

**UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO  
CENTRO TECNOLÓGICO  
DEPARTAMENTO DE INFORMÁTICA**

# **Primeiro Trabalho de Programação III**

## **Relatório**

**Matheus Gomes Arante de Souza  
Matheus Salomão**

**Vitória  
Outubro de 2018**

**UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO  
CENTRO TECNOLÓGICO  
DEPARTAMENTO DE INFORMÁTICA**

**Matheus Gomes Arante de Souza  
Matheus Salomão**

# **Primeiro Trabalho de Programação III**

## **Relatório**

Trabalho referente à disciplina de Programação III do curso de Ciência de Computação do Departamento de Informática da Universidade Federal do Espírito Santo.

**Vitória  
Outubro de 2018**

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Implementação das Classes do Projeto</b>	<b>2</b>
2.1	Diagrama UML . . . . .	2
2.2	Candidato . . . . .	3
2.2.1	Atributos . . . . .	3
2.2.2	Função compare . . . . .	3
2.2.3	Função ToString . . . . .	3
2.3	Partido . . . . .	4
2.3.1	Atributos . . . . .	4
2.3.2	Função votosTotais . . . . .	4
2.3.3	Função numCandidatosEleitos . . . . .	4
2.3.4	Função dadosDoPartido . . . . .	4
2.3.5	Função compare . . . . .	4
2.4	Coligação . . . . .	5
2.4.1	Atributos . . . . .	5
2.4.2	Função ToString . . . . .	5
2.4.3	Função votosTotais . . . . .	5
2.4.4	Função dadosColigacao . . . . .	5
2.4.5	Função compare . . . . .	5
2.5	Eleição . . . . .	6
2.5.1	Atributos . . . . .	6
2.5.2	Construtor . . . . .	6
2.5.3	Função numVagas . . . . .	6
2.5.4	Função totalVotosNominais . . . . .	6
2.5.5	Função dadosDosPartidos . . . . .	6
2.5.6	Função dadosDasColigacoes . . . . .	7
2.5.7	Função candidatosEleitos . . . . .	7
2.5.8	Função candidatosMaisVotados . . . . .	7
2.5.9	Função votacaoMajoritaria . . . . .	7
2.5.10	Função beneficiadosSistProp . . . . .	7
2.6	Testador . . . . .	8
2.6.1	Função LerArquivo . . . . .	8
2.6.2	Main . . . . .	8
<b>3</b>	<b>Conclusão</b>	<b>9</b>
<b>4</b>	<b>Referências</b>	<b>10</b>

# 1 Introdução

Esse trabalho tem como principal objetivo pôr em prática os conhecimentos adquiridos em sala de aula sobre programação orientada a objetos, mais especificamente a linguagem de programação Java.

Dado um arquivo de entrada com os resultados de uma eleição municipal, o código visa armazenar e analisar essas variáveis retornando na saída padrão os seguintes relatórios:

- Número de vagas;
- Candidatos eleitos;
- Candidatos mais votados dentro do número de vagas;
- Candidatos não eleitos e que seriam eleitos se a votação fosse majoritária;
- Candidatos eleitos no sistema proporcional vigente, e que não seriam eleitos se a votação fosse majoritária;
- Votos totalizados por coligação ou partido;
- Votos totalizados por partido;
- Total de votos nominais.

Cada um dos relatórios estará ordenado a partir do número de votos contabilizados pelo item em questão e cada um destes apresentará informações consideradas pertinentes, por exemplo, junto ao vereador estará vinculado seu partido, coligação e número de votos.

Este relatório tem como objetivo apresentar cada passo do código, explicando cada classe e função criada, além das abordagens escolhidas e os problemas e dificuldades que ocorreram durante o processo de desenvolvimento do trabalho.

O processo de explicação do código será desenvolvido em etapas. Cada classe será dividida em um capítulo específico da seção de Implementação das Classes do Projeto[2] e dentro de cada capítulo serão explicadas as funções e atributos referentes a classe.

## 2 Implementação das Classes do Projeto

### 2.1 Diagrama UML

Diagrama de classes criado para fornecer os aspectos estruturais relativos a implementação deste projeto, apresentando os atributos, operações e as relações entre cada classe.

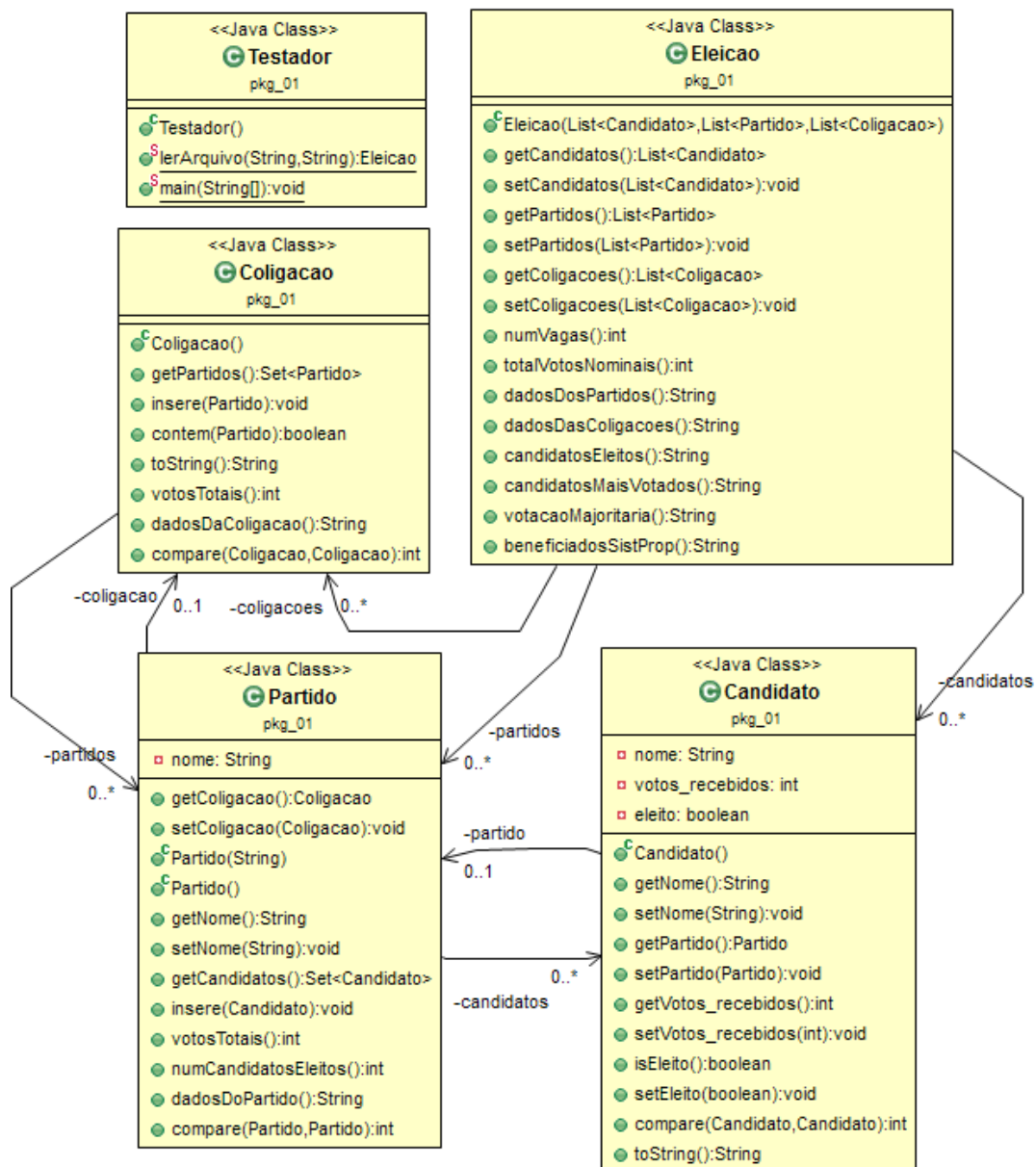


Figura 1: Diagrama de classes

## 2.2 Candidato

### 2.2.1 Atributos

A classe de Candidato possui os seguintes atributos:

- o nome do candidato;
- a referência para o partido que ele pertence;
- o número de votos recebidos;
- e o campo eleito.

Cada um desses atributos possui sua função getter e setter correspondente.

### 2.2.2 Função compare

A função compare recebe dois vereadores e retorna um inteiro. Através da subtração do número de votos recebidos do primeiro pelo número de votos do segundo ela possui três tipos de retorno diferentes:

- $> 0$ , se o número de votos do primeiro for maior que do segundo;
- $= 0$ , caso os valores sejam iguais;
- $< 0$ , caso o número de votos do primeiro seja menor que o do segundo.

Essa função foi criada para que a lista de vereadores possa ser ordenada posteriormente, a fim de plotar os resultados de forma decrescente de número de votos.

### 2.2.3 Função ToString

Essa função é responsável por retornar uma String com as informações necessárias sobre o vereador. Essa função possui um if-else para controlar o número de informações passadas. Caso o candidato não esteja em uma coligação com mais de um partido não é necessário informar a coligação de que ele pertence pois isso representaria apenas o seu próprio partido.

As ideias que giram em torno da abordagem de um partido sem coligação ser encaixado em uma coligação com apenas um partido será abordada no capítulo referente a classe Coligação [2.4].

## **2.3 Partido**

### **2.3.1 Atributos**

A classe Partido possui os seguintes atributos:

- o nome do partido;
- a lista de vereadores afiliados ao partido;
- a referência para a coligação do partido.

O nome do partido e a sua coligação possuem suas respectivas funções getter e setter. Por outro lado, a lista de vereadores possui uma função getCandidatos, responsável por retornar uma cópia da lista, e uma função insere para adicionar um novo vereador a lista.

### **2.3.2 Função votosTotais**

Essa função consiste em somar o número de votos recebidos por cada vereador afiliado ao partido, isto é, da lista de vereadores do partido e retornar o resultado total.

### **2.3.3 Função numCandidatosEleitos**

Mais uma vez acessando a lista de vereadores do partido, essa função verifica se o campo eleito do vereador é verdadeiro, caso seja, uma variável auxiliar é incrementada. Ao final do loop é possível saber o número de vereadores eleitos que pertencem àquele partido. Esse valor é retornado pela função.

### **2.3.4 Função dadosDoPartido**

O nome do partido, número total de votos contabilizados e o número de candidatos eleitos é adicionada a uma String. Além disso, existe um if-else no final dessa função para determinar a flexão de número do substantivo "eleito" que será adicionado no fim da String, que será retornada após isso.

### **2.3.5 Função compare**

Assim como a classe Candidato, esta classe também possui uma função que compara dois objetos, no caso partidos, a partir do seu número de votos contabilizados. A comparação se através do mesmo método, ou seja, uma subtração que pode retornar um número positivo, negativo ou zero, cada uma com seu respectivo significado. Para mais informações verifique a função compare da classe Candidato [2.2].

## **2.4 Coligação**

### **2.4.1 Atributos**

A classe Coligação possui os seguintes atributos:

- uma lista de partidos

E relacionado a essa lista existe uma função que retorna uma cópia dessa lista, denominada getPartidos, uma função insere que adiciona um novo partido a coligação e uma função contem que verifica se um partido está contido na lista de partidos da coligação.

### **2.4.2 Função ToString**

Essa função é responsável por criar uma String contendo o nome de cada partido pertencente a coligação. Seguindo o mesmo padrão do arquivo de entrada, cada partido é separado por uma barra.

### **2.4.3 Função votosTotais**

Através de um loop, os votos de cada partido filiado a coligação são somados e o resultado final, que representa o número de votos totais contabilizados pela coligação são retornados.

### **2.4.4 Função dadosColigacao**

Inicialmente, o número de votos totais recebidos e o número de candidatos eleitos pela coligação são contados e adicionados a variáveis. Através da função ToString a lista de partidos é adicionada a uma String vazia, e em seguida o número de votos e candidatos eleitos.

A função possui um if-else para determinar a flexão de número correta ao substantivo relacionado ao número de vereadores eleitos.

### **2.4.5 Função compare**

A coligação também possui uma função de comparação própria que determina qual coligação é "maior" ou "menor" a partir do número de votos. Essa função segue o mesmo raciocínio das funções de comparação da classe Candidato [2.2] e Partido [2.3].

Para mais informações verifique a função compare no capítulo sobre a classe Candidato [2.2].



## **2.5 Eleição**

### **2.5.1 Atributos**

A classe Eleição possui os seguintes atributos:

- uma lista de vereadores;
- uma lista de partidos;
- uma lista de coligações.

Cada uma com seus respectivos getters e setters.

### **2.5.2 Construtor**

O construtor de uma eleição recebe a lista de candidatos, a lista de partidos e a lista de coligações. Essas listas são ordenadas em ordem decrescente através das funções de comparação que foram definidas para cada uma dessas classes. Somente após serem ordenadas que elas são adicionadas a eleição.

### **2.5.3 Função numVagas**

Essa função é responsável por contar o número total de vagas da eleição. Considerando o número de vagas inicialmente como zero, a lista de vereadores é percorrida e caso o candidato em questão tenha sido eleito o número de vagas é incrementado. Ao final desse processo o valor é retornado pela função.

### **2.5.4 Função totalVotosNominais**

Através de um loop, a lista de vereadores é percorrida por completo para somar todos votos e determinar o total de votos contabilizados pela eleição.

### **2.5.5 Função dadosDosPartidos**

Essa função tem como objetivo reunir as informações necessárias para plotar a lista de votação (nominal) dos partidos, apresentando também o número de candidatos eleitos pelo partido.

Através de um loop a lista de partidos é percorrida. A cada iteração, um partido é selecionado, a função dadosDoPartido da classe Partido [2.3] é chamada, retornando portanto o nome do partido, o número de votos recebidos e candidatos eleitos. E além disso, o número de votos é incrementado a uma variável denominada votostotais, inicialmente setada com zero, que no final da função retornará o total de votos contabilizados durante a eleição.

É importante mencionar que caso o partido não possua candidatos, ele não será apresentado na lista de partidos.

### **2.5.6 Função dadosDasColigacoes**

Essa função é responsável por agrupar dados para plotar a lista de votação (nominal) das coligações e numero de candidatos eleitos em cada uma delas.

Percorrendo a lista de coligações, são armazenadas em uma string as informações de cada uma delas e no final esse conjunto de dados é retornado.

### **2.5.7 Função candidatosEleitos**

Essa função é responsável por reunir informações sobre os candidatos que foram eleitos e retornar uma string contendo esses dados.

### **2.5.8 Função candidatosMaisVotados**

A partir da lista de candidatos, serão agrupadas informações sobre os candidatos mais votados, em ordem decrescente de votação e respeitando numero de vagas.

### **2.5.9 Função votacaoMajoritaria**

Essa função reúne informações sobre os candidatos que teriam sido eleitos se a votação fosse majoritária e retorna uma string contendo os dados destes candidatos.

### **2.5.10 Função beneficiadosSistProp**

Essa função consiste em reunir informações sobre os candidatos que se beneficiaram do sistema proporcional (com sua posição no ranking de mais votados).

A lista de candidatos, que está ordenada de forma decrescente de número de votos, é percorrida a partir da posição  $\text{num-vagas}+1$ , caso o candidato tenha sido eleito, o toString dele é concatenado a uma string que será retornada ao final da função, representando portanto, que ele se beneficiou do sistema proporcional.

## **2.6 Testador**

### **2.6.1 Função LerArquivo**

Dado um arquivo de entrada, através de scanners a função ignora o cabeçalho, isto é, a primeira linha do arquivo, e analisa cada linha individualmente. As linhas são divididas em campos separados por um ";", e para a leitura de cada campo o scanner utiliza o mesmo tipo de delimitador.

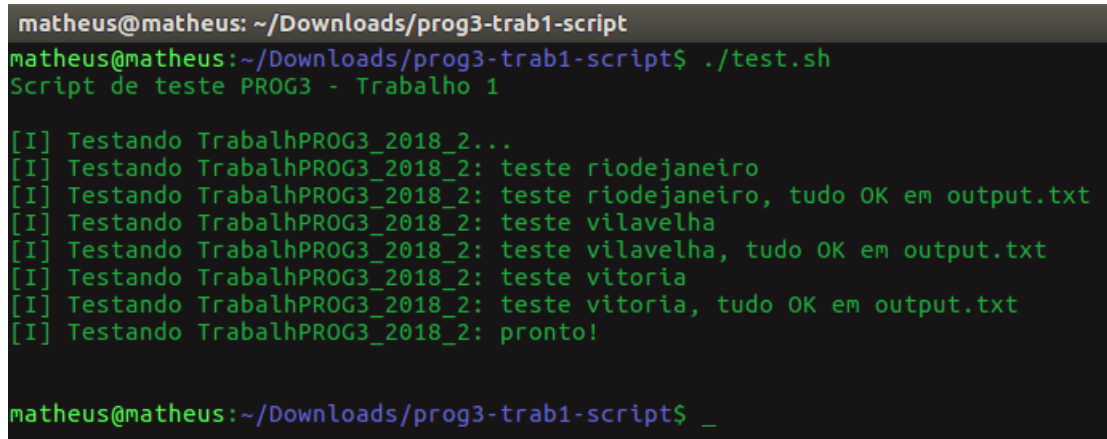
A partir da leitura dos campos é definido o nome do candidato, seu partido e coligação, o número de votos recebidos e se ele foi eleito ou não. Demais informações são ignoradas. Com esses dados são geradas listas de candidatos, partidos e coligações, e estas são adicionadas a uma eleição [2.5] que é retornada ao final da função.

### **2.6.2 Main**

A função main do código cria uma eleição que recebe os dados retornados pela função LerArquivo, e partir disso, ela chama as funções responsáveis por gerar os relatórios especificados na introdução do relatório [1].

### 3 Conclusão

Utilizando o script de teste, disponibilizado no site da disciplina, foi possível notar, com o auxílio do utilitário `diff`, que estávamos gerando resultados relativamente diferentes em relação aos arquivos `output.txt` utilizados como comparativo. Entretanto, como se tratavam de diferenças sutis, foi possível ajustar sem maiores dificuldades a saída gerada pelo nosso programa para ficar de acordo com o padrão esperado.

A terminal window with a dark background and light green text. The prompt is 'matheus@matheus: ~/Downloads/prog3-trab1-script'. The user enters './test.sh'. The output shows the script running tests for 'TrabalhPROG3\_2018\_2' with various cities: riodejaneiro, vilavelha, and vitoria. Each test is reported as 'tudo OK em output.txt'. The prompt returns to the user's shell.

```
matheus@matheus: ~/Downloads/prog3-trab1-script
matheus@matheus:~/Downloads/prog3-trab1-script$ ./test.sh
Script de teste PROG3 - Trabalho 1

[I] Testando TrabalhPROG3_2018_2...
[I] Testando TrabalhPROG3_2018_2: teste riodejaneiro
[I] Testando TrabalhPROG3_2018_2: teste riodejaneiro, tudo OK em output.txt
[I] Testando TrabalhPROG3_2018_2: teste vilavelha
[I] Testando TrabalhPROG3_2018_2: teste vilavelha, tudo OK em output.txt
[I] Testando TrabalhPROG3_2018_2: teste vitoria
[I] Testando TrabalhPROG3_2018_2: teste vitoria, tudo OK em output.txt
[I] Testando TrabalhPROG3_2018_2: pronto!

matheus@matheus:~/Downloads/prog3-trab1-script$ _
```

Figura 2: Execução do script de testes no terminal.

Além dos arquivos de entrada utilizados para efetuar os testes citados anteriormente, também testamos com as seguintes cidades:

- Manaus
- São Paulo

Por meio deste trabalho foi possível colocar em prática diversos conceitos apresentados em sala de aula sobre a linguagem Java e notar as nuances do paradigma orientado ao objeto.

## 4 Referências

- <https://nemo.inf.ufes.br/jpalmeida/ensino/2018-02-prog-iii/>