimento deste trabalho. Também serão apresentadas as ferramentas utilizadas para construção do *framework*.

Segundo Mattson (2014, p.12), o processo de desenvolvimento e uso de um *framework* orientado a objeto compreende três atividades macros:

a) Análise de domínio: Identificar de forma completa os requisitos de um problema, organizar o conhecimento para dar suporte a solução destes problemas;

b) Projeto do framework: O objetivo desta atividade é tornar o *framework* o mais flexível possível. O esforço desta atividade demanda muita pesquisa sobre *Design Patterns* para tentar encontrar as abstrações corretas e identificar as variáveis para permitir a extensibilidade de flexibilidade de forma que atenda necessidades futuras;

c) Instanciação do *framework*: A instanciação difere dependendo do tipo do *framework*, podendo ser classificada como “Caixa branca” quando o usuário constrói classes a partir das classes disponíveis ou “Caixa Preta” quando o usuário tem que escolher uma das classes disponíveis. Um *framework* pode ser instanciado uma ou mais vezes dentro da mesma aplicação ou em diferentes aplicações.

Atualmente existem poucas metodologias voltadas para o desenvolvimento de *frameworks*, a metodologia adotada para desenvolvimento do presente trabalho chama-se *Example-Driven Design* (Projeto Dirigido por Exemplos) e foi apresentada por JHONSON (2014).

Nesta metodologia, a construção de um *framework* deve seguir três etapas que se assemelham as etapas definidas por Mattson (2014, p.12):

a) Analisar o domínio do problema, estudar as abstrações conhecidas, coletar exemplos de programas que podem ser construídos com o *framework*;

b) Projetar abstrações que podem ser especializadas para construir os exemplos;

c) Testar o *framework* através do desenvolvimento de exemplos. Cada exemplo é uma aplicação separada e a execução do teste consiste em implementar o *software*.

3.1 TECNOLOGIA ENVOLVIDA

As ferramentas utilizadas para desenvolvimento do *framework* e da aplicação de teste serão apresentadas no Quadro 4.

Ferramenta	Fabricante	Site	Versão
IDE Eclipse	Eclipse Foundation	https://www.eclipse.org/	Free
Plugin ADT	Google	http://developer.android.com/	Free
Java JDK	Oracle	http://www.oracle.com/	Free
Android SDK	Google	http://developer.android.com/	Free
Banco SQLite	Sqlite	http://www.sqlite.org/	Free
Object Aid Class Diagram	Object Aid LLC	http://www.objectaid.com/	Free

Quadro 4 - Ferramentas Utilizadas
Fonte: Autoria Própria

A utilização das ferramentas citadas acima é apresentada no diagrama da Figura 2.

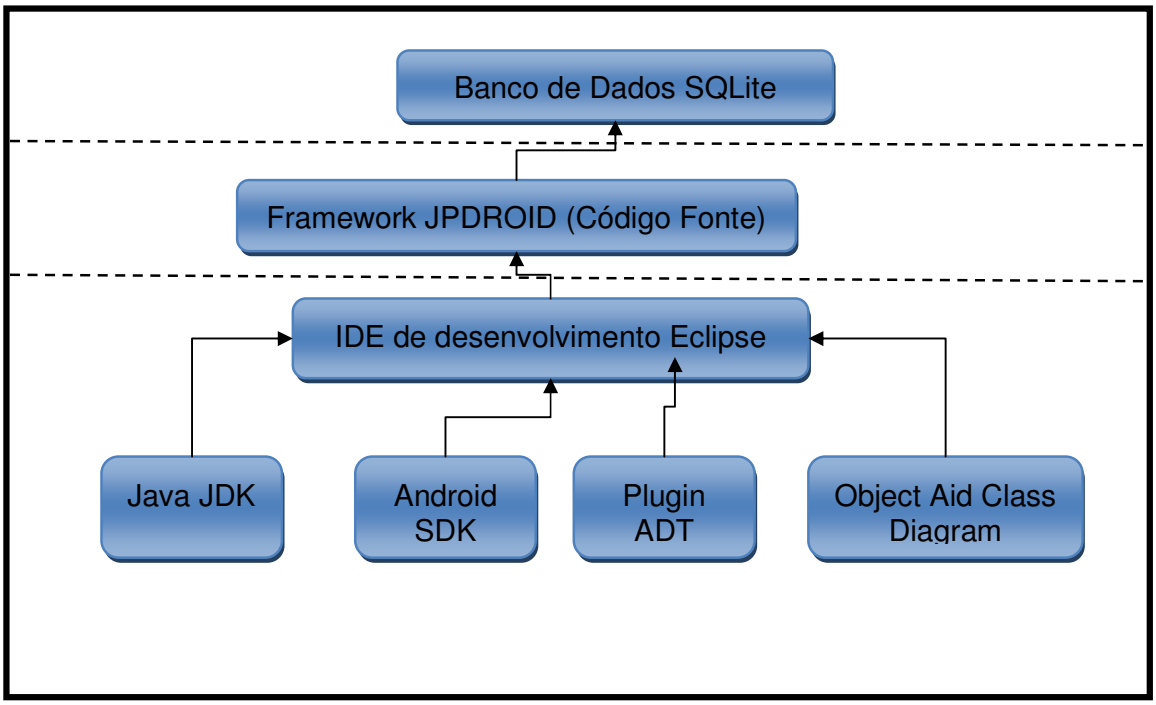

Figura 2 - Diagrama de Ferramentas
Fonte: Autoria Própria

Conforme apresentado na Figura 2, observa-se que o banco de dados SQLite é acessada exclusivamente pelo *framework* resultado do presente trabalho – o JPDroid, este foi desenvolvido usando a IDE de desenvolvimento Eclipse, que usa recursos presente no Java JDK, Android SDK, Plugin ADT e Object Aid Class Diagram. As características de cada ferramenta serão apresentadas na sequência:

a) Java JDK: O JDK (*Java Development Kit*) é um kit de desenvolvimento Java, que disponibiliza a máquina virtual Java (JVM), compilador, APIs e ferramentas utilitárias. A IDE Eclipse requer que o Java SDK esteja previamente configurado, assim como o Android SDK. (JAVA, 2014);

b) Android SDK: O Android SDK (*Software Development Kit*) é um kit de desenvolvimento que disponibiliza ferramentas para criação e teste dos aplicativos Android. Através desta SDK é possível depurar códigos em um dispositivo real ou virtual. (ANDROID, 2014);

c) Plugin ADT: O ADT (*Android Development Tools*) é um *plugin* que foi projetado para ampliar os recursos do Eclipse, desta forma é possível criar projetos Android com o suporte das ferramentas do Android SDK. (Android, 2014);

d) IDE Eclipse: O Eclipse é uma plataforma de desenvolvimento de *software*, livre e extensível, baseada em Java, que proporciona um ambiente de desenvolvimento compatível com diversas plataformas. O ambiente pode ser customizado através de *plugins* que podem ser facilmente instalados através do Eclipse Marketplace. O Projeto Eclipse foi originalmente criado pela IBM em novembro de 2001. A Eclipse Foundation foi criada em janeiro de 2004 como uma organização sem fins lucrativos e independente com o objetivo de gerenciar o desenvolvimento contínuo pela comunidade Eclipse.(IBM, 2014);

e) Banco de Dados SQLite: O SQLite (2014) é um banco de dados relacional que está disponível nativamente na plataforma Android, portanto não requer instalação ou configuração;

f) *Object Aid Class Diagram*: O ObjectAid (2014) é um *plugin* para visualização de diagramas UML gerados e atualizados a partir das classes do projeto. Através dos diagramas gerados pelo *plugin* é possível navegar entre os diagramas e o código fonte.

4 APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS

Este capítulo apresenta em detalhes as etapas referentes à construção do *framework*.

4.1 ANALISE DO DOMÍNIO

De acordo com o problema levantado neste trabalho foram analisadas e implementadas pequenas aplicações utilizando *frameworks* conhecidos como ORMLite (2014) e GreenDAO (2014).

Através de aplicações de teste e leitura de vários tutoriais foi possível assimilar as abstrações existentes, requisitos pendentes, métodos de configuração e funcionalidades, que serão listadas a seguir:

- Configuração por anotações (Mapeamento Objeto Relacional);
- Encapsulamento das opções básicas CRUD;
- Recuperação de objetos em cascata;
- Conversão de Objetos;
- Importação e Exportação de Dados;
- Controle de Transações.

Assim, estas funcionalidades foram levadas em consideração para o desenvolvimento do *framework* sugerido.

4.2 PROJETAR ABSTRAÇÕES

A partir do levantamento efetuado na primeira etapa, as características referentes à configuração, armazenamento e manipulação de dados foram projetadas para exigir o mínimo de configuração, basicamente o desenvolvedor precisará adicionar a biblioteca JPDROID em seu projeto e mapear as tabelas.

Para encapsular e centralizar o acesso às funcionalidades do *framework* foi utilizado o padrão de projeto *Singleton*. De acordo com Gamma at. al. (2000, p.130) este padrão garante a existência de apenas uma única instância de uma determinada classe, mantendo um ponto global de acesso ao objeto.

4.2.1 Arquitetura

A Figura 3 apresenta uma visão geral da estrutura de classes do *framework* desenvolvido. Para melhor visualização os atributos e métodos foram omitidos.

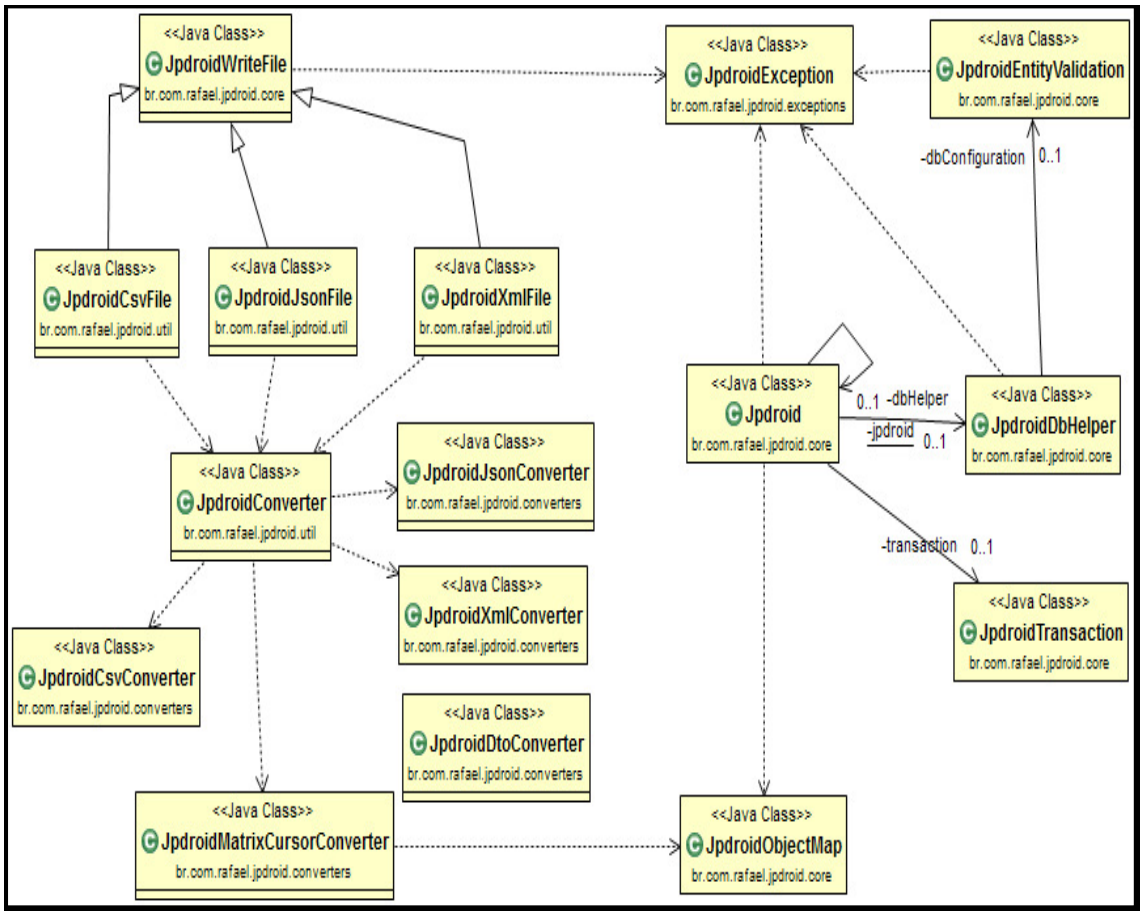

Figura 3 - Diagrama de Classes do Framework.
Fonte: Autoria Própria

Os detalhamentos das classes serão apresentados no decorrer deste trabalho.

O *framework* proposto depende basicamente de duas APIs Java, a Java *Annotation* e Java *Reflection*, o mapeamento através das anotações pode conter parâmetros que definem ou modificam o funcionamento do *framework*.

4.2.2 Classes Camada de Persistência

A camada de persistência é composta por seis classes principais. A Figura 4 apresenta em detalhes todos os métodos e atributos destas classes e como elas se relacionam.

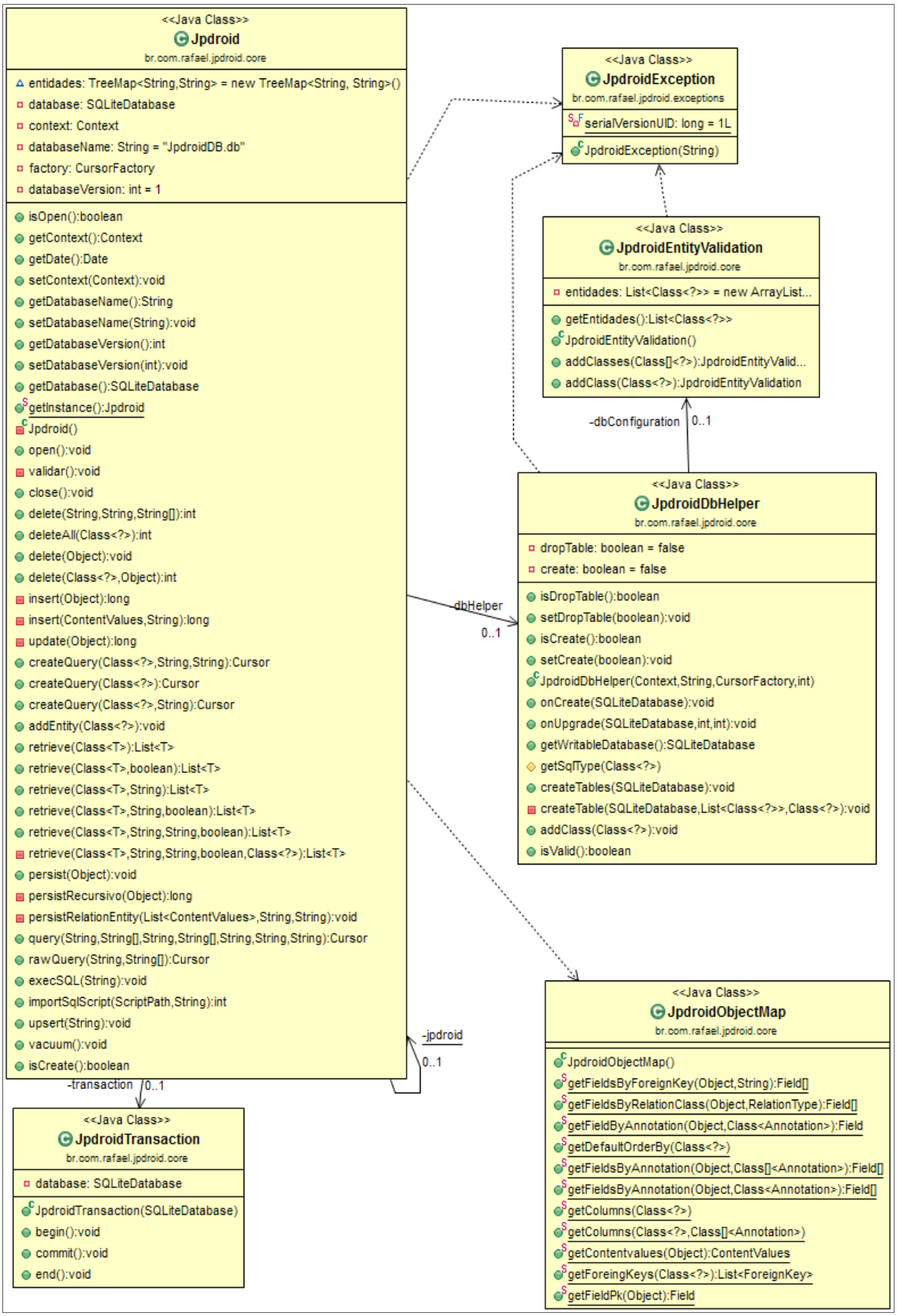

Figura 4 - Diagrama de Classes Persistência.
Fonte: Autoria Própria

A seguir as classes da Figura 4 serão detalhadas.

a) Jpdroid: O mecanismo de persistência depende da classe *singleton* chamada Jpdroid, esta classe é responsável por encapsular todas as operações. Uma vez instanciado e configurado, o objeto poderá ser invocado de forma global no aplicativo;

b) JpdroidTransaction: Permite o controle das transações, implementando comandos como *Commit* e *RollBack*, por padrão ao executar os métodos “persist()” e “delete()” o framework inicia uma transação, efetua *commit* ao concluir e *rollback* caso ocorra erro;

c) JpdroidDbHelper: Responsável pela criação do banco e tabelas conforme mapeamento realizado;

d) JpdroidObjectMap: Encapsula métodos estáticos responsáveis por obter dados dos objetos mapeados;

e) JpdroidEntityValidation: Responsável por validar o mapeamento objeto relacional;

f) JpdroidException: As exceções disparadas pelas validações do *framework* serão lançadas pela classe JpdroidException.

4.2.3 Classes Conversão de Objetos

Além das operações de banco, o *framework* proporciona que os objetos mapeados sejam convertidos para quatro formatos: Xml, Csv, Json e MatrixCursor. Os objetos convertidos também podem ser gravados em arquivos no dispositivo móvel, com exceção do formato “MatrixCursor” que é utilizado apenas para apresentação dos dados em listas. A Figura 5 apresenta todas as classes responsáveis por esta funcionalidade:

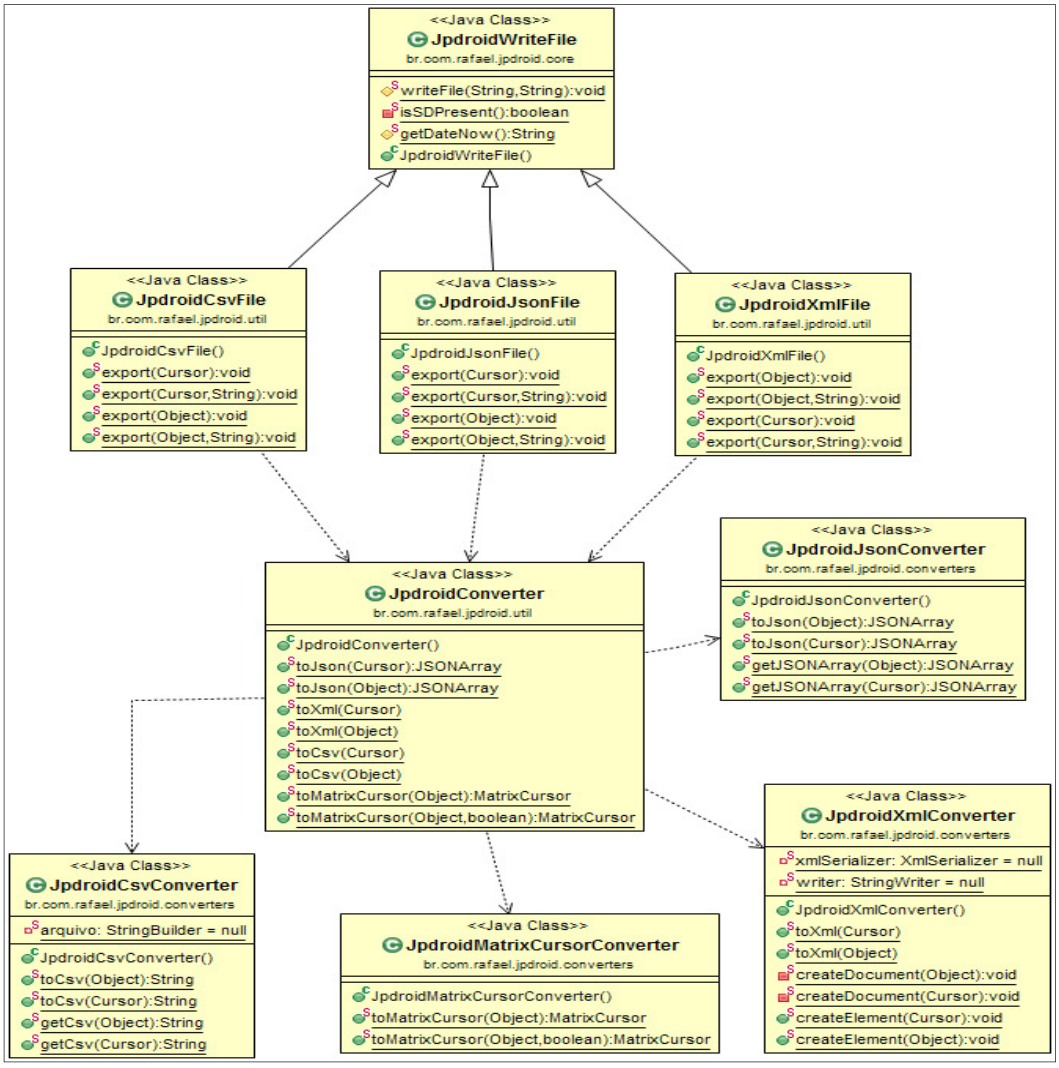

Figura 5 - Diagrama de Classes Exportação
Fonte: Autoria Própria

A seguir as classes da Figura 5 serão detalhadas.

a) JpdroidConverter: Encapsula métodos estáticos responsáveis por realizar as conversões de objeto para os formatos: Xml, Json, Csv e MatrixCursor;

b) JpdroidCsvConverter: Responsável por converter objetos para o formato Csv;

c) JpdroidJsonConverter: Responsável por converter objetos para o formato Json;

d) JpdroidXmlConverter: Responsável por converter objetos para o formato Xml;

e) JpdroidMatrixCursorConverter: Responsável por converter objetos configurados como entidade para MatrixCursor;

f) JpdroidWriteFile: Responsável por escrever os objetos convertidos em arquivos no cartão de memória;

g) JpdroidCsvFile: Encapsula métodos para exportação de objetos convertidos em Csv para arquivo no cartão de memória;

h) JpdroidJsonFile: Encapsula métodos para exportação de objetos convertidos em Json para arquivo no cartão de memória;

i) JpdroidXmlFile: Encapsula métodos para exportação de objetos convertidos em Xml para arquivo no cartão de memória.

4.2.4 Anotações JPDroid

As anotações presentes no *framework* permitem realizar o mapeamento objeto relacional de forma semelhante a implementações existentes da especificação JPA (*Java Persistence API*). ORACLE (2014)

A seguir será apresentada uma lista com as anotações presentes no *framework* e uma breve descrição:

a) @Entity: Para funcionamento do *framework* é necessário o mapeamento de uma classe com uma tabela no banco de dados. Esta é a primeira anotação a ser definida, pois ela é responsável por identificar o mapeamento da tabela na aplicação;

b) @PrimaryKey: Toda classe mapeada como tabela deve possuir obrigatoriamente um atributo como sendo chave-primária, e esta anotação é responsável por identificar o atributo como chave-primária;

c) @ForeignKey: Quando existir uma associação entre classes que caracterize uma chave estrangeira no banco o atributo deve receber a anotação @Foreignkey, esta anotação será responsável por realizar este mapeamento;

d) @Column: Todo atributo da classe mapeada que corresponda a uma coluna no banco de dados deverá ser identificada por esta anotação;

e) @RelationClass: Identifica a associação entre classes, com a presença desta configuração o *framework* é capaz de realizar operações em cascata, como recuperação, inclusão, exclusão e atualização de registros;

f) @ViewColumn: Havendo a necessidade de apresentar algum dado referente a outra tabela relacionada, o *framework* permite apresenta-lo desde que exista associação correspondente a este atributo identificado pela anotação @ForeignKey;

g) @Ignorable: Identifica atributo para não ser exportado para arquivos;

h) @Dto: Identifica classe como objeto de transferência de dados. Quando necessário converter a classe do modelo de dados em um objeto de transferência, este novo objeto deve ser anotado como DTO (*Data Transfer Object*), esta anotação será utilizada pelo método “convert()” da classe JpdroidDtoConverter;

i) @DtoField: Identifica atributo da classe Dto como coluna correspondente ao modelo mapeado;

j) @DefaultOrder: Define ordenação padrão dos objetos pelo atributo anotado, esta anotação será utilizada ao recuperar objetos através do método retrieve().

4.2.5 Tipos de Dados

Ao realizar o mapeamento é importante analisar o tipo do atributo. O banco de dados SQLite possui poucos tipos de dados (NULL, INTEGER, REAL, TEXT e BLOB). Para estender a utilização de outros tipos foi realizado um mapeamento dos tipos da linguagem Java para os tipos do banco SQLite, de acordo com o Quadro 5 o *framework* realiza as conversões necessárias de forma transparente para o desenvolvedor.

Java	SQLite
String	TEXT
Boolean	TEXT
Date	TEXT
Calendar	TEXT
Double	REAL
Float	REAL
Integer	INTEGER
Long	INTEGER
Short	INTEGER
Byte[ ]	BLOB
Bitmap	BLOB

Quadro 5 - Tipos de Dados JAVA x SQLite
Fonte: Autoria Própria

O método de configuração por anotações aplicado neste trabalho foi baseado na API de persistência JPA que está disponível à comunidade pela ORACLE (2014), no entanto foi desenvolvido de forma a atender as necessidades especificas do *framework* Jpdroid.

O Quadro 6 contém um exemplo da configuração para mapeamento de uma tabela. Para simplificar foram omitidos os métodos *getters* e *setters*, métodos que por convenção utilizam o prefixo '*get*' para obter o valor de uma variável e '*set*' para atribuir valor de uma determinada variável.

```
1.  @Entity
2.  public class Pessoa {
3.    @PrimaryKey
4.    @Column
5.    private long _id;
6.    @DefaultOrder(order=Order.asc)
7.    @Column
8.    private String nome;
9.    @RelationClass(relationType=RelationType.ManyToOne,
          joinColumn="idPessoa")
10.   private List<Endereco> endereco;
11.   @Column
12.   private Bitmap foto;
13.   @RelationClass(relationType=RelationType.ManyToOne,
        joinColumn="idPessoa")
14.   private List<Contato> contato;
15. }
```

Quadro 6 - Exemplo Mapeamento JPDroid.
Fonte: Autoria Própria

Detalhamento da Listagem do Quadro 6:

1. Anotação responsável por mapear a classe como tabela;

2. Declaração da classe Pessoa;

3. Identifica atributo como chave-primária;

4. Identifica atributo como coluna;

5. Atributo correspondente à coluna no banco;

6. Define a ordem padrão durante a recuperação de objetos. Neste caso ao recuperar uma lista do objeto “Pessoa”, está será ordenada atributo nome na ordem alfabética;

7. Identifica atributo como coluna;

8. Atributo correspondente à coluna no banco;

9. Identifica atributo como classe associada, neste caso a cardinalidade é ManyToOne pois existe vários endereços para mesma pessoa, e a coluna que faz esta relação é a coluna “idPessoa” que está na tabela “Endereco”;

10. Atributo responsável por manter uma lista de endereços;

11. Identifica atributo como coluna;

12. Atributo correspondente à coluna no banco, neste caso como trata-se de um Bitmap a foto será gravada em uma coluna do tipo BLOB;

13. Identifica atributo como classe associada, neste caso a cardinalidade é ManyToOne pois existe vários contatos para mesma pessoa, e a coluna que faz esta relação é a coluna “idPessoa” que está na tabela “Contato”;

14. Atributo responsável por manter uma lista de contatos;

15. Final classe Pessoa.

4.3 TESTES COM O FRAMEWORK

Nesta seção serão apresentados detalhes do desenvolvimento de um aplicativo, cujo objetivo é testar todas as funcionalidades do *framework*.

Este aplicativo é nomeado como “PedidoJpdroid”, e propõe automatizar o processo de emissão de pedidos de vendas com as seguintes funcionalidades:

- Cadastro de Pessoas;
- Cadastro de Produtos;
- Pedidos de Venda;
- Importação e Exportação de Arquivos.

A Figura 6 apresenta a tela inicial do aplicativo.

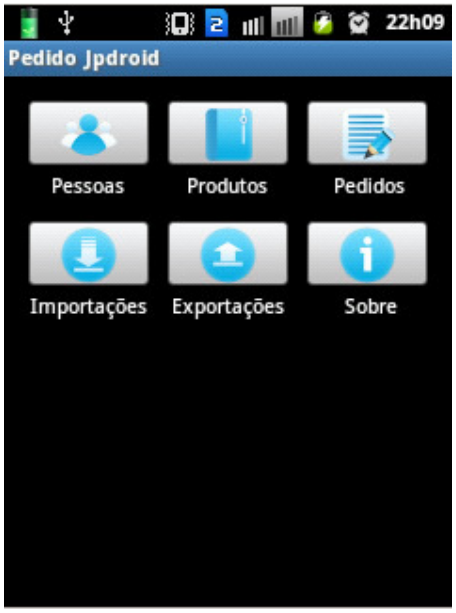

Figura 6 - Tela Inicial Aplicativo PedidoJpdroid.
Fonte: Autoria Própria

Para o desenvolvimento do aplicativo, uma série de classes de persistência e mapeamentos deve ser codificada no aplicativo. Estas são apresentadas na Figura 7.

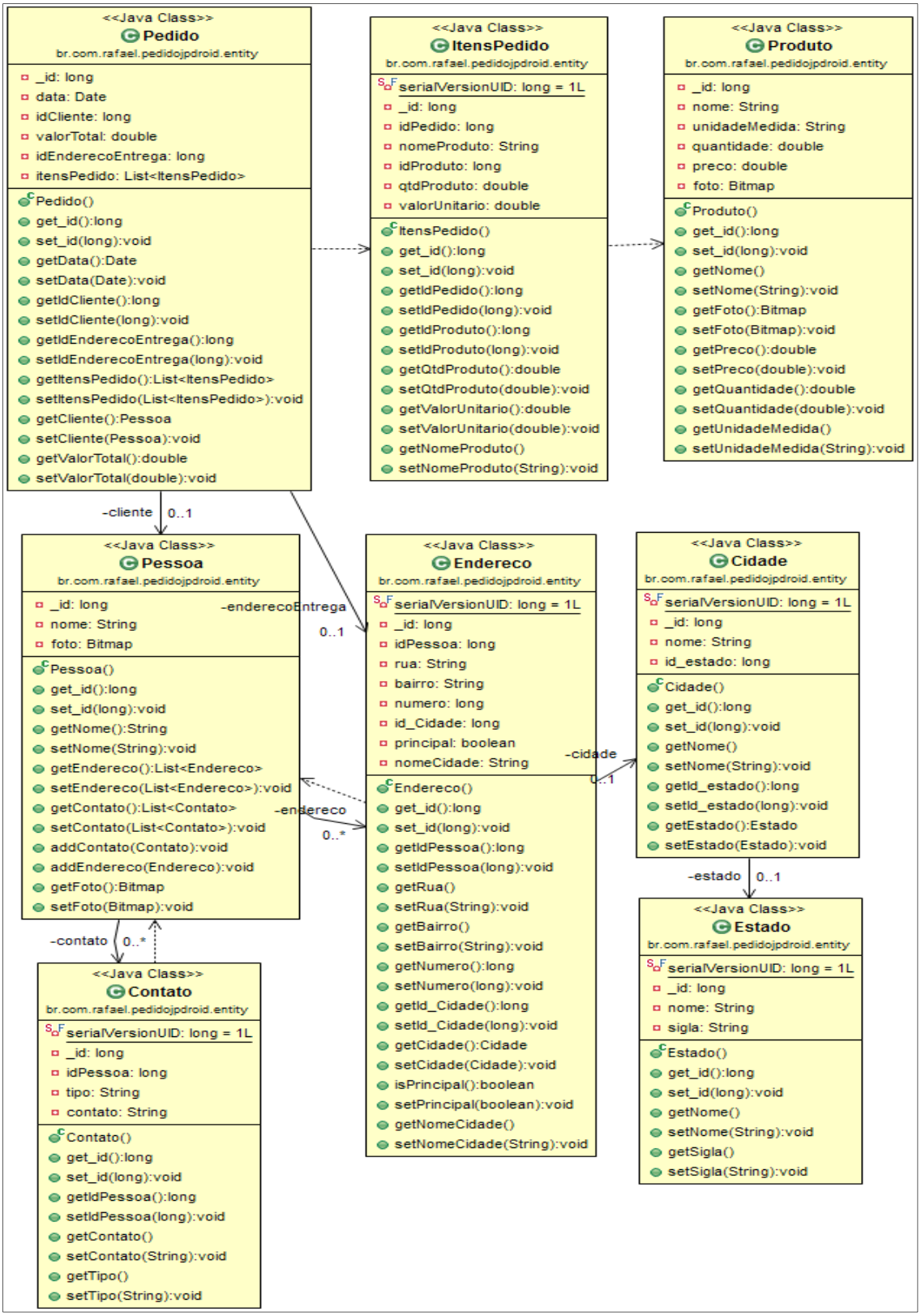

Figura 7 - Diagrama de Classes do Modelo de Dados do PedidoJpdroid.
Fonte: Autoria Própria

4.3.1 Configuração

Procedimentos como a criação do projeto, desenvolvimento das interfaces gráficas e modelo de interação com o usuário não serão detalhados, já que fogem do escopo do trabalho. O projeto completo, com códigos fonte, está disponível para *download* no repositório oficial do *framework*. JPDROID(2014).

Para iniciar o desenvolvimento utilizando o *framework* Jpdroid é necessário baixar a biblioteca e adiciona-la ao "*Build Path"* do projeto.

Para baixar a biblioteca acesse o diretório "JPDROID_Jar" que está disponível no repositório oficial do *framework*, e efetue o download do arquivo "jpdroid.jar".

Figura 8 - Repositório Jpdroid.
Fonte: Autoria Própria

Para adicionar a biblioteca ao "Build Path", primeiro é necessário colar o jar na pasta libs, conforme Figura 9:

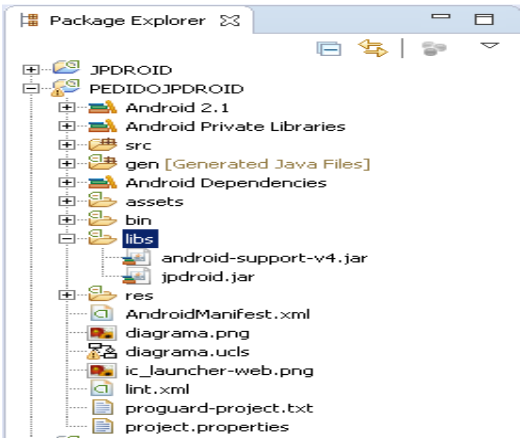

Figura 9 - Projeto PedidoJpdroid.
Fonte: Autoria Própria

Após colar a biblioteca na pasta libs, a mesma deverá ser vinculada como uma biblioteca do projeto, conforme mostra Figura 10:

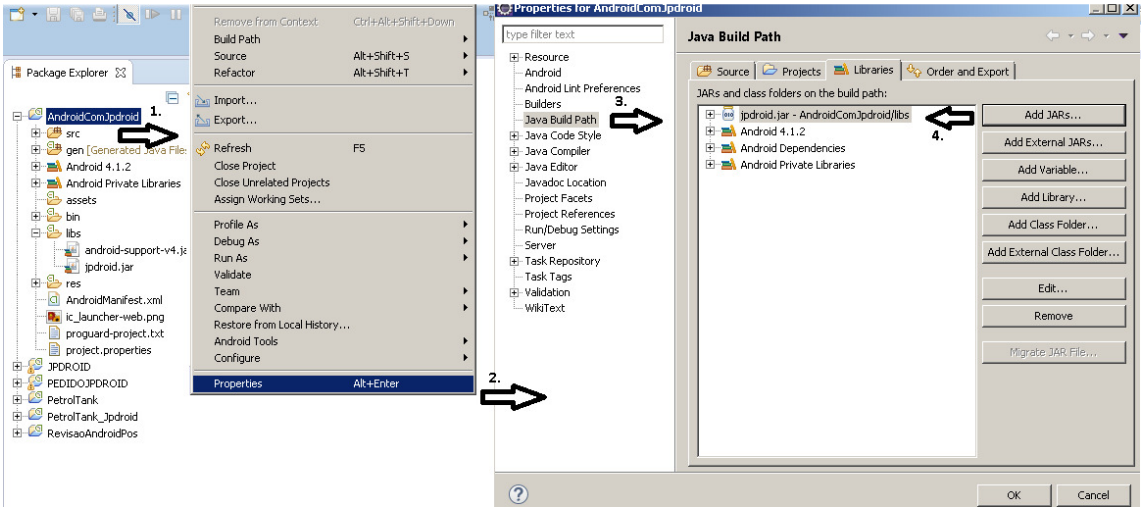

Figura 10 - Propriedades do Projeto.
Fonte: Autoria Própria.

Nas propriedades do projeto, guia “Java Build Path”, aba “Libraries”, clique no botão “Add JARs...”, localize o projeto, abra a pasta “libs” e selecione a biblioteca “jpdroid.jar”.

4.3.2 Mapeamento ORM

Após realizar esta configuração inicial, é necessário mapear as entidades utilizando as anotações oferecidas pelo *framework*.

```
1.  @Entity
2.    public class Cidade {

3.    @PrimaryKey
4.    @Column
5.    private long _id;

6.    @Column
7.    private String nome;

8.    @ForeignKey(joinEntity=Estado.class,joinPrimaryKey="_id")
9.    @Column
      private long id_estado;

10.   @RelationClass(relationType=RelationType.OneToMany,
      joinColumn="id_estado")
11.   private Estado estado;
12. }
```

Quadro 7 - Exemplo Mapeamento JPDroid.
Fonte: Autoria Própria

Detalhamento da Listagem do Quadro 7:

1. Anotação responsável por mapear a classe como tabela;
2. Declaração da classe Cidade;
3. Identifica atributo como chave-primária;
4. Identifica atributo como coluna;
5. Atributo correspondente à coluna no banco;
6. Identifica atributo como coluna;
7. Atributo correspondente à coluna no banco;
8. Define a existência de uma chave estrangeira, neste caso o atributo "id_estado" está relacionado à chave primária "_id" da tabela "Estado". O parâmetro "joinEntity" define a qual entidade a chave estrangeira corresponde, e o parâmetro "joinPrimaryKey" define qual o atributo da entidade relacionada se referência ao atributo anotado;
9. Identifica atributo como coluna;
10. Atributo correspondente à coluna no banco;
11. Identifica atributo como classe associada, neste caso a cardinalidade é OneToMay pois existe um estado para varias cidades, e a coluna que faz esta relação é a coluna "id_estado". A cardinalidade do relacionamento entre as tabelas é definida pelo parâmetro "relationType" podendo ser do tipo OneToOne, OneToMay, ManyToOne e ManyToMany. O parâmetro "joinColumn" define qual atributo da classe relacionada corresponde à chave estrangeira, e está deve ser marcada pela anotação ForeingKey.
12. Atributo responsável por manter o objeto relacionado;
13. Fim da classe Cidade.

4.3.3 Parametrização

Com as entidades mapeadas em algum ponto da inicialização do aplicativo o *framework* de persistência deve ser parametrizado e inicializado. Aconselha-se fazer isso nas *Activities* que fazem uso do banco de dados, preferencialmente no seu método "onCreate()".

```
1. Jpdroid dataBase = Jpdroid.getInstance();
2. dataBase.setContext(this);
3. dataBase.addEntity(ItensPedido.class);
4. dataBase.addEntity(Cidade.class);
5. dataBase.addEntity(Estado.class);
6. dataBase.addEntity(Endereco.class);
7. dataBase.addEntity(Pessoa.class);
8. dataBase.addEntity(Contato.class);
9. dataBase.addEntity(Produto.class);
10. dataBase.addEntity(Pedido.class);
11. dataBase.open();
```

Quadro 8 - Exemplo Parametrização do Framework JPDroid.
Fonte: Autoria Própria

Detalhamento da Listagem do Quadro 8:

1. Obtém instância da classe Jpdroid;
2. Adiciona o contexto;
3. Adiciona a entidade ItensPedido;
4. Adiciona a entidade Cidade;
5. Adiciona a entidade Estado;
6. Adiciona a entidade Endereco;
7. Adiciona a entidade Pessoa;
8. Adiciona a entidade Contato;
9. Adiciona a entidade Produto;
10. Adiciona a entidade Pedido;
11. Abre conexão com o banco de dados, neste momento o *framework* irá criar o banco assim como as tabelas mapeadas e adicionadas nesta configuração inicial.

4.3.4 Persistência

A persistência de objetos ocorre através do método “persist()”, este método é responsável por atualizar e inserir registros.

```
1. Jpdroid dataBase = Jpdroid.getInstance();
2. Pessoa pessoa = new Pessoa();
3. pessoa.setNome(etNome.getText().toString());
4. pessoa.setFoto(loadBitmapFromView(ivFoto));
5. pessoa.setContato(contato);
6. pessoa.setEndereco(endereco);
7. dataBase.persist(pessoa);
```

Quadro 9 - Exemplo Persistência de Objeto.
Fonte: Autoria Própria

Detalhamento da Listagem do Quadro 9:

1. Obtém instância da classe Jpdroid;
2. Cria instância da classe Pessoa;
3. Atribui para o atributo nome o valor digitado no Edit Text;
4. Atribui para o atributo foto o valor contido no ImageView, o método “loadBitmapFromView()” é responsável por obter o Bitmap carregado no ImageView;
5. Atribui para o atributo contato uma lista de contatos previamente preenchida;
6. Atribui para o atributo endereço uma lista de endereços previamente preenchida;
7. Persiste os objetos no banco de dados.

4.3.5 Consulta Nativa

A recuperação de dados pode ocorrer por meio de consultas SQL, esta funcionalidade está disponível através dos métodos “createQuery()”, “query()” ou “rawQuery()”, onde o retorno será um Cursor.

```
1. Jpdroid dataBase = Jpdroid.getInstance();
2. Cursor retorno1 = dataBase.rawQuery("SELECT * FROM CIDADE”,null);
3. Cursor retorno2 = dataBase.createQuery(Cidade.class,”nome = ‘Barracao’”);
4. Cursor retorno3 = dataBase.query("CIDADE", new String[] { "CIDADE._ID", "CIDADE.NOME"
   }, "CIDADE._ID=?",new String[] { String.valueOf(1) }, null, null, null, null);
```

Quadro 10 - Exemplo Consulta Nativa
Fonte: Autoria Própria

Detalhamento da Listagem do Quadro 10:

1. Obtém instância da classe Jpdroid;
2. Invoca o método “rawQuery()” passando como parâmetro uma instrução SQL nativa. O resultado será um cursor com a lista de cidades cadastradas;
3. Invoca o método “createQuery()” passando como parâmetro a entidade que deseja consultar e uma restrição. O resultado será um cursor onde o atributo “nome” seja igual a “Barracao”;
4. Invoca o método “query()” passando por parâmetro o nome da tabela, a lista de atributos e uma restrição. O resultado será um cursor onde o atributo “_id” seja igual a 1.

4.3.6 Recuperação de Objetos

Para recuperação de objetos o *framework* possui o método “retrieve()”, este método permite recuperar objetos em cascata, por exemplo a entidade “Pessoa”, possui as entidades “Contato” e “Endereco”como classes relacionadas, ao invocar o método passando o valor “true” para o parâmetro “fillRelationClass” estes objetos serão carregados automaticamente. Também é possível informar uma restrição para recuperar objetos.

```
1. Jpdroid dataBase = Jpdroid.getInstance();
2. List<Pessoa> listaPessoa = dataBase.retrieve(Pessoa.class,true);
3. List<Pessoa> listaRestrita  = dataBase.retrieve(Pessoa.class,"_id = 1", true);
```

Quadro 11 - Exemplo Recuperação de Objetos.
Fonte: Autoria Própria

Detalhamento da Listagem do Quadro 11:

1. Obtém instância da classe Jpdroid;
2. Recupera todos os objetos do tipo Pessoa e seus dependentes, como por exemplo, contatos e endereços;
3. Recupera o objeto do tipo Pessoa onde o atributo “_id” seja igual a 1.

4.3.7 Exclusão de Dados

A exclusão de registros ocorre através dos métodos “delete()” e “deleteAll()”. Uma exclusão pode ser seletiva, basta passar os parâmetros de restrição.

```
1. Jpdroid dataBase = Jpdroid.getInstance();
2. dataBase.delete(Pessoa.class, "_id = 1");
3. dataBase.deleteAll(Pessoa.class);
```

Quadro 12 - Exemplo Exclusão de Objetos.
Fonte: Autoria Própria

Detalhamento da Listagem do Quadro 12:

1. Obtém instância da classe Jpdroid;
2. O método “delete()” irá excluir o registro onde a chave primária seja igual a um;
3. O método “deleteAll()” irá excluir todos os registros da entidade Pessoa.

4.3.8 Importação de Dados

Ao iniciar o aplicativo pela primeira vez, o banco de dados pode necessitar de uma carga inicial de dados, como por exemplo, cidades e estados. O Jpdroid possui um método chamado “importSqlScript()”, este método é capaz de importar e executar scripts SQL, estes arquivos podem estar armazenados na pasta Assets do projeto Android ou no cartão de memória do dispositivo.

```
1. if(dataBase.isCreate()){
2.    dataBase.importSqlScript(ScriptPath.Assets, "import.sql");
3. }
```

Quadro 13 - Exemplo Importação Dados.
Fonte: Autoria Própria

Detalhamento da Listagem do Quadro 13:

1. Quando o aplicativo é iniciado pela primeira vez, o *framework* irá criar a base de dados, que por padrão será armazenado no diretório “/data/data/<nome-do-pacote>/databases/nome-do-arquivo.db”, o método “isCreate()” retorna “*true*” neste momento, caso a base já exista retorna “*false*”;

2. O método “importSqlScript()” irá importar o arquivo “import.sql” que está na pasta “Assets” e executa os comandos SQL para manipulação dos dados.

4.3.9 Exportação de Dados

A exportação de dados em arquivos pode ocorrer em três formatos disponíveis: Xml, Json e Csv.

```
1. JpdroidCsvFile.export(dataBase.retrieve(Pedido.class, true), "PedidoExport.csv");
2. JpdroidXmlFile.export(dataBase.retrieve(Pedido.class, true), "PedidoExport.xml");
3. JpdroidJsonFile.export(dataBase.retrieve(Pedido.class, true),"PedidoExport.json");
```

Quadro 14 - Exemplo Exportação de Dados.
Fonte: Autoria Própria

Detalhamento da Listagem do Quadro 14:

1. Exporta para o cartão de memória um arquivo csv chamado “PedidoExport.csv” contendo uma lista de pedidos que foram recuperados pelo método “retrieve()”;

2. Exporta para o cartão de memória um arquivo xml chamado “PedidoExport.xml” contendo uma lista de pedidos que foram recuperados pelo método “retrieve()”;

3. Exporta para o cartão de memória um arquivo json chamado “PedidoExport.json” contendo uma lista de pedidos que foram recuperados pelo método “retrieve()”.

4.3.10 Conversão de Dados

A simples conversão de objetos sem a escrita de arquivos pode ser feita através da classe JpdroidConverter.

```
1. JSONArray json = JpdroidConverter.toJson(dataBase.retrieve(Pedido.class));
2. String xml = JpdroidConverter.toXml(dataBase.retrieve(Pedido.class));
3. String csv = JpdroidConverter.toCsv(dataBase.retrieve(Pedido.class));
```

Quadro 15 - Exemplo Conversão de Objetos.
Fonte: Autoria Própria

Detalhamento da Listagem do Quadro 15:

1. Converte para json uma lista de pedidos e atribui a variável json do tipo JSONArray;

2. Converte para xml uma lista de pedidos e atribui a variável xml do tipo string;

3. Converte para csv uma lista de pedidos e atribui a variável csv do tipo string.

4.3.11 Regras

É importante resaltar que o *framework* possui algumas regras, como por exemplo, toda chave, seja ela primária ou estrangeira deve ser do tipo primitivo long.

Toda entidade deve possuir obrigatoriamente um atributo “_id” do tipo long e anotado com @Column e @PrimaryKey.

Quando ocorrer algum erro de configuração a classe JpdroidEntityValidation irá disparar exceções que poderão ser consultadas no “Log Cat”, conforme mostrado na Figura 11.

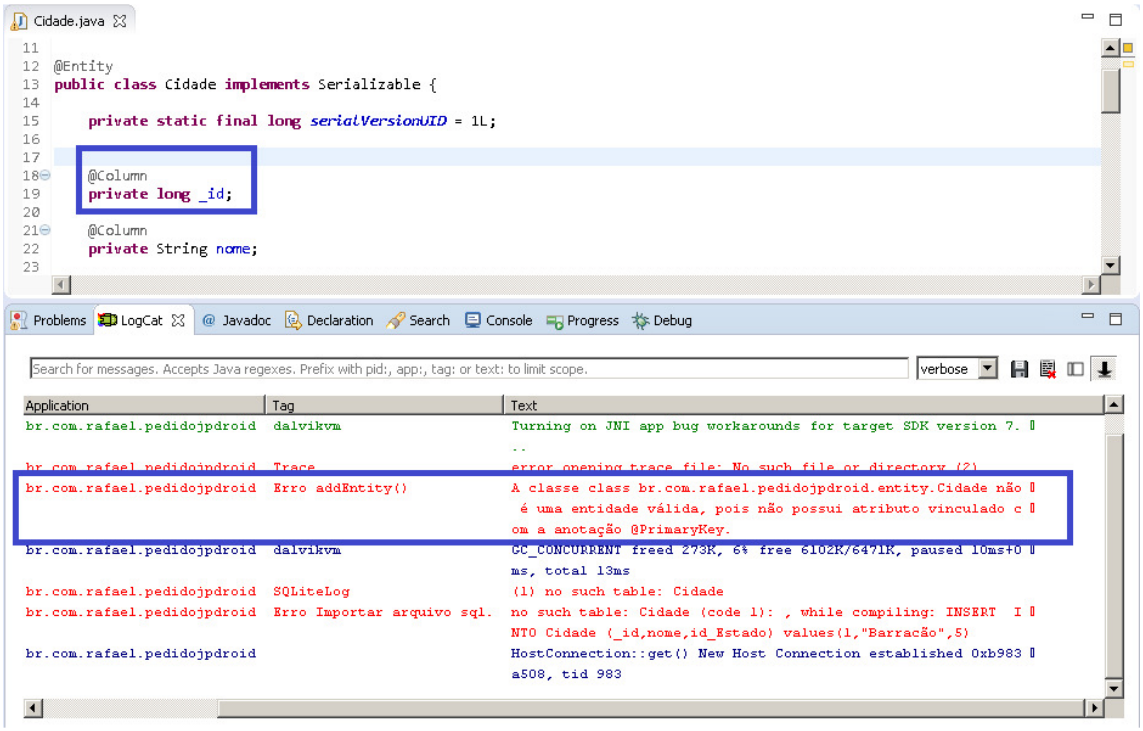

Figura 11 - Validações JpdroidEntityValidation
Fonte: Autoria Própria

Para exemplificar a validação realizada pelo *framework*, foi removida a anotação @PrimaryKey da entidade Cidade, ao executar aplicativo uma exceção foi lançada, esta pode ser consultada no “Log Cat” do Eclipse, conforme destacado na Figura 11.

5 CONSIDERAÇÕES FINAIS

Neste trabalho foi desenvolvido um *framework* de mapeamento objeto relacional para plataforma Android. O *framework* desenvolvido chama-se “JPDroid” e realiza operações de persistência no banco SQLite através de um mecanismo de persistência simples e transparente para o desenvolvedor. Diferenciando-se dos demais *frameworks* existentes pelo tamanho da biblioteca, apenas 49kb, simplicidade na utilização e pelas funcionalidades de exportação e importação de dados. Também disponibiliza um conjunto de anotações para o mapeamento dos objetos e suporte a recuperação de objetos em cascata.

Como etapa final do projeto foi desenvolvido um aplicativo chamado “PEDIDOJPDROID”, o objetivo deste projeto é testar todas as funcionalidades do *framework*, como manipulação e consulta de múltiplas tabelas, relacionamentos, exportação e importação de dados.

O código fonte do *framework* e do aplicativo de teste estão disponíveis no formato *Open-Source* para *download* no repositório oficial. (JPDROID, 2014).

Para proteger os direitos autorais e garantir que a distribuição continue livre o JPDroid é distribuído sob licença GPL v3. Esta licença exige que ao distribuir cópias, as mesmas liberdades sejam mantidas. Deve-se garantir também a distribuição dos fontes ou como adquiri-los. Os termos desta mesma licença devem acompanhar o código fonte, para que todos conheçam seus direitos. De uma forma resumida a licença concede permissão legal para copiar, distribuir e/ou modificar. (GNU, 2014).

O *framework* desenvolvido pode ser utilizado em diversos tipos de aplicativos que necessitem de um mecanismo de persistência, como por exemplo, sistemas para pesquisa de mercado, coletores de dados, vendas porta á porta, jogos, entre outros.

A utilização do *framework* pode colaborar com o desenvolvedor nos seguintes aspectos:

Produtividade: Como toda lógica para criação do banco e manipulação de dados já está implementada na biblioteca há uma redução em linhas de código. Permitindo assim o desenvolvedor criar aplicações de forma mais ágil.

Qualidade: Ao atribuir a responsabilidade ao framework para gerar o banco e manipular os dados é possível evitar erros de programação, pois não será necessário implementar comandos DML (Data Manipulation Language).

5.1 PROBLEMAS ENCONTRADOS

Durante o desenvolvimento do *framework*, foram encontradas algumas dificuldades, como por exemplo, o uso de reflexão, mas graças às pesquisas realizadas durante o desenvolvimento do *framework*, o conhecimento adquirido tem ajudando também em outros projetos fora do mundo acadêmico.

Algumas situações como, atribuir valor via reflexão a atributos do tipo Long, ocorre exceção, esta situação está presente somente no Android, por este motivo toda chave, primaria ou estrangeira deve ser do tipo primitivo long.

Outro problema foi à necessidade de obrigar o desenvolvedor criar o atributo "_id", esta restrição foi necessária, pois o Android exige este atributo nas listagens que utilizam *CursorAdapter*.

Outra situação que não pôde ser resolvida foi à validação do mapeamento objeto relacional, atualmente as validações ocorrem somente em tempo de execução, é necessário consultar o "*Log Cat" do Eclipse* para visualizar as exceções, interessante seria realizar as validações em tempo de desenvolvimento.

5.2 TRABALHOS FUTUROS

Para trabalhos futuros, sugiro o desenvolvimento das seguintes funcionalidades:

a) Carregamento Preguiçoso (*Lazy Loading*): Ao invocar métodos *getters* referentes às classes relacionadas, o *framework* deverá se encarregar de recuperar os objetos no banco;

b) Exportação de dados diretamente para um serviço rest: Desenvolver uma classe que permita através de uma simples parametrização o envio de objetos para um serviço Rest (*Representational State Transfer*);

c) Desenvolver uma linguagem de consulta orientada a objetos: Um exemplo conhecido é o Críteria do Hibernate.

REFERÊNCIAS

ABLESON, W. F. et al. **Android em Ação.** 3.ed. Rio de Janeiro: Elsivier, 2012.

ANDROID. Disponível em: http://developer.android.com/images/system-architecture.jpg. Acesso em: 11 Jun. 2014.

BAUER, Cristian; KING, Gavin. **Hibernate In Action**. MANNING, 2004.

BORAKS, Silvio. **Programação com Java: Uma Introdução Abrangente**. Porto Alegre: BookMan, 2013.

GAMMA, E. R. HELM; JOHNSON, R.; VLISSIDES J. **Padrões de Projeto:** Soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

GERRA, Eduardo. **Componentes reutilizáveis em Java com Reflexão e Anotações**. São Paulo: Casa do Código, 2013.

GERRA, Eduardo. **Design Patterns com Java:** Projeto orientado a objetos guiado por padrões. São Paulo: Casa do Código, 2013.

GNU, **General Public Licence**. Disponível em: http://www.gnu.org/copyleft/gpl.html Acesso em: 18 Ago. 2014.

GREENDAO. Disponível em: http://greendao-orm.com/. Acesso em: 15 Jan. 2014.

HIBERNATE. Disponível em: http://hibernate.org/. Acesso em: 28 Ago.2014.

IBM. **Introdução a Plataforma Eclipse**. Disponível em: http://www.ibm.com/developerworks/br/library/os-eclipse-platform/. Acesso em: 16 Jul. 2014.

JAVA. **Desenvolvendo programas Java com o JDK**. Disponível em: https://www.java.com. Acesso em:21 Jul. 2014.

JCP, **JSR 175**: **A Metadata Facility for the JavaTM Programming Language**. Disponível em: https://www.jcp.org/en/jsr/detail?id=175. Acesso em: 26 Jun. 2014.

JOHNSON, Ralph E. **HOW TO DESIGN FRAMEWORKS**, Disponível em: http://st-www.cs.illinois.edu/users/johnson/cs497/notes98/day18.pdf. Acesso em: 01 Jul. 2014.

JPDROID, Disponível em: https://github.com/RafaelCentenaro/jpdroid. Acesso em: 25 Ago. 2014.

LISBOA, Flavio G.S; MOTA, Luciana C. **Introdução ao Demoiselle Framework, SERPRO**. Disponível em: http://www.youblisher.com/p/98445-Introducao-ao-Demoiselle-Framework/. Acesso em: 30 Jan. 2014.

MATTSSON, Michael. **Evolution and Composition of Object-Oriented Frameworks**. Disponível em: http://www.bth.se/faculty/mma/papers/michael.mattsson.ph.d.thesis.pdf. Acesso em: 01 Jun. 2014.

OBJECTAID. Disponível em: http://www.objectaid.com/. Acesso em: 30 Jul. 2014.

ORACLE, **Generic Types**, Disponível em: http://docs.oracle.com/javase/tutorial/java/generics/types.html. Acesso em: 26 Jun. 2014.

ORMLITE. Disponível em: http://ormlite.com/. Acesso em: 02 Jan. 2014.

PEREIRA,L,C,O;SILVA,M,L. **Android para Desenvolvedores**. Rio de Janeiro: Brasport, 2009.

PRADANOV,Cleber Cristiano; Freitas, Ernani Cesar.**Metodologia do Trabalho Científico:** Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico. 2. ed. Rio Grande do Sul: Universidade Feevale, 2013.

RAMAKRISHNAN, Raghu, GEHRKE, Johannes. **Sistemas de gerenciamento de banco de dados**. McGraw Hill Brasil, 2008.

SERSOR, Roberto Rubinstein. **Programação Orientada a Objetos com Java 6 - Curso universitário**. Rio de Janeiro: Brasport, 2007.

SHALLOWAY, Alan; TROTT, James. **Design Patterns Explained. A New Perspective on Object-Oriented Design**. Ed. Addison-Wesley, 2001.

SQLITE. Disponível em: http://www.sqlite.org. Acesso em: 01 Mai. 2014.