Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis - IMESA

**ELCIO RODRIGUES DE BRITO**

# DESENVOLVIMENTO DE APLICAÇÕES COMERCIAIS EM JAVA USANDO REFLECTION

ASSIS
2012

**ELCIO RODRIGUES DE BRITO**

# DESENVOLVIMENTO DE APLICAÇÕES COMERCIAIS EM JAVA USANDO REFLECTION

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Instituto Municipal do Ensino Superior de Assis – IMESA e Fundação Educacional do Município de Assis – FEMA, como requisito para a obtenção do Certificado de Conclusão.

**ORIENTADOR:** Dr. Almir Rogério Camolesi
**Área de Concentração**: Informática

ASSIS
2012

**FICHA CATALOGRÁFICA**

BRITO, Elcio Rodrigues

Desenvolvimento de Aplicações Comerciais em Java Usando Reflection / Elcio Rodrigues de Brito. Fundação Educacional do Município de Assis – FEMA – Assis, 2012.

35p.

Orientador: Dr. Almir Rogério Camolesi

Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis – IMESA

1. Reflection, Linguagem de programação, JAVA, sistemas de informação, UML

CDD: 001.6

Biblioteca da FEMA

# DESENVOLVIMENTO DE APLICAÇÕES COMERCIAIS EM JAVA USANDO REFLECTION

**ELCIO RODRIGUES DE BRITO**

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, analisado pela seguinte comissão examinadora**.**

**Orientador:** Dr. Almir Rogério Camolesi
**Analisadora:** Esp**.** Diomara Martins Reigato Barros

ASSIS
2012

## DEDICATÓRIA

Dedico este trabalho a minha
família, que sempre me apoiou.

## AGRADECIMENTOS

Agradeço a Deus primeiramente pela força nos momentos mais difíceis.

Ao meu orientador, Dr. Almir Rogério Camolesi, pelo auxilio e confiança no desenvolvimento do trabalho.

Não posso esquecer os colegas e amigos que fiz durante o período de quatro anos e a todos os professores da FEMA que criaram a base para o meu conhecimento.

E agradecer a minha família por todo o apoio dado nesse decorrer do curso.

## RESUMO

Este trabalho apresenta o conceito de reflexão computacional usando orientação a objetos na tecnologia Java. A reflexão Computacional torna o sistema flexível, tornando o código mais reutilizável. Neste contexto, procura-se adicionar técnicas existentes e melhorar o conteúdo fazendo pesquisas e testes em aplicações comerciais, a fim de tirar resultados satisfatórios sobre o desempenho de um sistema utilizando reflexão computacional. O trabalho também tem como objetivo exemplificar através de um estudo de caso, um sistema comercial que use o conceito de reflexão computacional.



**Palavra-chave:** Reflection, linguagem de programação, JAVA, sistema, UML

## ABSTRACT

This job introduces the concept of computational reflection using object orientation in Java technology. Computational reflection makes the system flexible, making the code more reusable. In this context, it seeks to add existing techniques and to improve content doing research and testing in commercial applications in order to take satisfactory results on the performance of a system using computational reflection. The work also aims to illustrate through a case study, a commercial system that uses the concept of computational reflection.



**Keyword:** Reflection`, programming language, JAVA, system, UML

## LISTA DE FIGURAS

## LISTA DE CÓDIGOS

**SUMÁRIO**

# 1 – INTRODUÇÃO

Sistemas modernos requerem um alto grau de confiabilidade e tolerância a falhas, como por exemplo, centrais telefônicas e sistemas distribuídos. Na medida em que estes sistemas vão evoluindo, proporcionalmente o grau de complexidade evolui, porém exige dos desenvolvedores de software o conhecimento de novos conceitos e técnicas para que possam suprir as dificuldades e simplificar o desenvolvimento (Barth, F.J. 2000).

Com o conceito de reflexão computacional pode-se examinar o software em tempo de execução ou até mesmo modificar o seu comportamento. Na tecnologia Java [Tutorial Java: O que é Java, 2009] a reflexão permite a inspeção de classes, interfaces, campos e métodos em tempo de execução sem saber os nomes das interfaces, métodos e campos. Também permite a instanciação de objetos e invocação de métodos.

Refletir, segundo Silva, R.C. (1997), é a habilidade de conhecer as próprias idéias e meditar nelas, tornando-as mais aperfeiçoadas e flexíveis à situações inesperadas. De acordo com Ferreira, A. B. H. (2000), o ato de refletir se divide em dois: sendo o primeiro tal qual descrito anteriormente e o segundo como reflexo de uma superfície refletora. Esta pesquisa irá abordar o conceito de Reflexão Computacional fundamentado nestas duas vertentes.

Devido as constantes falhas de sistema em tempo de execução (Silva, R.C. 1997), têm-se havido inúmeros estudos sobre formas de gerar mais rapidez e segurança no software com o intuito de evitar tais falhas ou ao menos diminuí-las. Com a capacidade de refletir-se em si mesmo, o software usando o conceito de Reflection acarretará em uma melhoraria no seu desempenho, pois irá *manipular representações de suas próprias operações e estruturas* (Silva, R.C. 1997).

Smith, B. C. (1985), defende que se *sistemas computacionais podem ser construídos para "raciocinar" sobre o mundo externo através da manipulação de representações daquele mundo, também podem ser construídos para "raciocinar" sobre si mesmos manipulando representações de suas próprias operações e estruturas*, ou seja, antecipando-se às intempéries e conhecendo suas soluções.

## 1.2 – OBJETIVOS

O objetivo deste trabalho é apresentar o conceito de programação usando reflexão (*Reflection*) e como o mesmo é implementado em um ambiente de programação Java. Ao fim dessa pesquisa foi desenvolvido um estudo de caso utilizando reflection para demonstrar os conceitos estudados e a utilização dos mesmos no desenvolvimento de aplicações comerciais.

## 1.3 – JUSTIFICATIVA

O conceito de reflection não é novo, pois faz parte da tecnologia Java desde as suas primeiras versões (versão 1.1). Porém existem poucos estudos voltados na área de desenvolvimento de software utilizando reflection. Neste sentido este trabalho tem o intuito de mostrar o uso de reflection no desenvolvimento de software, e como os softwares poderão reagir no desempenho utilizando tais conceitos.

## 1.4 – MOTIVAÇÃO

Com o estudo de reflection na tecnologia Java abordando na área de desenvolvimento de sistemas comerciais, ajuda a fortificação do paradigma e sua aceitação pelas empresas de desenvolvimento.

## 1.5 – PERSPECTIVAS DE CONTRIBUIÇÃO

A perspectiva de contribuição é que este trabalho mostre os conceitos de reflection no desenvolvimento de software para gestão de empresas comerciais, onde possa estimular novas pesquisas relacionadas ao assunto e aumentar o campo de discussão.

## 1.6 – METODOLIA DE PESQUISA

A metodologia de pesquisa adotada para este trabalho foi a experimental. Inicialmente foram realizadas pesquisas e leituras em livros, teses, revistas, artigos, internet, aulas virtuais e anotações derivadas de discussões sobre o assunto. Posteriormente foi definido um estudo de caso com o objetivo de demonstrar a utilização destes conceitos e, por fim, implementado o estudo de caso proposto com o objetivo de aprofundar o aprendizado e de avaliar o uso das tecnologias estudadas.

## 1.7 – ESTRUTURAS DO TRABALHO

Este trabalho é constituído pelo capítulo de Introdução, seguido pelo capítulo 2 que discorre sobre os conceitos de reflexão computacional, a qual define os modelos de reflexão – arquiteturas reflexivas e conceitos básicos de orientação a objetos; o capítulo seguinte *Reflection* em tecnologia JAVA e descreve a API *Reflection*; o capítulo 4 é referente ao Estudo de Caso que demonstra a aplicação utilizando *Reflection; e* por fim, o Capítulo 5 conclui as idéias principais do trabalho.

## 2 – REFLEXÃO COMPUTACIONAL

Em desenvolvimento de sistemas comerciais, proporcionalmente o analista vai programá-lo para solucionar problemas, independente da quantidade de problemas. Esse sistema jamais vai modificar seu comportamento ou estrutura para solucionar um problema, pois ele irá realizar tudo o que foi programado sem fazer alterações em si mesmo, não permitindo que o sistema consiga se auto-analisar conforme a necessidade que aparecer.

Com base nesse modelo escrito acima, usando reflexão computacional é possível o sistema raciocinar sobre si mesmo, podendo solucionar problemas em tempo de execução, a qual sem uso de reflexão ele só vai fazer o que foi programado.

Segundo Barth, F. J. (2000), a idéia básica do paradigma de reflexão computacional não é exatamente nova. Esta idéia originou-se em lógica matemática e recentemente, mecanismos de alto nível tornam o esquema de reflexão um aliado de características operacionais ou não funcionais a módulos já existentes.

A reflexão pode ser definida como qualquer ação executada por um sistema computacional sobre si próprio, onde tal conceito foi apresentado pela primeira vez por Maes (1987), pode-se dizer que reflexão é a capacidade de um programa reconhecer detalhes internos em tempo de execução que não estavam disponíveis no momento da compilação do programa.

Reflexão Computacional tem dois significados distintos. Um é introspecção, que se refere ao ato de examinar a si próprio, o segundo é intercessão, onde está ligado ao redirecionamento da luz, ou seja, o termo reflexão computacional na área da informática tem a capacidade de examinar ou conhecer a si mesmo ou estrutura, podendo fazer alterações no seu comportamento através do redirecionamento ou de interceptação de operações efetuadas [Conceitos Básicos].

Segundo Barth, F. J. (2000), *a reflexão Computacional define em uma nova arquitetura de software. Este modelo de arquitetura é composto por um meta-nível, onde se encontram estruturas de dados e ações a serem realizadas sobre o sistema objeto*, *localizado no nível base.* Na arquitetura de meta-nível, considerando no

contexto acima é explorada para expressar propriedades não funcionais do sistema, de forma independente do domínio da aplicação.

Requisitos funcionais não apresentam nenhuma função a ser realizada pelo software, e sim comportamentos que este software deve utilizar.

A figura 1,ilustra a visão de um processo de reflexão em sistema computacional, o sistema pode ser dividido em vários níveis, o usuário envia uma informação ao sistema ou mensagem, e tratado pelo nível funcional, que é capacitado para executar a tarefa, assim o nível não funcional e responsável por realizar o gerenciamento do nível funcional.

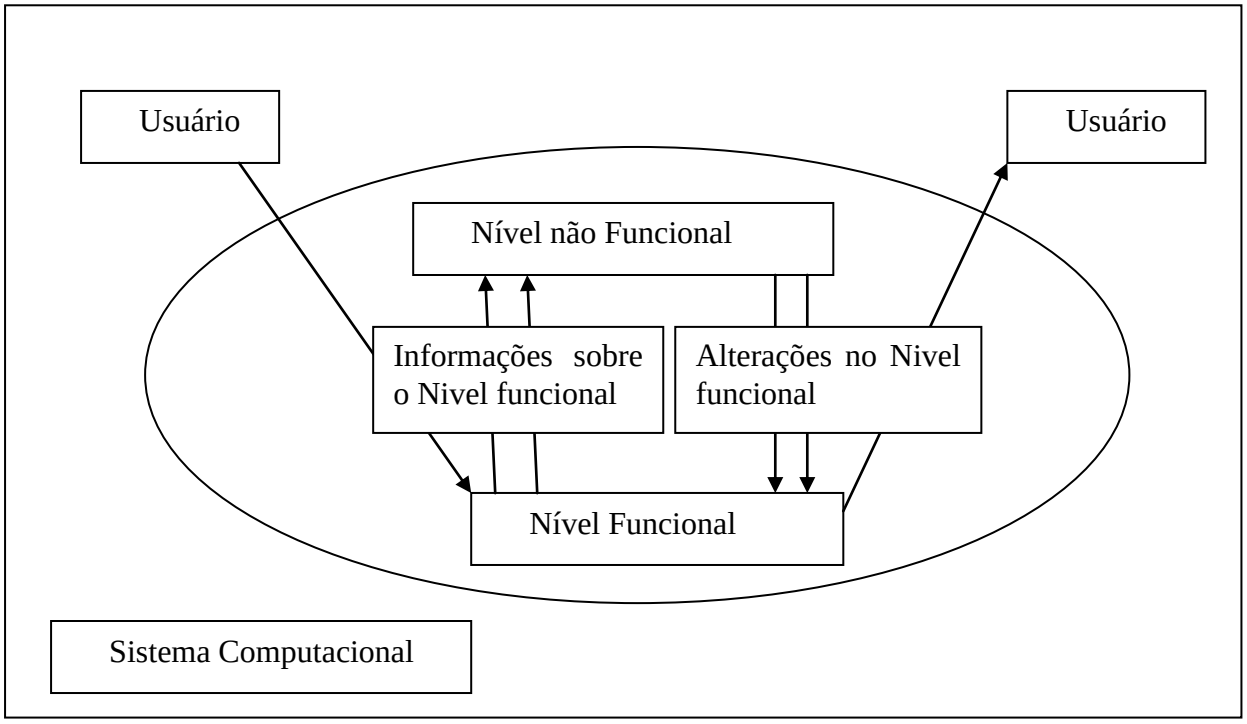



Figura 1: Visualização genérica de um sistema computacional reflexivo (Barth, F.J. 2000).

## 2.1 – MODELOS DE REFLEXÃO

A dois modelos de Reflexão Computacional, o primeiro é reflexão estrutural, pode ser definida por qualquer atividade exercida em uma metaclasse, a reflexão estrutural permite em tempo de execução fazer alterações em seus componentes no processo de execução ou compilação (Sallem, M. A. S**,** 2007).

Reflexão estrutural tem as funcionalidades de realizar algumas transformações básicas, tais como: informações sobre a classe, suas instâncias, modificar atributos e métodos de uma classe, alterar campos (Barth, F. J. 2000).

A Segunda Reflexão seria a reflexão comportamental, que leva mais a fundo sobre o comportamento de objetos, sem modificar a estrutura do objeto.

## 2.2 – ARQUITETURA REFLEXIVA

A reflexão Computacional define conceitualmente uma arquitetura em níveis, denominada arquitetura reflexiva. Em uma arquitetura reflexiva, um sistema computacional é visto como incorporando dois componentes: um representando os objetos (domínio da aplicação), e outro a parte reflexiva (domínio reflexivo ou autodomínio) [Arquitetura Reflexiva].

No nível-base são encontradas as funções dos objetos, essas computações dos objetos têm a funcionalidade de resolver problemas e retornar informações sobre o domínio externo, enquanto o nível reflexivo encontra-se no meta-nível, resolvendo os problemas do nível-base, e retorna informações sobre a computação do objeto, podendo adicionar funcionalidades extras a este objeto.

Em uma arquitetura reflexiva, tem uma visão que um sistema computacional tem incorporado uma parte objeto e outra parte reflexiva. Na parte da computação objeto é realizadas funções para resolver problemas e retornar informações sobre um domínio externo, e quanto ao nível reflexivo tem a funcionalidade de resolver problemas e retornar informação sobre a computação do objeto.

## 2.3 – ORIENTAÇÃO À OBJETOS

Orientação a Objetos é um paradigma para o desenvolvimento de software que se baseia na utilização de componentes individuais que colaboram para construir sistemas mais complexos, A colaboração entre os objetos é feita através do envio de mensagens.

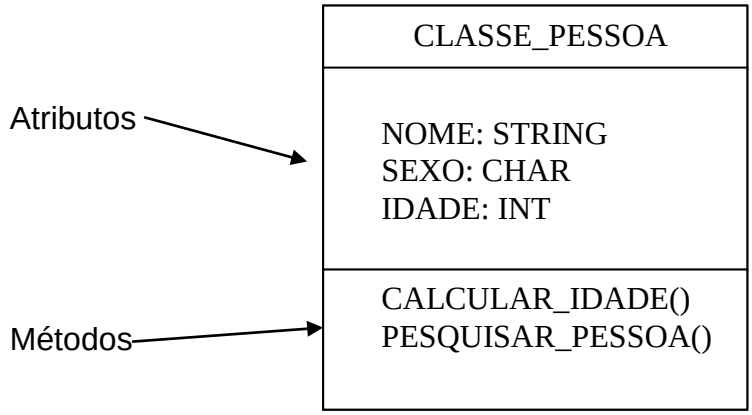
Um paradigma é um conjunto de regras que estabelecem fronteiras e descrevem como resolver problemas dentro desta fronteira. Um paradigma ajuda a organizar e coordenar a maneira como olhamos o mundo.

Orientação à objetos baseia-se nos seguintes conceitos abaixo.

### 2.3.1 - CLASSES

Classe é a descrição de um conjunto de objetos, através da definição de uma classe, descrevem-se a propriedade ou atributos que o objeto terá. Com as definições das classes é possível descrever os comportamentos que o objeto pode ter, as funcionalidades dos objetos são descritas por meio de métodos.

Na figura 2: será representado um modelo de classe, com atributos e métodos.




A Classe consiste nas definições e operações dos objetos, atributos representam as características da classe, exemplo: na figura 2, mostram os atributos que definem a descrição de uma pessoa, tais itens: nome, sexo e idade. E os métodos representam qual a ação aquela classe teria, ou seja, calcular a idade de uma pessoa ou pesquisar uma pessoa.

#### 2.3.1.1 – OBJETOS

As classes provêem a estrutura para a construção de objetos, como podem ser descritas instância de classes, as instâncias pode ser entendida por características de uma pessoa**.**

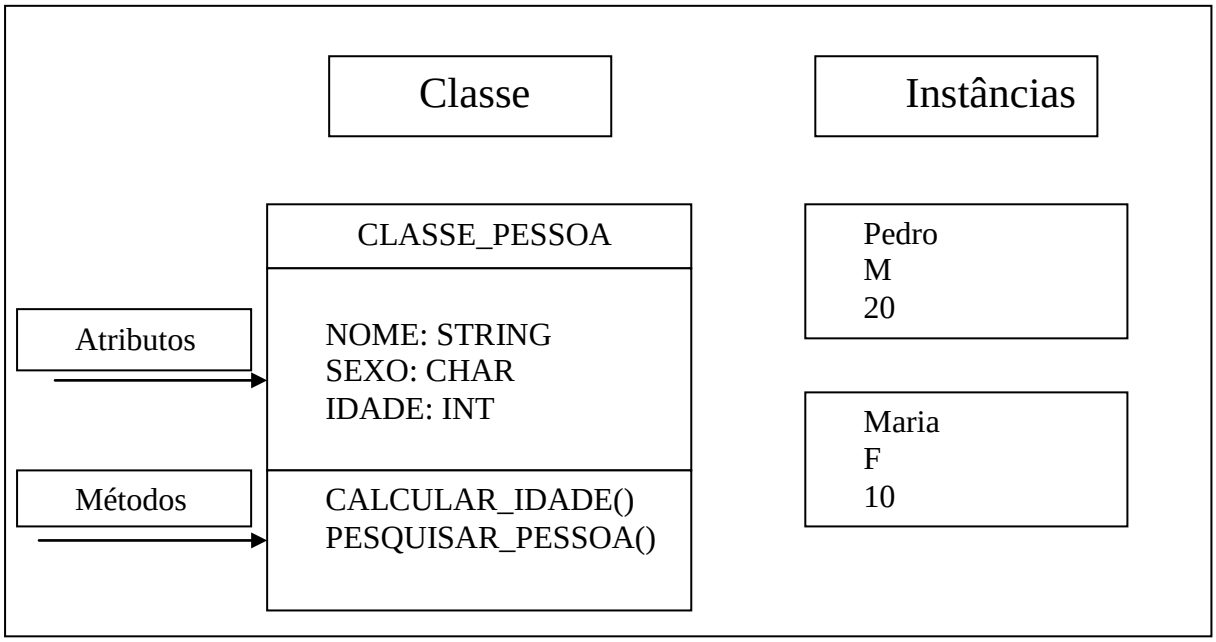



Figura 3: Representação de Objetos

As instâncias demonstradas na figura 3 são um conjunto de objetos da *classe_pessoa*, objetos consistem no estado e no comportamento, um objeto armazena seu estado em campos (variáveis ou atributos), e seu comportamento é efetuado por meio de métodos.

**2.3.1.2 – HERANÇA**

Herança no paradigma da orientação a objetos permite a reutilização de estrutura e do comportamento da classe base para redefinições de outras classes, a partir de uma classe base, ela fornece a estrutura e métodos para que classes derivadas possam herdar atributos de outra classe. Uma subclasse pode herdar atributos, métodos da classe base. A Herança permite a criação de classes complexas sem que seja necessário repetir o código, pois a classe herda do nível base (Ricarte, I. L. M, 2001).

A Figura 4 mostra as subclasses herdando da classe base ou superclasse.

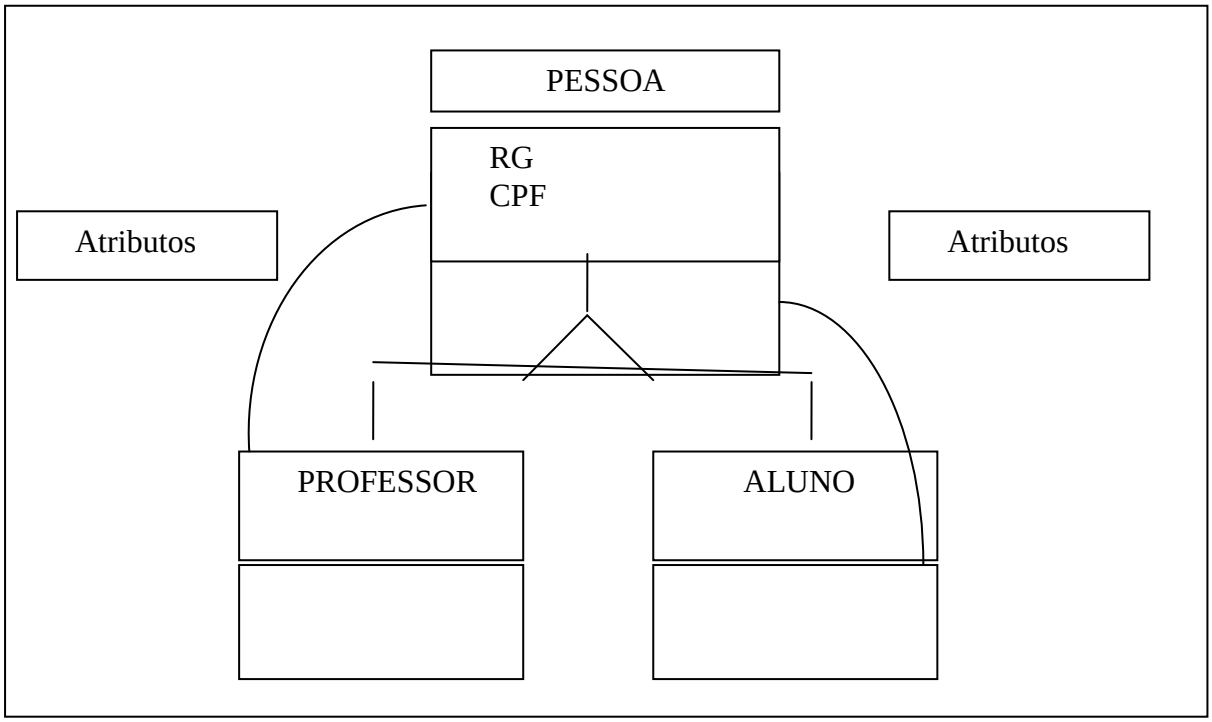



Figura 4: Herança

### 2.3.1.3 - POLIMORFISMO

Polimorfismo refere-se à capacidade de duas ou mais classes ou objetos responderem a mesma mensagem, ou seja, as subclasses tem o mesmo método que a superclasse. O Polimorfismo é a possibilidade de dois objetos de classes diferentes possuírem duas operações com o mesmo nome e implementações diferentes.

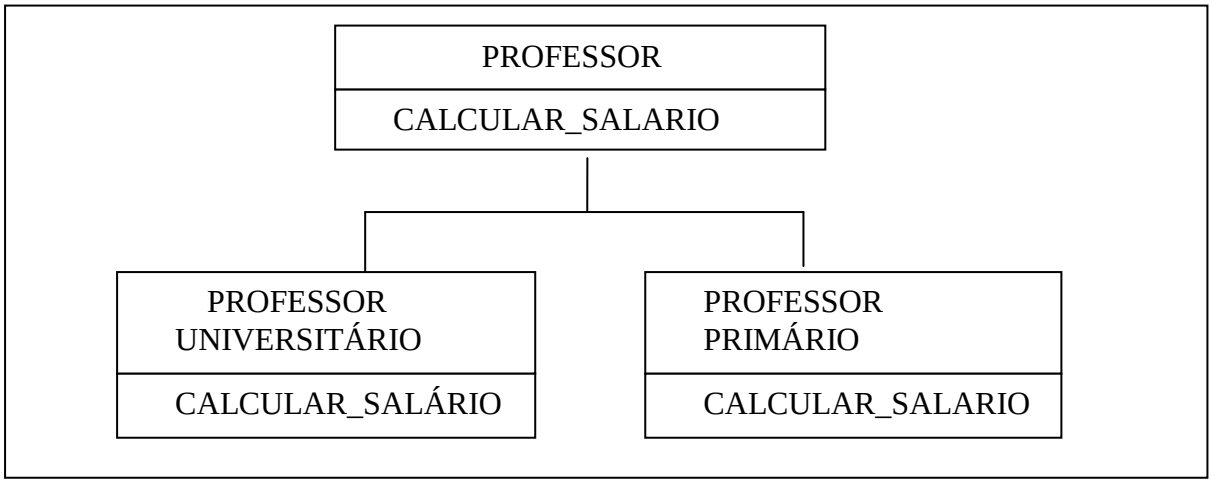



Figura 5: Polimorfismo

# 3 – REFLEXÃO EM TECNOLOGIA JAVA

Reflection é a habilidade que um programa pode possuir de examinar objetos em tempo de execução e agir conforme as informações encontradas.

## 3.1 – API DE REFLEXÃO JAVA

A API Java Reflection representa classes, interfaces e objetos presentes na JVM. Segundo [Trail: The Reflection API], reflection é uma técnica poderosa e pode permitir que os aplicativos possam executar operações que de outra forma seria impossível. Reflexão na tecnologia Java é um conceito muito poderoso e pode ser muito útil. Por exemplo, mapear objetos de uma classe gerando instruções SQL (updade, delete, insert) referente às tabelas no banco de dados em tempo de execução [Tutorial Java Reflection].

Segundo Sousa, F.C (2002), Reflection na tecnologia Java é constituída dos pacotes java.lang.reflect e Java.lang.string, e permite mudar o comportamento do programa, podendo fazer inspeções sobre definições da classe.

### 3.1.2 – CLASSES BÁSICAS

A seguir será apresentado definições de algumas classes existentes na tecnologia Java.

- **JAVA.LANG.REFLECT:** Pacote que contém as classes básicas, introduzido na release JDK 1.1.

- **JAVA.LANG.CLASS:** Representa Classes e Interfaces em um aplicativo Java em execução. *Class* não tem nenhum construtor público, os objetos *Class* são construídos automaticamente pelo Java Virtual Machine como as classes são carregadas.

- **JAVA.LANG.PACKAGE:** Provê Informações sobre um pacote.

- **JAVA.LANG.CLASSLOADER:** Provê informações sobre classes abstratas e operações para carregamento de classes

- **JAVA.LANG.REFLECT.MEMBER:** Interface que identifica informações sobre membros simples, atributos e métodos ou um construtor, e fornece informações e acesso a um único método em uma classe ou interface. O método refletido pode ser um método de classe ou um método de instancia (incluindo um método abstrato).

- **JAVA.LANG.REFLECT.MODIFIER:** Prover métodos estáticos e constantes para decodificar modificadores de acesso de classe e membros .

- **JAVA.LANG.REFLECT.ARRAY:** Prover métodos estáticos para criação e acesso a arrays Java.

- **JAVA.LANG.REFLECT.CONSTRUCTOR:** Prover informações para acessar os construtores de uma classe.

- **JAVA.LANG.REFLECT.FIELD:** Prover informações para acessar os campos de uma classe – campos de instancia e de classe ou interface.

- **JAVA.LANG.REFLECT.METHOD:** Prover informações para acessar os métodos de uma classe ou interface.

- **JAVA.LANG.REFLECT.PROXY:** Adicionado na release JDK 1.3, prover métodos estáticos para criação de proxies de classes dinamicamente.

- **JAVA.LANG.REFLECT.INVOCATIONHANDLER:** **I**nterface implementada pela instância de um Proxy.

# 4 – DESENVOLVIMENTO DE APLICAÇÕES COMERCIAIS USANDO JAVA REFLECTION

O Paradigma de Reflexão Computacional pode ser implementado em qualquer tipo de projeto orientado a objeto. Com foco em aplicar os conceitos de reflexão foi desenvolvido um sistema financeiro empresarial para empresas de pequeno porte, somente para ter controle de movimentação de dinheiro no dia a dia e um controle de cheque. O sistema foi implementado no intuito de reutilização de código, tornando o sistema flexível e um código legível e de fácil manutenção.

## 4.1 – O SISTEMA FINANCEIRO EMPRESARIAL

O sistema financeiro empresarial proposto possui como característica principal a manutenção do controle de débito e crédito, tanto de fornecedor como de cliente ou despesas de funcionários, também deve controlar movimentações de cheques, cheques a vencer, data de liberação. O sistema tem como público alvo empresas de pequeno porte.

O sistema possui um cadastro de crédito, que controla a quantidade que entrou de dinheiro no dia, é necessário selecionar o cliente, centro de custo (loja de roupa, açougue, padaria, etc -) e informar o valor, data e condição de pagamento. O sistema também funciona da mesma forma para o cadastro de débito.

O cadastro de cheque foi desenvolvido para facilitar operações numa devolução de cheques, depósito e cheques sustados, para eliminar marcações em papéis, sendo mais fácil de consular um cheque, verificar qual cheque pode ser depositado, se o cheque foi devolvido, já possui um cadastro do cliente para poder entrar em contato. Na figura 8, tem uma representação do sistema financeiro com ajuda de um mapa mental.

Para desenvolvimento do projeto será utilizada a tecnologia Java como linguagem de programação e o ambiente de desenvolvimento Eclipse, banco de dados será PostgreSQL 8.4 usando o ambiente de desenvolvimento pgAdmin III.

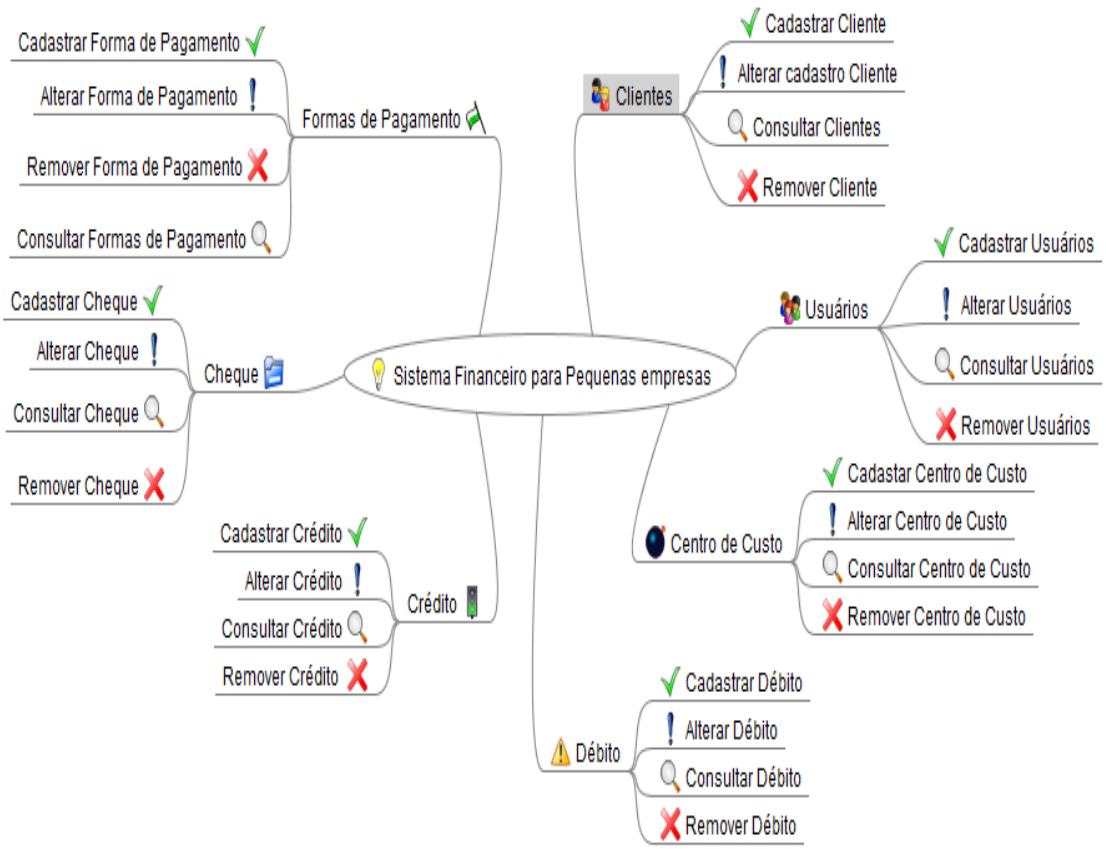



**Figura 8: Mapa Mental do Sistema Financeiro**

## 4.2 – MODELAGEM DA APLICAÇÃO

A seguir será exemplificada através do diagrama de Caso de Uso a total funcionalidade do sistema, o sistema só poderá ser acessado após o usuário confirmar o acesso por meio da interface de identificação (login) do Usuário ou Administrador. Depois de ter acessado o usuário, poderá alterar, cadastrar, excluir e consultar qualquer informação manipulada pelo sistema.

No diagrama de Caso de Uso é apresentado os itens manter (Cliente, Cheque, etc...). Neste contexto, o manter significa (alterar, inserir, remover, consular), métodos disponíveis em um sistema para manipular dados que estão no banco de dados ou dados que devem ser inseridos no banco de dados.

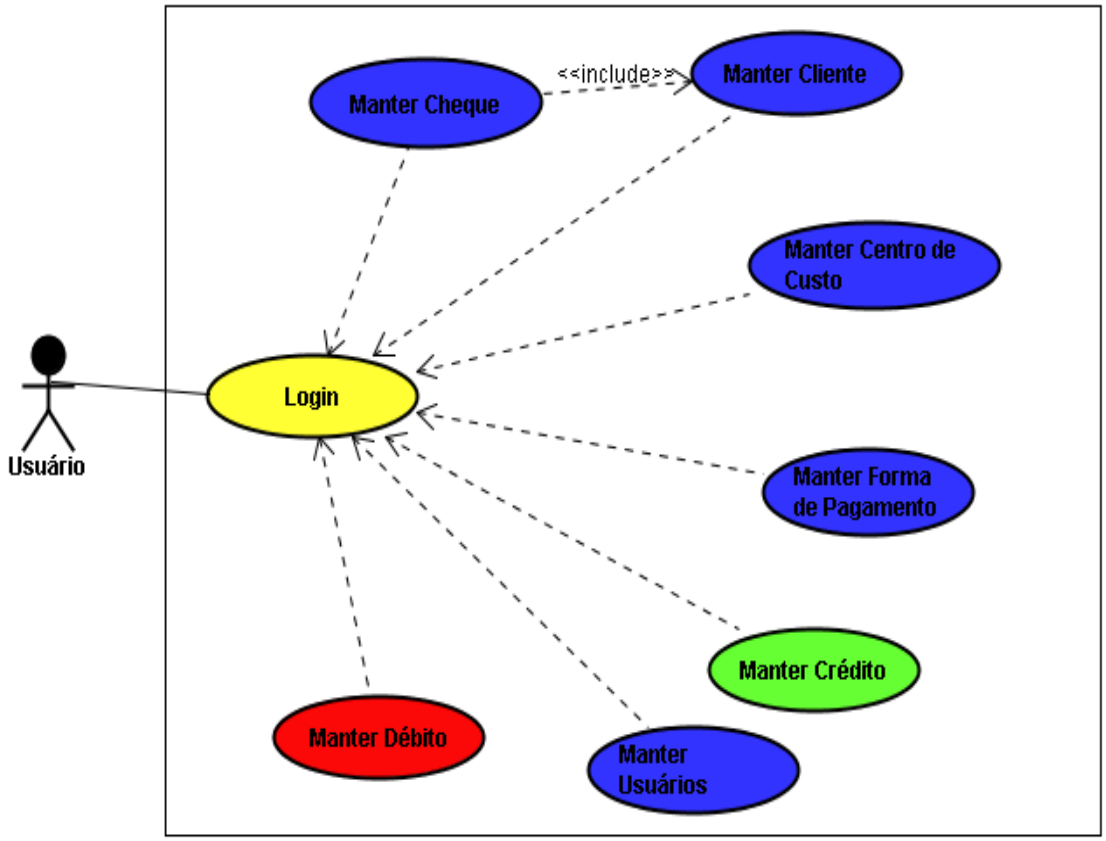



**Figura 6: Diagrama de Caso de Uso**

Na figura 7 a seguir será mostrado o diagrama de classes, e neste diagrama os mesmos nomes dos atributos, é o mesmo nome das colunas que vai ser gravado no banco de dados, no entanto qualquer nome dos atributos do diagrama de classe for diferente do nome da coluna da tabela que está no banco de dados, possivelmente ocorrerá um erro.

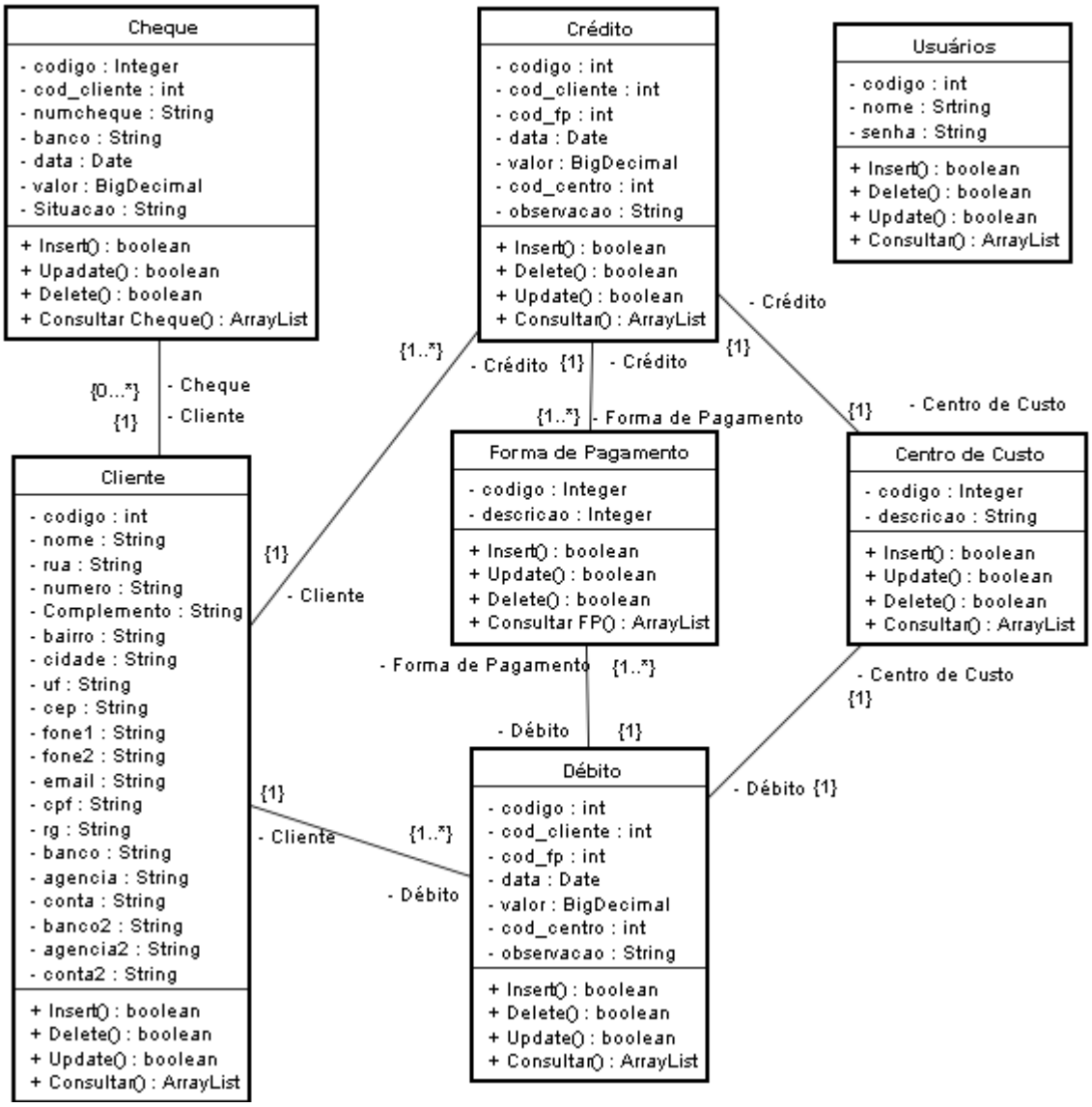



**FIGURA 7: DIAGRAMA DE CLASSES**

## 4.2 – IMPLEMENTAÇÃO DO REFLECTION

Neste sistema financeiro, a reflexão foi aplicada na construção de uma String SQL em tempo de execução para manipulação aos dados, utilizando um método genérico para a construção da SQL, a qual vai ser explicada mais a frente.

Sem usar reflexão, é preciso para cada classe modelo (classe que contém os atributos de uma tabela) tenha uma classe de persistência (classe que contém métodos para manipular os objetos e gravar em um banco de dados), se houver 30 classes modelos em um sistema, então será necessário 30 classes de persistência,

pois o método da persistência receberá a um objeto da classe modelo, e para poder recuperar os valores dos objetos que estão na classe modelo, será utilizado os métodos da classe modelo (get e set), no código 1 abaixo será apresentado um exemplo da classe modelo com seus atributos e métodos.

Código 1 – Classe Modelo com atributos e métodos

```
public class AutomovelBEANS {

    private Integer codigo_automovel;
    private String descricao_automovel;
    private String placa_automovel;

    public AutomovelBEANS(){
        this.codigo_automovel = 0;
        this.descricao_automovel = null;
        this.placa_automovel = null;
    }
    public AutomovelBEANS(Integer codigo_automovel,
            String descricao_automovel,
            String placa_automovel){
        this.codigo_automovel = codigo_automovel;
        this.descricao_automovel = descricao_automovel;
        this.placa_automovel = placa_automovel;
    }
    public Integer getCodigo_automovel() {
        return codigo_automovel;
    }

    public void setCodigo_automovel(Integer codigo_automovel) {
        this.codigo_automovel = codigo_automovel;
    }

    public String getDescricao_automovel() {
        return descricao_automovel;
    }

    public void setDescricao_automovel(String descricao_automovel) {
        this.descricao_automovel = descricao_automovel;
    }
```

No próximo código será apresentada a classe de persistência aos dados, sem reflexão, no código 2 tem um método *InserirAutomóvel*, que tem uma instância psmt do objeto *PreparedStatemant*, esse objeto tem um comando SQL, onde é passado a o nome da tabela e suas colunas, e seus valores são identificados por coringas (????), posteriormente estes coringas serão substituído pelos valores contidos na

classe modelo (Código 1), sendo recuperado com o método *getDescrição* ou *getPlaca*. Programando sem reflexão será necessário todo esse trabalho na classe de persistência, ficar repetindo e reescrevendo este método e os demais (Update, Delete) para cada classe modelo.

Código 2 – Classe Persistência sem Reflexão

```
public boolean InserirAutomovel(AutomovelBEANS beans){
    Connection con = null;
    PreparedStatement pstmt = null;
    con = Conexao.getInstance().getConnection();
    try {
        con.setAutoCommit(false);
        pstmt =  con.prepareStatement
       ("insert into automovel(descricao_automovel,placa_automovel)"
               + "values(?,?);");
        pstmt.setString(1,beans.getDescricao_automovel());
        pstmt.setString(2,beans.getPlaca_automovel());
        pstmt.execute();
        pstmt.close();
        con.commit();
        return true;
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        return false;
    }
}
```

Usando-se a reflexão, tem o trabalho de fazer a classe da persistência apenas uma vez, com isso pode ter várias classes modelo, com diferentes atributos e utilizando apenas uma classe para manipular os dados e salvar no banco de dados.

Na classe modelo será necessário acrescentar alguns métodos que não tinham na classe sem reflexão, no código 3 à seguir contém os métodos utilizados pela reflexão na classe modelo, nesses métodos é passado o nome da tabela que vai ser utilizada e o comando *this* referência o objeto da classe, a função é do tipo

boolean por que depois de executar os métodos será retornado true ou false, para a interface (Tela) informando ao usuário se foi efetuado ou não.

Código 3 – Métodos da classe modelo

```
public boolean Insere(boolean b){
    b = false;

    try {
        GenericoDAO per = new GenericoDAO();
        per.Insere("titular", this);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return false;
}

public boolean Update(boolean b){
    b = false;

    try {
        GenericoDAO per = new GenericoDAO();
        per.Update("titular", this);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
    }

    return false;
}

public boolean Delete(boolean b){
    b = false;
    try {
        GenericoDAO per = new GenericoDAO();
        per.Delete("titular", this);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
    }

    return false;
}
```

A seguir o código 4, 5 e 6 apresentará os métodos genéricos de Insert (Salva o registro no banco de dados), Update (Altera o registro no banco de dados) e Delete (Remove o registro no banco de dados) utilizando a reflexão.

Todos os métodos tem uma String SQL, essa String é concatenada de acordo com o objeto que é passado, esse objeto vem da classe modelo, primeiro precisa saber o nome da classe (Class classe = objeto.getClass), depois de ter o nome da classe, tem acesso a todos os atributos da classe, então é adicionado à String os atributos, um a um, e respectivamente os seus valores, e no final passa a SQL para

o método Manipular, que é responsável para fazer a conexão com o banco de dados.

Código 4 – Método Insert Genérico

```
public boolean Insert(String tabela, Object objeto) throws Exception{
    String sql = "INSERT INTO " + tabela.toUpperCase() + "(";

    Class classe = objeto.getClass();
    for (Field f : classe.getDeclaredFields()){
        f.setAccessible(true);
        sql += f.getName() + ",";
        if(f.getName().equals("codigo")){
            sql = sql.substring(0, sql.length() - 7);
        }
    }
    sql = sql.substring(0, sql.length() - 1);//retira a ultima virgula...
    sql += ") VALUES (";

    int cont = 1;
    for(Field f : classe.getDeclaredFields()){
        f.setAccessible(true);
        if(cont == 1){
        f.getType().getSimpleName();
        cont = 2;
        }else{
            if(f.getType().getSimpleName().equals("String")
                    || f.getType().getSimpleName().equals("Date")){
                sql += "'" + f.get(objeto) + "',";
            } else {
                sql += f.get(objeto) + ",";
            }
        }
    }

    sql = sql.substring(0, sql.length() - 1); //retira a ultima virgula
    sql += ")";

    //executa a query
    System.out.println(sql);
    if(!Manipular(sql)){
        throw new Exception();
    }
    return true;
}
```

O método Insert Genérico acima recebe dois parâmetros (String e Object), que vem da classe que a chamou, logo na primeira linha do método é criada uma variável do tipo *String* que recebe uma sql a qual vai ser construída em tempo de execução do sistema, em seguida existe um *for* que percorre todos os atributos declarados na classe que solicitou o método Insert, dentro do for tem um *if* que verifica se tem o atributo código, se existir o atributo código então ele é removido da

sql porque no banco de dados a coluna código foi definida como auto incremento, não sendo necessário informar o valor do atributo código.

O segundo *for* do método Insert, percorre todos os atributos da classe que solicitou o método insert, mas logo na segunda linha do *for* tem um *if* que verifica se o contador é igual a 1, se for igual a 1 pula para o próximo atributo. Ele pula para o próximo atributo porque sempre o primeiro atributo declarado na classe é o código, como a coluna código é de auto incremento, não pega o valor do atributo código, então é passado para o segundo *if* que verifica se o atributo é do tipo *String* ou *Date,* e se for de algum desses tipos então o valor é colocado entre “ ` ` ” e depois é adicionado “,” no fim do valor do atributo, se o atributo for de outro tipo, é adicionado somente “,” no fim e continua percorrendo outros atributos.

Código 5 – Método Update Genérico

```
public boolean Update(String tabela, Object objeto) throws Exception{
    String sql = "UPDATE " + tabela.toUpperCase() + " SET ";
    int cont = 1;
    //percorre os atributos vendo o nome dos campos
    Class classe = objeto.getClass();
    String val = "";
    int cod = 0;
    for (Field f : classe.getDeclaredFields()){
        f.setAccessible(true);
        sql += f.getName() + "=";
        if(f.getName().equals("codigo")){
            sql = sql.substring(0, sql.length() - 7);
        }
        if(cont == 1){
            f.getType().getSimpleName();
            val = f.get(objeto).toString();
            cod = Integer.parseInt(val.toString());
            cont = 2;
            }else{
                if(f.getType().getSimpleName().equals("String")
                        || f.getType().getSimpleName().equals("Date")){
                    sql += "'" + f.get(objeto) + "',";
                } else {
                    sql += f.get(objeto) + ",";
                    }
                }
    }

    sql = sql.substring(0, sql.length() - 1); //retira a ultima virgula
    sql += "WHERE codigo = '"+cod+"'";

    //executa a query
    System.out.println(sql);
    if(!Manipular(sql)){
        throw new Exception();
    }
    return true;
}
```

O método Update Genérico não é muito diferente do Insert, a diferença é que o método Update diferencia na *String* sql , essa *String* recebe primeiro *update* e o nome da tabela e depois *set*, em seguida repete o mesmo procedimento do método Insert e ao final adiciona uma clausula *where* passando o atributo código e seu respectivo valor, esses dois métodos é bem parecido um ao outro.

Código 6 – Método Delete Genérico

```
public boolean Delete(String tabela, Object objeto) throws Exception{
    String sql = "DELETE FROM " + tabela.toUpperCase() + " WHERE ";
    Class classe = objeto.getClass();
    for (Field f : classe.getDeclaredFields()){
        f.setAccessible(true);
        if(f.getType().getSimpleName().equalsIgnoreCase("Integer")
                && f.get(objeto) != "0" && f.getName().equals("codigo")){
            sql += f.getName() + " = ";
        }

        if(f.getType().getSimpleName().equalsIgnoreCase("Integer")
                && f.get(objeto) != "0" && f.getName().equals("codigo")){
            sql += f.get(objeto) + ";";
            }
    }

    //executa a query
    System.out.println(sql);
    if(!Manipular(sql)){
        throw new Exception();
    }


    return true;
}
```

Neste código 6 acima representa o método Delete Genérico, que recebe dois parâmetros igual aos métodos anteriores, código 4 e 5. Esse método e bem mais simples, pois só passa um atributo, e geralmente por padrão e o código, o método Delete só tem um *for* para construir a *String* sql, que procura um *objeto* do tipo *Integer* e nome igual a *código*, e com esses dados a *String* sql já está pronta.

Código 7 – Método Manipular SQL

```
public static boolean Manipular(String sql) throws SQLException{
    Connection con = null;
    PreparedStatement pstmt = null;
    con = Conexao.getInstance().getConnection();
    try {
        con.setAutoCommit(false);
        pstmt =  con.prepareStatement(sql);
        //int result = pstmt.executeUpdate(sql);
        pstmt.execute();
        pstmt.close();
        con.commit();
        System.out.println("Instrucao SQL realizada com sucesso");
            return true;

    } catch (SQLException e) {
        System.out.println(e.getMessage());
        return false;
    }
}
```

Até agora os métodos Insert, Update e Delete só construíram a sql em tempo de execução, mas não foi feita nenhuma comunicação com um banco de dados, e para não ficar repetindo linhas de códigos várias vezes nesses métodos, foi criado um método *Manipular* que está no código 7 acima e recebe como parâmetro uma variável do tipo String, esse método cria uma conexão com o banco de dados e executa a sql, com isso reduziu 14 linhas de código de cada método (Insert, Update e Delete).

## 5 – CONCLUSÃO

Durante este trabalho foram apresentados conceitos que possibilitam implementar sistemas reflexivos. No decorrer do trabalho foi possível destacar a vantagem de utilizar reflexão computacional. No Capitulo 2 foram mostrados o conceito de reflexão computacional, seus modelos e conceitos de orientação a objetos, que é a base para o estudo de reflexão, também apresentou-se as propriedades da classe.

O capitulo 3 descreve a API Reflection, mostrando suas classes básicas. No capitulo 4 foi mostrado o estudo de caso, foi implementado um sistema que utiliza-se dos conceitos de reflexão computacional.

No decorrer deste trabalho foi possível avaliar a reflexão, pois é uma forma excelente de programar, como foi explicado no capítulo anterior, há uma redução na programação, por que não é necessário ficar criando classe de persistência para cada classe modelo, repetindo métodos várias vezes, sem ter que preocupar com os nomes das colunas da tabela que tem no banco de dados, pois definindo isso na classe modelo não será necessário alterar em outra classe, pois o método genérico está pronto para receber qualquer classe.

### 5.1 – TRABALHOS FUTUROS

Com base na pesquisa desenvolvida, várias vertentes para futuros trabalhos podem ser identificadas, Como possíveis trabalhos futuros, pode-se apontar:

- Pesquisa para aplicação de reflexão em regra de negócios, como interesse sistêmico.

## 6 – REFERÊNCIAS BIBLIOGRÁFICAS

**Arquitetura Reflexiva**. Conceitos de arquitetura reflexiva. Disponível em: <"http://www.inf.ufrgs.br/gppd/disc/cmp167/trabalhos/sem991/T1/alex/2/arquitetura/arquitetura-reflexiva.htm">. Acessado em 11 de Jun. 2012

Barth, Fabrício Jailson. **Utilização da Reflexão Computacional para implementação de aspectos não funcionais em um gerenciador de arquivos distribuídos** Trabalho de Conclusão de Curso, Universidade Regional de Blumenau. 2000

Conceitos Básicos. Disponível em: <"http://www2.dbd.puc rio.br/pergamum/tesesabertas/0115636_02_cap_02.pdf">. Acessado em 07 de Junho. 2012

Ferreira, Aurélio Buarque de H. Minidicionário da Língua portuguesa, Editora Nova Fronteira, 4 Ed, 2000

Jenkov, Jakob. **Java Reflection Tutorial.** Disponível em: <http://tutorials.jenkov.com/java-reflection/index.html>. Acesso em: 12 jun. 2012.

Maes, P. Concepts and experiments in computational reflection. ACM Sigplan Notices, v. 22, n. 12, p. 147-155, Dec. 1987

Oracle. **Trail: The Reflection API**. Uses of Reflection. Disponível em: <"docs.oracle.com/javase/tutorial/reflect/index.html">. Acessado em 12 de Jun. 2012

Ricarte, Ivan Luiz Marques. **Programação Orientada a Objetos: Uma Abordagem com Java**, Universidade Estadual de Campinas (UNICAMP). 2001

Sallem, Márcio Augusto Sekeff. **Adapta: um arcabouço para o desenvolvimento de aplicações distribuídas adaptativas.** Trabalho de Pós-Graduação, Universidade Federal do Maranhão. 2007

Silva, Romulo Cesar. **Rstabilis: Uma Máquina Reflexiva de Busca**, Dissertação de mestrado, Universidade Estadual de Campinas (UNICAMP). 1997

Smith, Brian C. Prologue to **. Reflection and Semanticsin a Procedural Language**. In Morgan Kaufmann, editor, The Knowledge Representation Enterprise, pp. 31-39. R.J. Brachman and H.S. Levisque, 1985.

Sousa, Fabio Cordova. **Utilização da Reflexão Computacional para implementação de um monitor de software orientado a objetos em Java**. Trabalho de Conclusão de Curso, Universidade Regional de Blumenau. 2002

Tutoriais Admin. **Tutorial Java: O que é Java**. JavaFree. Disponível em: <“http://javafree.uol.com.br/artigo/871498/Tutorial-Java-O-que-e-Java.html”>. Acessado em 17 de abr. 2012