Relatório Parcial 6

Sistema de agendamento de passagens aéreas

Rafael Valença
Engenharias
rafa7lima@gmail.com

Dennis Merli
Engenharia de Software
neneds@gmail.com

Iago Leite
Engenharia de Software
iago8mendes@hotmail.com

Disciplina de Orientação a Objetos
Professora Dra. Milene Serrano

Gama, 21 de Outubro de 2013

UnB - Campus Universitário Gama

Área Especial de Indústria Projeção A - UnB/Brasília, Setor Leste. CEP: 72.444-240

GAMA – DF

Relatório Parcial 6
Professora Dra. Milene Serrano Outubro, 2013

Relatório Parcial

Rafael Valença[1], Dennis Merli[2], Iago Leite[3]

[1]Engenharias, Universidade de Brasília - Faculdade do Gama, Gama/DF
[2]Engenharia de Software, Universidade de Brasília - Faculdade do Gama, Gama/DF
[3]Engenharia de Software, Universidade de Brasília - Faculdade do Gama, Gama/DF

rafa7lima@gmail.com, nenends@gmail.com, iago8mendes@hotmail.com

Resumo. Esse relatório se refere à Etapa 6 do trabalho da disciplina de Orientação a Objetos. Nele, implementamos dois métodos estáticos que fazem uso de um atributo também estático. Além disso, a classe abstrata PessoaFisica é implementada.

Palavras-chave: Passagem aérea, Consulta, Orientação a Objetos, Java.

Sumário

1 Introdução 1
2 Breve Descrição 1
3 Modelagem 1
4 Implementação 4
5 Controle de Versão 7
6 Referências 8

1 Introdução

Nesse relatório técnico procuramos apresentar os resultados obtidos até o momento - Etapa 6 do trabalho solicitado na disciplina de Orientação a Objetos, UnB/FGA.

O objetivo do trabalho é implementar um sistema de compra online de passagens aéreas, similar ao que ocorre nos sites de companhias aéreas (ver, por exemplo, [1] e [2]).

O relatório está organizado em seções. Na Seção 2, é apresentada uma breve descrição da proposta, Na Seção 3, a modelagem em UML, na seção 4, a implementação das classes, na Seção 5, o controle de Versão.

2 Breve Descrição

O sistema proposto é um sistema de compra de passagens aéreas. Até o momento, o sistema conta com classes como: Cliente, Funcionário, Passageiro, Voo, Aeronave, Atendente, Operador, Adminitrador, Aeroporto, PortaoDeEmbarque, dentre outras.

O objetivo principal desse desenvolvimento foi exercitar na prática como modelar e implementar um sistema baseado nos conceitos, princípios e fundamentos da Orientação a Objetos, tais como abstração, hierarquia, modularização, etc.

Este sistema possui funções tanto para clientes de uma empresa de aviação fictícia quanto para os funcionários desta empresa. O cliente tem um cadastro junto a empresa, mas pode se tornar passageiro ao comprar uma passagem aérea. Os tipos de funcionário são:

- Atendente: realiza check-in, despacha bagagens, etc.
- Operador: realiza o cadastro dos voos.
- Administrador: realiza ações mais delicadas, como remover voos que foram cadastrados, remover passageiros, etc. Administrador é um operador (relação de hierarquia).

A seguir, apresentamos detalhes da implementaçãoo referente à Etapa 6 do Desenvolvimento do Sistema.

3 Modelagem

A modelagem foi feita usando linguagem UML (*Unified Modeling Language*) como base, na versão 2.0. Todos os diagramas foram desenvolvidos através da ferramenta *Astah Professional*, disponível para Windows, MacOS e Linux, com licença gratuita para estudantes. Os downloads dos produtos da família Astah podem ser encontados em http://astah.net/download . As versões utilizadas foram a Astah Professional 6.7.0/43495 para MacOS e para Windows.

A Figura 01 apresenta uma visualização geral das classes obtidas até agora, bem como as relações entre essas classes.

- Cliente: é um consumidor potencial, mas não necessariamente está vinculado a algum voo.
- Passageiro: é um cliente que está agregado a um voo. Um voo é um conjunto de n passageiros (onde n é, no máximo, a capacidade permitida pela aeronave). Cada cliente pode estar em vários voos.
- Funcionário: é um empregado da empresa. Pode ser atendente, operador, ou administrador dependendo da sua função.
- Voo: possui horário, origem e destino, dentre outros atributos. Está associado a aeronave. Cada voo só pode ser realizado através de 1 aeronave, mas cada aeronave pode realizar diversos voos.
- Aeroporto: marca os possíveis origens e destinos dos voos. Aeroporto é composto por Portões de embarque.
- Data e horários são classes úteis que ajudam a administrar o tempo dentro do sistema. Data é uma agregação de horários.
- Ler permite fazer leituras (strings e integers) através do teclado.

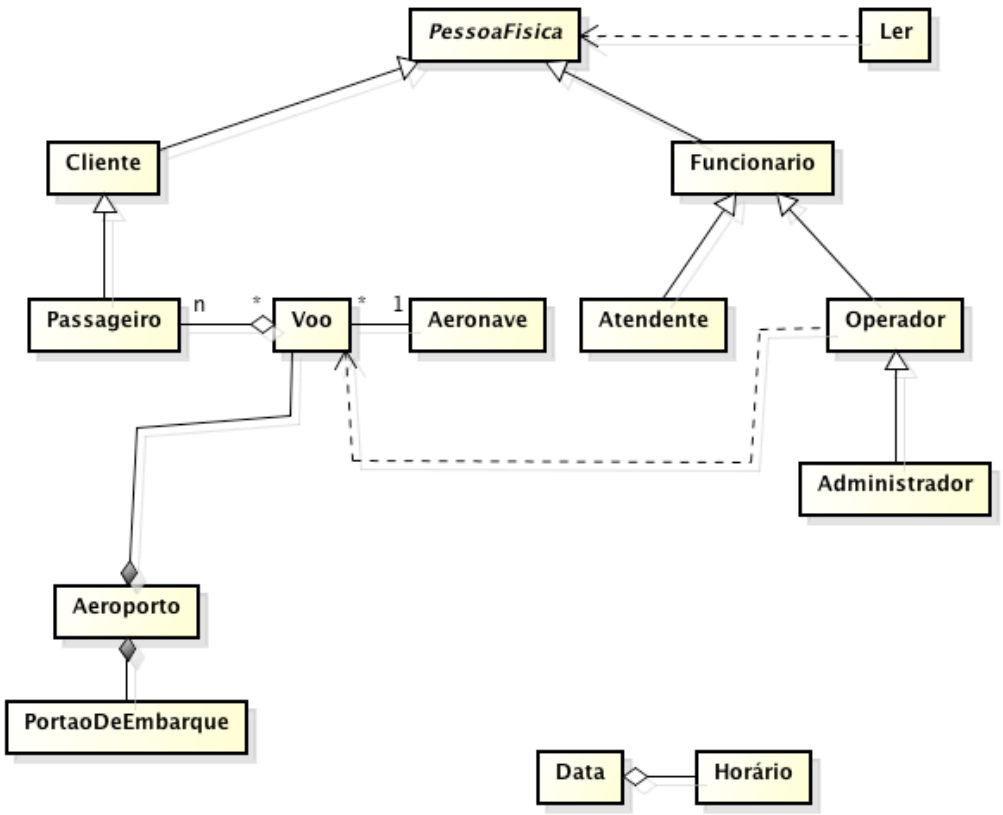

Figura 01: Modelagem - Diagrama de Classes (parcial).

A *main* não aparece no diagrama porque ela depende de um número grande de classes.

A classe abstrata implementada é a PessoaFisica. Essa é a razão pela qual o nome da classe aparece em itálico no diagrama de classes da Figura 01. O método cadastrarCliente (anteriormente, um método da classe Cliente) virou o método cadastrar da classe PessoaFisica. Isso faz com que tanto clientes quanto funcionários tenham método de cadastro no sistema. Haverá, contudo, polimorfismo neste método, que será sobrescrito nas classes Cliente e Funcionário. O método cadastrar na classe PessoaFisica é abstrato, mas é concreto nas classes Cliente e Funcionario.

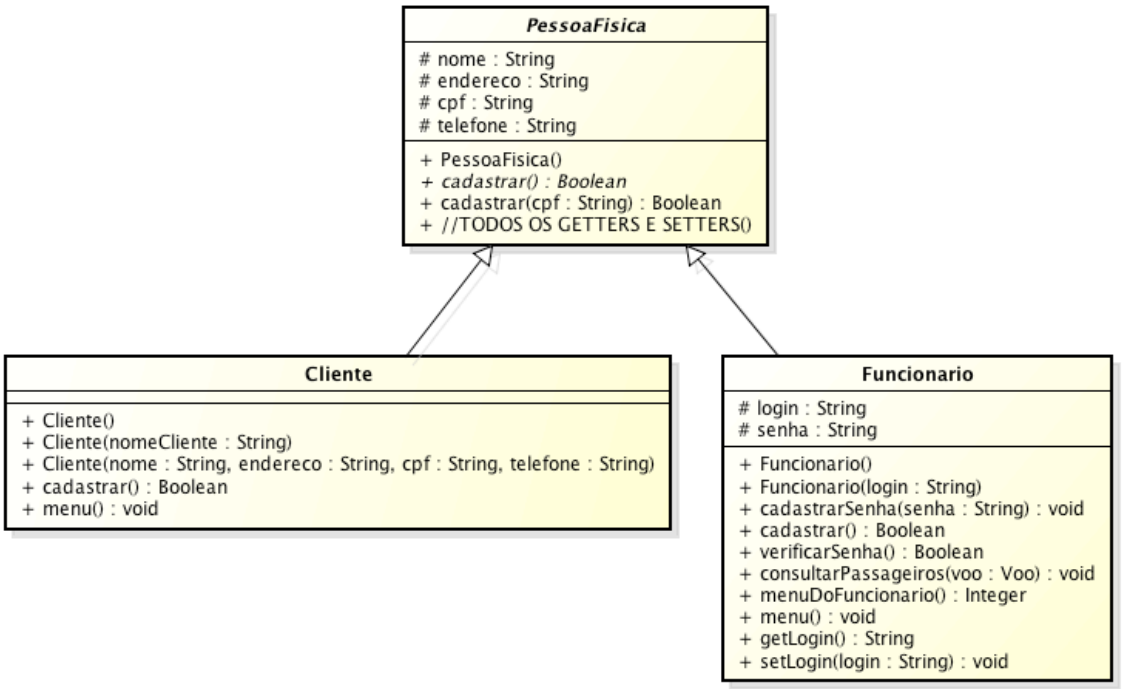

Figura 02: Modelagem – Classe Abstrata.

A classe Ler também foi implementada. Esta classe possui métodos que permitem receber informações do teclado.

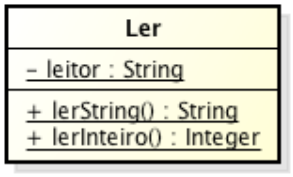

Figura 03: Modelagem – Classe Ler.

O atributo leitor e os métodos lerString e lerInteiro desta classe são estáticos (por isso aparecem sublinhados na modelagem). Eles bastante utilizados na *main* e nos menus do Sistema.

4 Implementação

Foram implementadas as classes da Figura 01, com seus respectivos relacionamentos.

As Figuras 04, 05 e 06 mostram o caso de polimorfismo. Nele, vemos que o método cadastrar(), que tem a mesma assinatura nas classes PessoaFisica, Cliente e Funcionario, funciona de forma ligeiramente diferente dependendo de qual classe pertence o objeto que chama este método. Na classe PessoaFisica (abstrata), o método é abstrato.

```
1  package sistemaVoo.entidades;
2
3  import sistemaVoo.Ler;
5
6  public abstract class PessoaFisica {
7
8      protected String nome = " ";
9      protected String endereco = " ";
10     protected String cpf = "";
11     protected String telefone = " ";
12
13     //CADASTRAR
14     public abstract boolean cadastrar();
15
```

Figura 04: Implementação – Classe abstrata PessoaFisica e método abstrato cadastrar().

```
25     //Cadastrar
26     public boolean cadastrar(){
27         System.out.println("Diga seu CPF (Apenas números):");
28         String cpf = Ler.lerString();
29         TesteCpf teste = new TesteCpf();
30         boolean validacaoCpf = teste.ValidarCpf(cpf);
31         if(validacaoCpf == true){
32             System.out.println("Insira os demais dados:");
33             System.out.println("Nome:");
34             this.nome= Ler.lerString();
35             System.out.println("Endereco:");
36             this.endereco=Ler.lerString();
37             System.out.println("Telefone:");
38             this.telefone=Ler.lerString();
39             this.cpf = cpf;
40             System.out.println("O cliente " + this.nome + " foi cadastrado com sucesso.\n");
41
42         }
43         else{
44             System.out.println("CPF inválido. Favor inserir um CPF válido.\n");
45         }
46         return validacaoCpf;
47     }
```

Figura 05: Implementação – Método cadastrar() chamado por uma instância de Cliente.

```
28      //Cadastrar
29      public boolean cadastrar(){
30          System.out.println("Diga seu CPF (Apenas números):");
31          String cpf = Ler.lerString();
32          TesteCpf teste = new TesteCpf();
33          boolean validacaoCpf = teste.ValidarCpf(cpf);
34          if(validacaoCpf == true){
35              System.out.println("Insira os demais dados:");
36              System.out.println("Login:");
37              this.login= Ler.lerString();
38              System.out.println("Senha:");
39              this.senha= Ler.lerString();
40              System.out.println("Nome:");
41              this.nome= Ler.lerString();
42              System.out.println("Endereco:");
43              this.endereco=Ler.lerString();
44              System.out.println("Telefone:");
45              this.telefone=Ler.lerString();
46              this.cpf = cpf;
47              System.out.println("O funcionário " + this.nome + " foi cadastrado com sucesso.\n");
48
49          }
50          else{
51              System.out.println("CPF inválido. Favor inserir um CPF válido.\n");
52          }
53          return validacaoCpf;
54      }
55
```

Figura 06: Implementação – Método cadastrar() chamado por uma instância de Funcionario.

A classe Ler é uma classe estática, que usa um atributo e dois métodos estáticos (Figura 07). Isso faz com que estes métodos possam ser usados sem que uma instância da classe Ler seja gerada. O atributo (String) leitor normalmente recebe um string lido do teclado. O método lerString() tem este String como saída. O método lerInteiro transforma, se possível, este String em um Integer.

O método cadastrar() foi testado através de um teste de unidade chamado TesteCadastroViaCpf. Nele, o cadastro de um cliente é realizado com um CPF válido e com um CPF inválido. O teste deve nos assegurar que o cadastro só seja realizado se for usado um CPF válido. O teste aparece na Figura 08.

Os códigos das demais classes do Sistema estão junto deste relatório em um arquivo .zip que foi submetido pelo moodle.

```java
package sistemaVoo;

import java.util.Scanner;

public class Ler {

    private static String leitor;

    public Ler() { }

    public static String lerString(){
        leitor = new Scanner(System.in).nextLine();
        return leitor;
    }

    public static int lerInteiro(){
        boolean erro = false;
        int i = 0;
        do{
            try{
                i = Integer.parseInt(lerString());
                erro = false;
            }catch(NumberFormatException e){
                System.out.println("Insira um número válido.");
                erro = true;
            }
        }while(erro);
        return i;
    }
}

```

Figura 07: Implementação – Classe estática Ler e seus métodos estáticos lerString() e lerInteiro().

```
11
12  public class TesteCadastroViaCpf extends TestCase {
13
14      @Before
15      protected void setUp() throws Exception {
16          super.setUp();
17          System.out.println("Iniciando...\n");
18      }
19
20      @After
21      protected void tearDown() throws Exception {
22          super.tearDown();
23          System.out.println("Fim.\n");
24      }
25
26      @Test
27      public void test() {
28          Funcionario funcionario = new Funcionario();
29
30          String cpfValido = "02545188182";
31          String cpfInvalido = "12345678901";
32
33          System.out.println("Teste com CPF válido:");
34          boolean resposta = funcionario.cadastrar(cpfValido);
35          assertTrue(resposta);
36
37          System.out.println("\nTeste com CPF inválido:");
38          resposta = funcionario.cadastrar(cpfInvalido);
39          assertFalse(resposta);
40
41      }
```

Figura 08: Implementação – Teste de unidade que verifica o método cadastro().

5 Controle de Versão

O repositório com o histórico de modificações no sistema implementado está disponível no endereço [4]. Este é o segundo repositório usado pelo grupo. O primeiro repositório, com as primeiras tentativas de modelagem, implementação e tentativas de usar o gitHub está acessível em [5].

Figura 09: Controle de Versão – Descrição do Repositório no GitHub.

6 Referências

[1] TAM < http://www.tam.com.br/ ; acessado em 20 de outubro de 2013 >

[2] Submarino Viagens < http://www.submarinoviagens.com.br/ ; acessado em 20 de outubro de 2013 >

[3] Portal de desenvolvedores do Brasil: < http://www.guj.com.br/java/287165-uml---sistema-de-compras-de-passagens-aereas ; acessado em 20 de outubro de 2013 >

[4] Repositório do GitHub usado atualmente: < https://github.com/rafa7lima/SistemaVoo2.0 ; acessado em 20 de outubro de 2013 >

[5] Repositório do GitHub usado atualmente: < https://github.com/rafa7lima/Sistema_OO ; acessado em 20 de outubro de 2013 >