INSTITUTO DE COMPUTAÇÃO– UNICAMP
1o. SEMESTRE DE 2009

Profa. Cecília M. F. Rubira

Sala 13, cmrubira@ic.unicamp.br

Versão: 1.1 Data: 22 de abril de 2009.

Estudo de Caso: Sistema de Folha de Pagamento

Este documento descreve um sistema de folha de pagamento que gera demonstrativos de pagamento para funcionários cadastrados de uma empresa comercial, onde os conceitos de classe e de relacionamentos de generalização/especialização, agregação/decomposição e associação entre classes são empregados.

1. Análise do Problema

O sistema de controle de folha de pagamento se destina a uma empresa comercial que tem dois tipos de funcionários cadastrados: gerentes e não-gerentes. Estes últimos incluem auxiliares administrativos e vendedores.

O sistema deve permitir o registro diário de eventos como eventuais atrasos de funcionários e reajustes salariais. Além da geração de demonstrativos de pagamento dos funcionários com lançamentos referentes ao seu salário base e aos eventos registrados. Além disso, o sistema deve permitir que o operador do sistema solicite estas operações, de adição de eventos e geração de demonstrativos para funcionários cadastrados, através de um terminal.

Um funcionário cadastrado possui as seguintes informações : nome do funcionário, data de admissão, o número de horas de atrasos, data de fechamento, salário base, uma lista de eventos associados a ele e uma lista de demonstrativos referentes a seus vários pagamentos ao longo do tempo. A data de admissão de um funcionário corresponde a data em que o mesmo foi admitido e o seu número de horas de atrasos indica atrasos na sua hora de chegada, se houver, pois existe um controle de ponto. Já data de fechamento de um funcionário é a data de fechamento referente ao seu último demonstrativo, que será igual à data de admissão, se recém admitido.

O salário base de um funcionário é definido conforme tipo do funcionário. O salário base de um funcionário não-gerente é de R$300,00 e o de um funcionário gerente é de R$400,00. O salário mensal de um funcionário é o seu salário base com descontos de valores em função de atrasos na sua hora de chegada. E ainda, o salário base pode ser atualizado através de reajustes salariais. As regras utilizadas para calcular o salário mensal de um funcionário são:

a. Reajustes salariais não podem ser inferiores ao valor do salário base corrente. Um reajuste implica na mudança do salário base para o novo valor especificado através do reajuste.
b. Para efeitos de cálculo, considera-se o valor da hora igual a 1/176 do salário base.
c. Atrasos são descontados com base no valor da hora.
d. Existe um limite máximo de 2 horas para um funcionário chegar atrasado à empresa. Caso um atraso do funcionário tenha quantidade superior á duas horas, o sistema não deve registrar o evento como um evento de atraso.

Os eventos de atraso de um funcionário e reajuste de salário, juntamente com o salário base de um funcionário, são utilizados para gerar o seu demonstrativo de pagamento. Um demonstrativo possui vários lançamentos, correspondente a cada linha do demonstrativo, onde são exibidos os valores a serem computados para o cálculo do seu salário mensal e a descrição do lançamento. O demonstrativo gerado corresponde a um período definido por uma data inicial e uma data final. A data inicial corresponde à data de fechamento do funcionário. A data final corresponde à data corrente.

2. Especificação de casos de uso

A modelagem de casos de usos do sistema é mostrada na Figura 1.

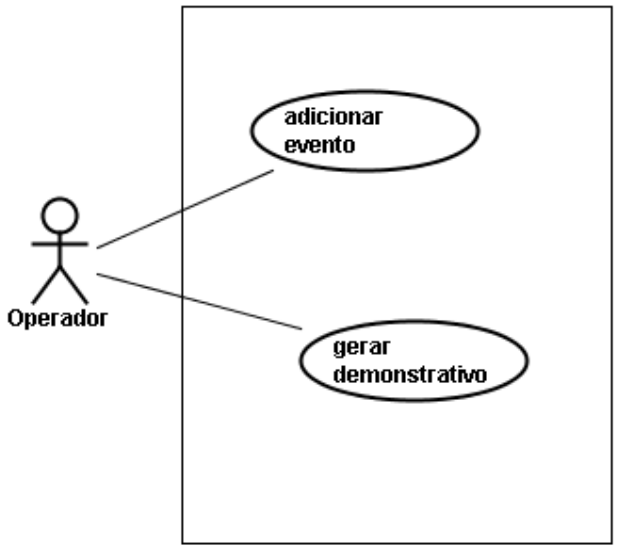


Figura 1 - Diagrama de Casos de Uso do Sistema de Folha de Pagamento

Nome do caso de uso: Adicionar evento

Breve descrição: Este caso de uso representa o processo de associar um evento de reajuste ou de atraso a um funcionário. O caso de uso se inicia quando o operador escolhe um usuário e solicita a operação de adicionar evento, especificando uma data e um valor para o mesmo.

Ator: operador

Pré-condições: existe ao menos um funcionário cadastrado no sistema. A data do evento a ser adicionado é posterior a data de fechamento do funcionário e anterior a corrente.

Fluxo básico de eventos:

1. O operador seleciona um funcionário.
2. O operador solicita adicionar evento.
3. O operador identifica o tipo do evento: reajuste ou atraso.
4. O operador especifica a data da ocorrência do evento e um valor associado.
5. Se ((evento = "Evento Atraso" e o "Valor Evento" <= 2) ou (evento = "Evento Reajuste")).
   5.1. O sistema cria um novo evento contendo a data, o valor e o tipo do evento especificado.
   5.2. O sistema adiciona o evento ao conjunto de eventos do funcionário selecionado.

Nome do caso de uso: gerar demonstrativo do funcionário.

Breve descrição: este caso de uso representa o processo de gerar um demonstrativo para funcionários conforme as regras de cálculo do salário mensal especificadas na Seção 1. O demonstrativo gerado corresponde a um período definido por uma data inicial e uma data final.

Ator: operador

Pré-condições: existe pelo menos um funcionário cadastrado no sistema.

Pós-condições: um demonstrativo deve ser gerado contendo os lançamentos referentes a eventos de atrasos e/ou reajustes associados ao funcionário e salário base do funcionário. A data de fechamento do funcionário deve estar atualizada para a data final do demonstrativo.

Fluxo básico de eventos:

1. O operador seleciona um funcionário.
2. O operador solicita gerar demonstrativo.
3. Para cada evento associado ao funcionário:
   3.1. O sistema verifica se a data do evento é posterior a data de fechamento do funcionário.
      3.1.1. Se (evento = "Evento Reajuste" e "Valor Evento" > salário base do funcionário).
         3.1.1.1. O sistema atualiza o salário base do funcionário:
         salário base := valor do evento reajuste
      3.1.2. Se evento = "Evento Atraso"
         3.1.2.1. O sistema atualiza o número de horas de atraso associadas ao funcionário:
         quantidade horas atraso := quantidade horas atraso + valor do evento atraso.
4. O sistema cria um novo demonstrativo para o funcionário selecionado.
5. O sistema inicializa o período do demonstrativo
   Data inicial = data de fechamento do funcionário
   Data final= data corrente.
6. O sistema inclui um novo lançamento ao demonstrativo criado:
   "Salário Base": valor do salário base.
7. Se "quantidade horas de atraso" > 0
   7.1. O sistema calcula o valor a ser debitado do demonstrativo:

valor = - (número de horas de atraso * 8 * salário base / 176)

7.2. O sistema inclui um novo lançamento ao demonstrativo criado:

"Horas Atrasos": valor

8. O sistema atualiza a data de fechamento do funcionário:

   data fechamento do funcionário = data final do demonstrativo.

9. O sistema adiciona o demonstrativo criado ao conjunto de demonstrativos do funcionário.
10. O sistema imprime o demonstrativo com seus lançamentos e o valor total a ser pago para o funcionário conforme valores dos lançamentos do demonstrativo.

3. Projeto da Hierarquia de Classes

A partir da descrição do problema e especificação dos casos de uso podemos identificar as classes representadas no diagrama de classes da Figura 2.

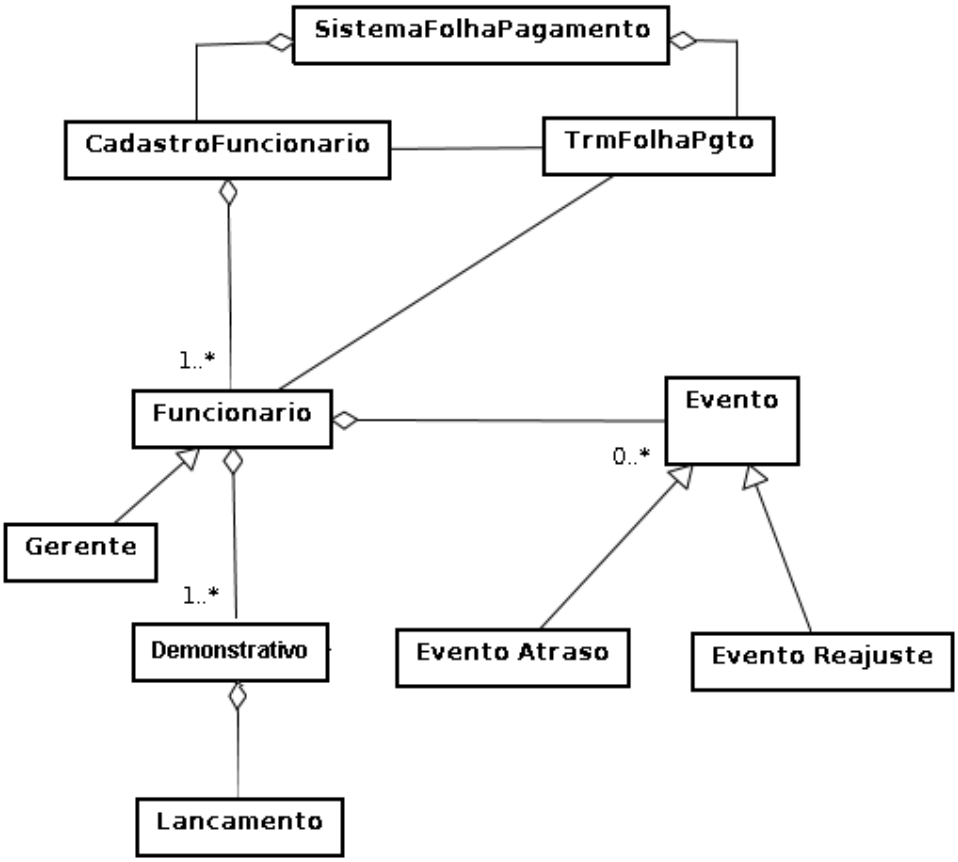


Figura 2 - Diagrama de Classes do Sistema de Folha de Pagamento

Um funcionário é gerente ou não-gerente. As semelhanças de comportamentos entre esses dois tipos de funcionários permitem definir uma relação de supertipo / subtipo entre os mesmos: ambos são admitidos, reagem a eventos como reajustes salariais e atrasos. Funcionário gerentes e funcionário não-gerentes diferem somente no valor do salário base. Definimos, portanto, uma

classe Funcionario que representa os funcionários não-gerentes e será a superclasse da subclasse Gerente.

Os eventos podem ser tratados de maneira semelhante, pois possuem como atributos uma data e uma quantidade associada. Podemos definir uma classe Evento que será supertipo de subclasses específicas para cada tipo de evento: EventoAtraso e EventoReajuste.

Os demonstrativos associados ao funcionário podem ser representados pela classe Demonstrativo. Uma vez que demonstrativos são constituídos por lançamentos, definimos também a classe Lancamento que será responsável por prover acesso à descrição e valor de cada lançamento.

Os funcionários cadastrados no sistema são representados pela classe CadastroFuncionario e a interação do operador com o sistema será de responsabilidade da classe TrmFolhaPgto.

4. Projeto e implementação das Classes

A seguir iremos apresentar o projeto e o código completo de cada uma das classes do sistema da folha de pagamento seguido de comentários relativos aos aspectos particulares de seu projeto e implementação, quando houver. Todas as classes implementadas pertencem ao pacote chamado sisFolhaPgto.

4.1. A classe Evento e suas subclasses

A Figura 3 mostra a classe Evento.

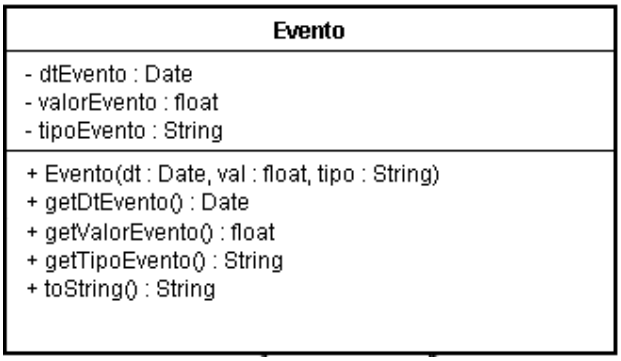


Figura 3 - Classe Evento

No projeto da classe Evento, foram definidos três atributos: dtEvento para a data do evento, o valor do evento chamado valorEvento, quando houver e tipoEvento para identificar o tipo do evento.

Um objeto da classe Evento serve para registrar a ocorrência de um evento e permitir que objetos de outras classes recuperem os valores associados aos atributos desse evento. Para isso são necessárias apenas operações que registrem um evento e leiam os seus atributos, conforme descrito a seguir.

O registro de um evento pode ser realizado pelo construtor Evento(...) da classe, que deverá garantir a consistência dos seus atributos, tal como o limite máximo de 2 horas para um funcionário chegar atrasado à empresa.

Definimos a operação getDtEvento() para retornar a data do evento e a operação getValorEvento() para retornar o valor do evento. Além disso, visando apenas simplificar a utilização da classe Evento por outras classes, temos a operação getTipoEvento(), responsável por retornar o tipo do evento. A identificação do tipo do evento é de responsabilidade do construtor da subclasse.

Finalmente, assim como para todas as classes, iremos redefinir a operação toString() da classe Object. A classe Object é uma classe que serve de superclasse para todas as classes existentes em Java. A operação toString() devolve um String com uma representação do evento, que possa ser impressa ou exibida no terminal.

A seguir a implementação da classe Evento.

```
package sisFolhaPgto;

import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.Date;

public class Evento {

	protected Date dtEvento;
	private float valorEvento;
	private String tipoEvento;

	public Evento(Date dt, float val, String tipo) {
		this.dtEvento = dt;
		Calendar cal = new GregorianCalendar();
		cal.setTime(dtEvento);
		this.valorEvento = val;
		this.tipoEvento = tipo;
	}
	public Date getDtEvento() {
		return this.dtEvento;
	}
	public float getValorEvento() {
		return this.valorEvento;
	}
	public String getTipoEvento() {
		return (this.tipoEvento);
	}

	public String toString()
	{
		return getTipoEvento()+" em " +
		this.dtEvento.toString()+ "valor=" +
		this.valorEvento;
	}
}
```

Com exceção do método construtor, que é exclusivo da classe onde é definido, as demais operações serão herdadas pelas subclasses de Evento. Para cada uma dessas subclasses

deverá, portanto, ser definido apenas o seu construtor e operações específicas da classe, se necessário.

A seguir são apresentadas as implementações das subclasses da classe Evento.

Implementação da classe EventoAtraso.

```
package sisFolhaPgto;

import java.util.Date;

public class EventoAtraso extends Evento {
      public EventoAtraso(Date dt, float qtd) {
            super(dt, qtd, "EventoAtraso");
            if (qtd > 2) {
                  System.out.println("\n (Numero de horas de atraso > 2: "
                              + "nao sera computado um evento do tipo atraso)
\n");
            }
      }
}
```

O construtor da classe EventoAtraso verifica se a quantidade de horas de atraso é maior que dois. Conforme regras utilizadas para calcular o salário mensal, descritas na Seção 1, caso este teste condicional retorne verdadeiro o sistema não deve registrar o evento como um evento de atraso.

Implementação da classe EventoReajuste.

```
package sisFolhaPgto;

import java.util.Date;

public class EventoReajuste extends Evento {
      public EventoReajuste(Date dt, float val) {
            super(dt, val, "EventoReajuste");
      }
}
```

4.2. A Classe Funcionario e Sua Subclasse

A Figura 4 mostra a classe Funcionario e sua subclasse Gerente.

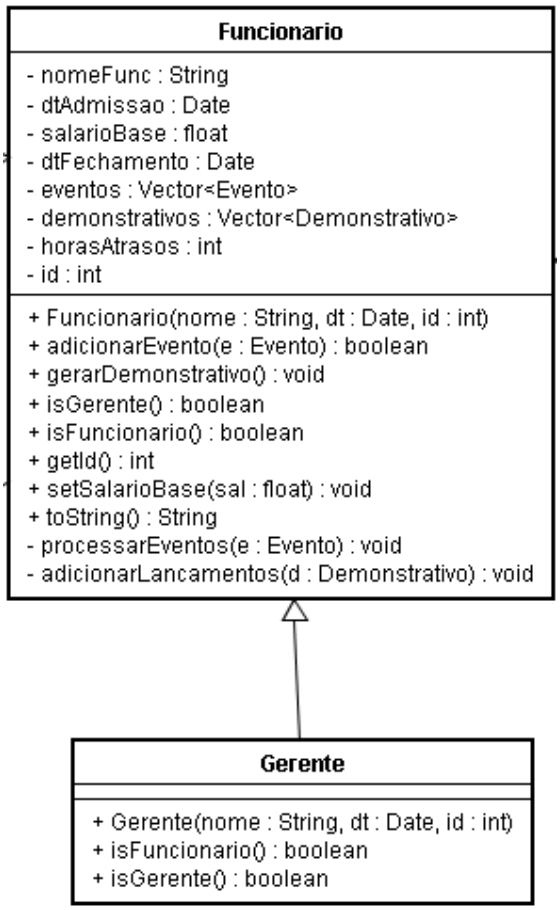


Figura 4 - Classe `Funcionario` e sua subclasse `Gerente`

Nesse nosso estudo de caso iremos nos restringir aos atributos de um funcionário que são essenciais para o cálculo da folha de pagamento, a seguir relacionados:

`nomeFunc e dtAdmissao` - nome do funcionário e data de admissão;

`salarioBase`- salário base atual, que será igual à R$300,00 para não-gerentes e R$400,00 para gerentes, na ausência de reajustes salariais.

`dtFechamento` - data de fechamento do período anterior, que será igual à data de admissão, se recém admitido, ou à data de emissão do último demonstrativo de pagamento;

`eventos` - lista de eventos ocorridos com o funcionário, desde o seu ingresso na empresa;

`demonstrativos` - lista de demonstrativos de pagamento já emitidos para o funcionário;

`horaAtrasos` - quantidade de horas de atrasos associadas ao funcionário.

`id` – identificador único do funcionário a ser usado por outras classes.

Todos esses atributos são comuns a qualquer tipo de funcionário e serão definidos na classe `Funcionario`. O registro de um funcionário pode ser realizado pelo construtor da classe

Funcionario(...), que deverá garantir a consistência de seus atributos, tal como o salário base no valor de R$300,00 para um funcionário não-gerente.

A operação adicionarEvento() é responsável por incluir um evento na lista de eventos do funcionário, após verificar se a data do evento é válida, por exemplo, se a data do evento é posterior a data de fechamento do funcionário.

A operação gerarDemonstrativo() percorre a lista de eventos do funcionário selecionando os eventos posteriores à data de fechamento do funcionário. Para cada um destes eventos é executado a operação processarEventos(), que processa os eventos atualizando os valores dos atributos horas de atrasos e/ou salário base do funcionário conforme o tipo e valor do evento. O registro da admissão de um funcionário pode ser deixado a cargo do método construtor da classe, bem como a especificação do salário base.

Já a operação adicionarLancamentos() é responsável por incluir novos lançamentos no demonstrativo do funcionário. Estes lançamentos correspondem a débitos referentes ao evento atraso, caso exista, e crédito referente ao salário base do funcionário. Os valores a serem incluídos são computados conforme as regras utilizadas para calcular o salário mensal, descritas na Seção 1.

A classe Gerente, assim como a classe Funcionario, possui duas operações que permitem representar melhor as características da classe: isGerente() e isFuncionario().

A seguir as implementações, respectivamente, da classe Funcionario e da classe Gerente.

```
package sisFolhaPgto;


import java.util.Date;
import java.util.Vector;
import java.util.Enumeration;

public class Funcionario {

	private String nomeFunc;
	private Date dtAdmissao;
	private float salarioBase;
	private Date dtFechamento;
	private Vector<Evento> eventos;
	private Vector<Demonstrativo> demonstrativos;
	private int horasAtrasos;
	private int id;

	public Funcionario(String nome, Date dt, int id) {
		this.nomeFunc = nome;
		// salario base do funcionaro: 300
		this.setSalarioBase(300);
		this.dtAdmissao = dt;
		this.dtFechamento = dt;
		this.eventos = new Vector<Evento>(10, 10);
		this.demonstrativos = new Vector<Demonstrativo>(10, 10);
		this.id = id;
	}

	public boolean adicionarEvento(Evento e) {
```

```
            Date hoje = new Date();
            if (!(e.getDtEvento()).after(this.dtFechamento)) {
                  System.out.println
                  ("Evento com data anterior aa do fechamento.");
                  return false;
            } else if ((e.getDtEvento()).after(hoje)) {

                  System.out.println("Evento com data futura.");
                  return false;
            } else {
                  // evento adicionado com sucesso
                  this.eventos.addElement(e);
                  return true;
            }
      }
      public void gerarDemonstrativo() {

            Evento e;
            Demonstrativo d;
            Date hoje = new Date();
            Enumeration<Evento> listaEventos = this.eventos.elements();
            while (listaEventos.hasMoreElements()) {
                  e = (Evento) listaEventos.nextElement();
                  if (e.getDtEvento().after(this.dtFechamento)) {
                        processarEventos(e);
                  }
            }

            d = new Demonstrativo(this, this.dtFechamento, hoje);

            adicionarLancamentos(d);
            this.dtFechamento = hoje;
            this.demonstrativos.add(d);
            d.imprimir();
      }
      public boolean isGerente() {
            return false;
      }
      public boolean isFuncionario() {
            return true;
      }
      public int getId() {
            return id;
      }
      public void setSalarioBase(float sal) {
            this.salarioBase = sal;
      }
      public String toString() {
            return ("Funcionario: " + this.nomeFunc);
      }
      private void processarEventos(Evento e) {
            if (e.getTipoEvento().equals("EventoReajuste"))
                  if (e.getValorEvento() < this.salarioBase)
                        System.out.println
                        ("\n Reajuste menor que salario base."
                                    + " O salario base nao sofrera
alteracoes");
                  else
                        this.salarioBase = e.getValorEvento();
            else if (e.getTipoEvento().equals("EventoAtraso"))
                  this.horasAtrasos += e.getValorEvento();
```

```
            else
                  System.out.println
                  ("Evento invalido para funcionario.");
      }

      private void adicionarLancamentos(Demonstrativo d) {

            float valor;
            float salarioHora;
            salarioHora = this.salarioBase / 176;
            d.incluirCredito("Salario Base",
                        this.salarioBase);
            if (this.horasAtrasos > 0) {
                  valor = this.horasAtrasos * salarioHora;
                  d.incluirDebito
                  ("Atrasos (" + this.horasAtrasos + " hs)", valor);
            }
      }

}
```

Implementação da classe Gerente:

```
package sisFolhaPgto;

import java.util.Date;

public class Gerente extends Funcionario {

      public Gerente(String nome, Date dt, int id) {
            super(nome, dt, id);

            // salario base gerente: 400,
            this.setSalarioBase(400);
      }
      public boolean isFuncionario() {
            return false;
      }
      public boolean isGerente() {
            return true;
      }
}
```

4. 3. A classe Demonstrativo e a classe Lancamento

A Figura 5 mostra a classe Demonstrativo.

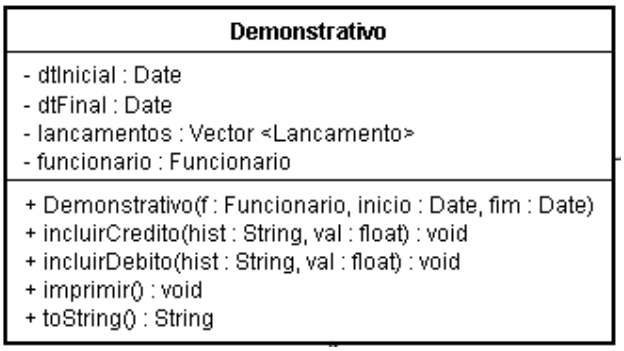


Figura 5 - Classe Demonstrativo

No projeto da classe Demonstrativo foram definidos quatro atributos, dtInicial e dtFinal que representam, respectivamente, a data inicial e a data final do demonstrativo, um conjunto de lançamentos do demonstrativo chamado lancamentos e o funcionario ao qual o demonstrativo está associado.

As operações incluirCredito(...) e incluirDebito(...), são responsáveis por, adicionar lançamentos com valores, respectivamente, positivo e negativo no demonstrativo. A operação imprimir() imprime o demonstrativo especificando o seu período, o funcionário associado e a descrição e valor de todos os lançamentos do demonstrativo.

A seguir a implementação da classe Demonstrativo.

```
package sisFolhaPgto;

import java.util.Date;
import java.util.Enumeration;
import java.util.Vector;

public class Demonstrativo {
	private Date dtInicial;
	private Date dtFinal;
	private Vector<Lancamento> lancamentos;
	private Funcionario funcionario;
	public Demonstrativo(Funcionario f, Date inicio, Date fim) {
		this.dtInicial = inicio;
		this.dtFinal = fim;
		this.lancamentos = new Vector<Lancamento>(10, 10);
		this.funcionario = f;
	}
	public void incluirCredito(String hist, float val) {
		lancamentos.addElement(new Lancamento(hist, val));
	}
	public void incluirDebito(String hist, float val) {
		lancamentos.addElement(new Lancamento(hist, -val));
	}
	public void imprimir() {
		Lancamento l;
		float total = 0;
		System.out.println(this);
		Enumeration<Lancamento> lista = this.lancamentos.elements();
		while (lista.hasMoreElements()) {
			l = (Lancamento) lista.nextElement();
			System.out.println(l);
			total += l.getValor();
		}
		System.out.println("Total a pagar: " + total);
	}
	public String toString() {
		return ("Demonstrativo de Pagamento:"
				+ "\n Periodo de " + dtInicial
				+ " a " + this.dtFinal + "\n "
				+ this.funcionario);
	}
}
```

A Figura 6 mostra a classe Lancamento.

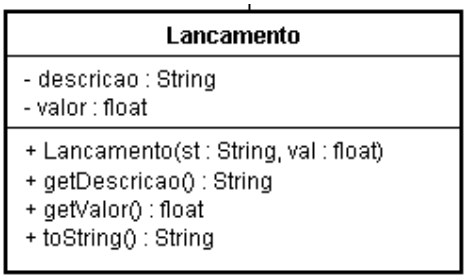


Figura 6 - Classe Lancamento

Para a classe Lancamento foram definidos dois atributos, descricao e o valor do lançamento. O valor pode ser positivo ou negativo.

As operações getDescricao() e getValor() permitem que objetos de outras classes observem, respectivamente, a descrição e o valor de um lançamento.

A seguir a implementação da classe Lancamento.

```
package sisFolhaPgto;

public class Lancamento {
	private String descricao;
	private float valor;

	public Lancamento(String st, float val) {
		this.descricao = st;
		this.valor = val;
	}
	public String getDescricao() {
		return this.descricao;
	}
	public float getValor() {
		return valor;
	}
	public String toString() {
		return (this.descricao + "\t" + valor);
	}
}
```

4.4. A classe CadastroFuncionario

A Figura 7 mostra a classe CadastroFuncionario.

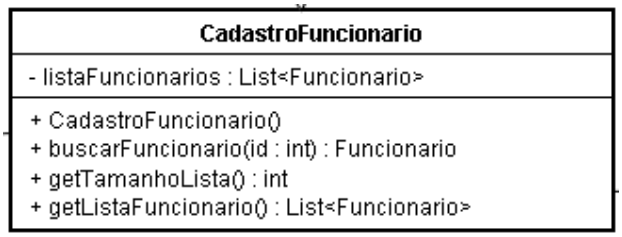


Figura 7 - Classe CadastroFuncionario

O projeto da classe CadastroFuncionario define o atributo lista de funcionários cadastrados chamado listaFuncionarios.

A operação bucarFuncionario(...) permite recuperar um funcionário cadastrado através do seu identificador único. Já as operações getTamanhoLista() e getListaFuncionario() são responsáveis por, respectivamente, retornar o tamanho da lista de funcionários cadastrados e a lista com todos os funcionários cadastrados.

A seguir a implementação da classe CadastroFuncionario.

```
package sisFolhaPgto;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.List;

public class CadastroFuncionario {

	private List<Funcionario> listaFuncionario;

	public CadastroFuncionario() {

		Calendar cal = new GregorianCalendar();
		cal.set(Calendar.MONTH, cal.get(Calendar.MONTH) - 1);
		Funcionario func1 = new Gerente("Joao", cal.getTime(), 0);
		Funcionario func2 = new Gerente("Pedro", cal.getTime(), 1);

		this.listaFuncionario = new ArrayList<Funcionario>();
		this.listaFuncionario.add(func1);
		this.listaFuncionario.add(func2);
	}
	public Funcionario buscarFuncionario(int id) {
		List<Funcionario> listaFunList = this.listaFuncionario;
		if (listaFunList != null)
			for (int i = 0; i < listaFunList.size(); i++) {
				Funcionario funcionario = listaFunList.get(i);
				if (id == funcionario.getId()) {
					return funcionario;
				}
			}
		return null;
	}
	public int getTamanhoLista() {
		return this.listaFuncionario.size();
	}
	public List<Funcionario> getListaFuncionario() {
		return this.listaFuncionario;
	}
}
```

4.5 Classe TrmFolhaPgto

A Figura 8 apresenta a classe TrmFolhaPgto.

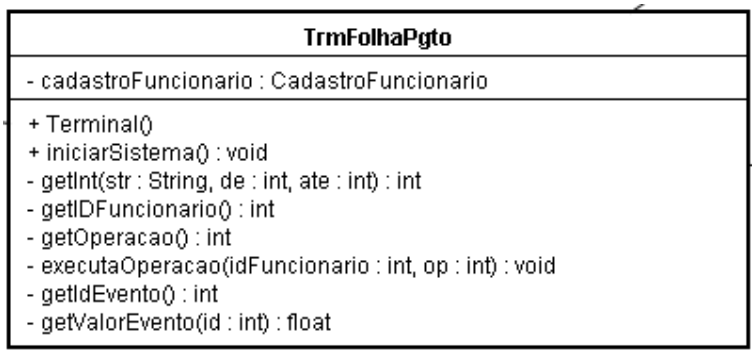


Figura 8 - Classe TrmFolhaPgto

A classe `TrmFolhaPgto` implementa a interação entre operador do sistema e o sistema. A operação `iniciarSistema()` é responsável por iniciar a interface com a qual o operador irá interagir. A operação `getInt()` é responsável por realizar a leitura das opções especificadas através do teclado pelo operador.

Primeiramente, a operação `getIdFuncionario()` solicita ao operador o ID do funcionário para o qual ele deseja adicionar eventos ou gerar demonstrativos. Posteriormente, a operação `getOperacao()` é responsável por exibir o menu "*(1) adicionar evento, (2) gerar demonstrativo (3) sair do sistema"*. Ressaltamos que as operações (1) e (2) são referentes ao funcionário previamente especificado.

Em seguida, o sistema executa a opção escolhida pelo operador do sistema através da operação `executaOperacao(...)`. Caso o operador escolha a opção "*(1) adicionar eventos",* são exibidas mais duas opções "*(1.1) Evento Reajuste (1.2) Evento Atraso",* estas duas opções são exibidas pela operação `getIdEvento()`. A operação `getValorEvento()` permite ao operador especificar o valor do evento.

A seguir é apresentada a implementação da classe `TrmFolhaPagto`.

```
package sisFolhaPgto;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import java.io.StreamTokenizer;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;


public class TrmFolhaPgto {

	private CadastroFuncionario cadastroFuncionario;

	public TrmFolhaPgto() {
		this.cadastroFuncionario = new CadastroFuncionario();
	}
	public void iniciarSistema() {

		int idFuncionario;
		int op;
		System.out.println("\nSistema de Pagamento");
		idFuncionario = this.getIDFuncionario();
		// seleciona usuario
		while (idFuncionario > -1) {
			op = this.getOperacao(); // seleciona operacao
			this.executaOperacao(idFuncionario, op); // e executa
			idFuncionario = this.getIDFuncionario();

		}
	}
	// realiza leitura teclado
	private int getInt(String str, int de, int ate) {
		BufferedReader r =
			new BufferedReader(new InputStreamReader(System.in));
		StreamTokenizer st = new StreamTokenizer(r);
		do {
			System.out.println("Entre com " + str);
			try {
				st.nextToken();
			} catch (IOException e) {
				System.err.println("Erro na leitura do teclado");
				return (0);
			}
		} while (st.ttype != StreamTokenizer.TT_NUMBER || st.nval < de
				|| st.nval > ate);
		return ((int) st.nval);
	}
	private int getIDFuncionario() {

		int qtdFuncionario = this.cadastroFuncionario.getTamanhoLista();
		int id = getInt
		("id do funcionario " + "( ou digite -1 para terminar)",
				-1, (qtdFuncionario - 1));
		return id;
	}
	private int getOperacao() {
		int op = getInt("operacao:\n1=adicionar evento,"
				+ " 2=gerar demonstrativo, 3=sair", 1, 4);
		return (op);
	}
	private void executaOperacao(int idFuncionario, int op) {
		Funcionario f1 = this.cadastroFuncionario
				.buscarFuncionario(idFuncionario);
```

```
            switch (op) {
            case 1:
                  Calendar cal = new GregorianCalendar();
                  Date hoje = cal.getTime();
                  int idEvento = this.getIdEvento();
                  float valor = this.getValorEvento(idEvento);
                  if (idEvento == 1)
                        f1.adicionarEvento
                        (new EventoReajuste(hoje, valor));
                  else
                        f1.adicionarEvento
                        (new EventoAtraso(hoje, valor));
                  break;
            case 2:
                  f1.gerarDemonstrativo();
                  break;
            }
      }
      private int getIdEvento() {

            int idEvento = getInt
            ("evento:\n1= reajuste," + " 2= horas atraso", 1,
                        2);
            return idEvento;
      }
      private float getValorEvento(int id) {

            int valor = 0;
            switch (id) {
            case 1:
                  valor = getInt
                  ("valor do reajuste: \n", 1, 50000);
                  break;
            case 2:
                  valor = getInt
                  ("o numero de horas de atraso: \n", 1, 2);
                  break;
            default:
                  System.err.println("Opcao invalida");
            }
            return valor;
      }
}
```

5. O Programa Principal

A classe Principal tem como objetivo único inicializar o sistema através da classe TrmFolhaPgto. A fim de testar as classes definidas, foi feita uma simulação com apenas dois funcionários previamente definidos na classe CadastroFuncionario. Ainda, todos os eventos adicionados são registrados com a data corrente. A seguir a classe Principal.

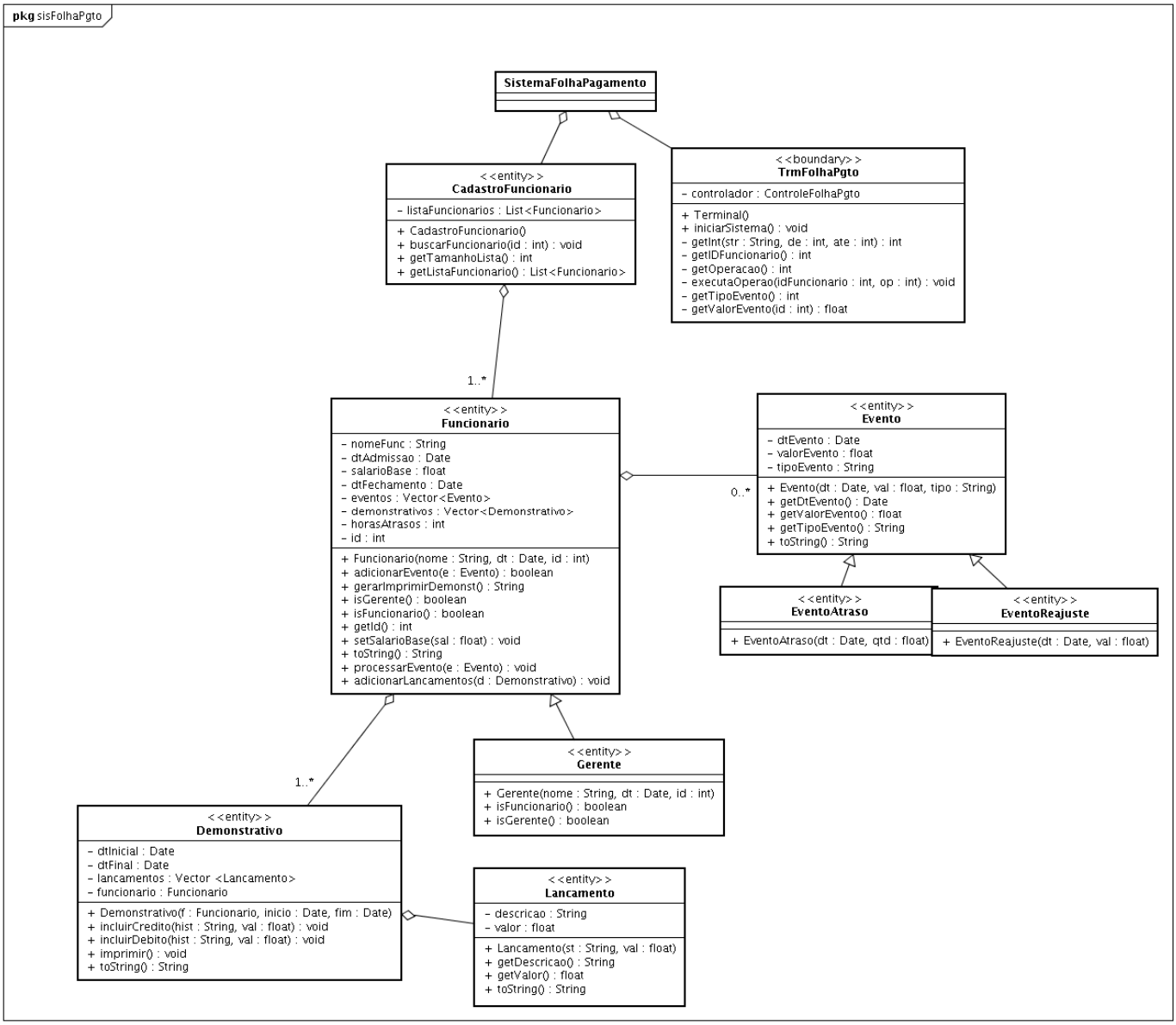
```
package sisFolhaPgto;
public class Principal {
      public static void main(String Arg[]) {
            TrmFolhaPgto simulacao = new TrmFolhaPgto();
            simulacao.iniciarSistema();
      }
```

```
}
```

Segue um exemplo de sessão de teste deste sistema de folha de pagamento. As entradas fornecidas são destacadas em negrito.

```
Sistema de Folha de Pagamento
Entre com id do funcionario ( ou digite -1 para terminar)
1
Entre com operacao: 1=adicionar evento, 2=gerar demonstrativo, 3=sair
1
Entre com evento:
1= reajuste, 2= horas atraso
1
Entre com o valor do reajuste:
500
Entre com id do funcionario ( ou digite -1 para terminar)
1
Entre com operacao:
1=adicionar evento, 2=gerar demonstrativo, 3=sair
1
Entre com evento:
1= reajuste, 2= horas atraso
2
Entre com o numero de horas de atraso:
1
Entre com id do funcionario ( ou digite -1 para terminar)
1
Entre com operacao:
1=adicionar evento, 2=gerar demonstrativo, 3=sair
2
Demonstrativo de Pagamento:
 Periodo de Tue Mar 10 19:32:36 BRT 2009 a Fri Apr 10 19:32:50 BRT 2009
 Funcionario: Pedro
Salario Base      500.0
Atrasos (1 hs)    -2.840909
Total a pagar: 497.1591
```

6. Diagrama de classes do Sistema de Folha de Pagamento



7. Instruções Compilar/Executar o Sistema de Folha de Pagamento

As instruções a seguir são apropriadas tanto para sistemas Unix quanto Windows. Ressalta-se que o código-fonte só é compatível com JDK 6 ou maior.

1- Descompacte o arquivo SistemaFolhaPagamentoVersao1_1.zip
2- Abra uma janela de comando
3- Mude para o diretório: SistemaFolhaPagamentoVersao1_1
4- Compile o arquivo
javac sisFolhaPgto/Principal.java
5- Execute o arquivo Principal.java através do comando
java sisFolhaPgto/Principal

Obs.: Caso o "Sistema de Folha de Pagamento" seja seu primeiro programa Java, o site http://java.sun.com/docs/books/tutorial/getStarted/cupojava/index.html pode ser útil.