Primeiro Relatório de Programação III

Elaine Dias Pires
Filipe Gomes Arante de Souza

6 de fevereiro de 2022

Sumário

1 Introdução

Este trabalho tem como objetivo praticar os conceitos de Programação Orientada a Objetos (POO) vistos em aula. Para isso, implementou-se um sistema que processava dados referentes à votação em municípios para o cargo de vereador. Uma vez que o Sistema Eleitoral brasileiro é um sistema proporcional, foi preciso analisar tanto os votos nominais de cada candidato, como também os votos de legenda de cada partido, além de verificar se os mesmos eram votos válidos. Assim, foram gerados diversos relatórios sobre os candidatos que foram eleitos, sobre os candidatos que teriam sido eleitos, caso a eleição fosse majoritária, entre outros.
A linguagem de programação utilizada pra praticar os conceitos de POO neste trabalho foi Java.

2 Desenvolvimento

2.1 Diagrama UML

Obs: Para uma melhor visualização, a imagem em PNG será enviada junto aos outros arquivos do trabalho.

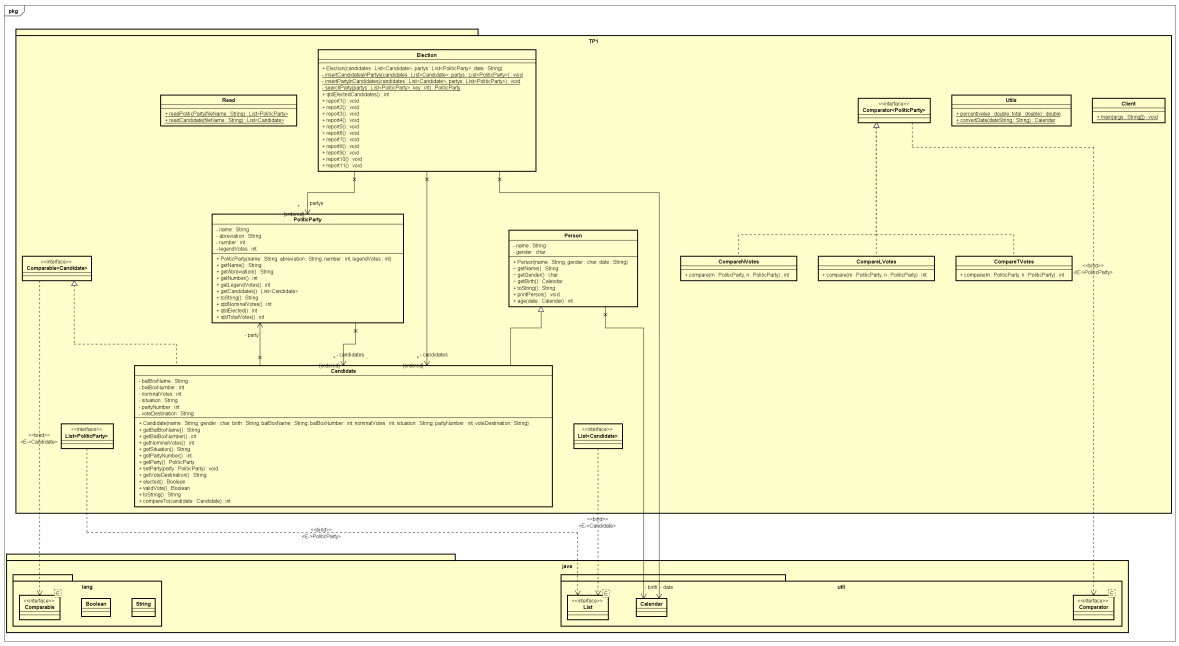

Figura 1: Diagrama UML das classes.

2.2 Leitura dos arquivos de entrada

Nesse trabalho era preciso realizar a leitura de dois arquivos: partidos.csv e candidatos.csv.
Todas as informações de ambos arquivos são passadas para os construtores das classes para criar todos os objetos necessários. Como não sabemos de início a quantidade de partidos e candidatos, e além disso, era necessário realizar diversas ordenações tanto nos partidos quanto nos candidatos, a estrutura de dados escolhida pra armazenar essas informações foi uma lista encadeada. Assim, criou-se uma lista encadeada para partidos e outra para candidatos.

2.3 Classes implementadas

Buscou-se implementar bastante os conceitos de POO no trabalho. Assim, utilizamos construtores em quase todas as classes e também utilizamos a API do java, sobrescrevendo os métodos *toString* e *compareTo*. Também criamos classes implementando nossos próprios comparadores, utilizando a interface *comparable*.

2.3.1 Person

Tendo em vista o conceito de Herança e sabendo que todo candidato é uma pessoa, decidimos criar a classe Person a fim de implementar métodos que estejam mais relacionados a pessoas do que a candidatos. Os atributos da classe pessoa são mostrados abaixo:

- Nome;
- Gênero;
- Data de Nascimento;

2.3.2 Candidate

Essa classe é muito importante para o trabalho. Os atributos e métodos da classe Person são extendidos, sendo acrescentados os seguintes atributos do candidato:

- Nome na urna;
- Número da urna;
- Quantidade de votos nominais;
- Situação;
- Número do partido que pertence;
- Destino de seus votos;
- Partido político;

Principais métodos desta classe:

- **elected**: Verifica se um candidato foi eleito;
- **validVote**: Verifica se os votos de um candidato foram válidos;
- **compareTo**: Auxiliará na ordenação de uma lista de candidatos. Esse método determina o critério de ordenação dos candidatos conforme os votos nominais;

A ordenação dos candidatos é feita pelos votos nominais, caso os votos nominais sejam iguais, o desemparte é feito pela idade, sendo que o candidato mais velho tem prioridade sobre o mais novo.

2.3.3 PoliticParty

Todo partido político possui os seguintes atributos:

- Nome;
- Abreviação;
- Número;
- Quantidade de votos de legenda;
- Lista de candidatos pertencentes ao partido;

Principais métodos desta classe:

- **qtdNominalVotes**: Determina a quantidade de votos nominais de um partido;
- **qtdElected**: Determina a quantidate de candidatos eleitos de um partido;
- **qtdtotalVotes**: Determina o total de votos de um partido (votos nominais + de legenda);

2.3.4 Utils

Criamos essa classe com o intuito de colocar métodos mais genéricos do trabalho, que não fossem específicos de nenhuma classe. Ao final do trabalho, a classe Utils não foi muito utilizada, tendo apenas dois métodos, que são:

- **percent**: Calcula a porcentagem de um valor em relação a um total;
- **convertDate**: Converte uma string que representa uma data para o tipo Calendar;

2.3.5 Read

Essa classe implementa a lógica de leitura dos arquivos partidos.csv e candidatos.csv, armazenando as informações dos mesmos em duas listas encadeadas, uma de partidos e outra de candidatos. Uma vez que os métodos dessa classe são próprios da classe e não de um objeto, esses métodos foram criados como métodos *static* .

2.3.6 Election

Toda eleição possui vários partidos e vários candidatos que tentam se eleger. Após a eleição, são divulgados os dados de quem se elegeu, a qual partido essas pessoas eram filiadas, a quantidade de votos que receberam. Também é comum que sejam geradas estatísticas como a porcentagem de homens e mulheres que se elegeram, entre outros. Pensando nisso, foi criada uma classe Eleição, a qual possui dois atributos:

- Lista de candidatos;
- Lista de partidos políticos;

A classe Eleição é responsável por todas as operações realizadas na lista de partidos e candidatos e é ela também que gera todos os dados provenientes da eleição, isto é, os relatórios pedidos na especificação do trabalho. Assim sendo, todos os seus métodos são muito importantes.

2.3.7 Client

A classe Client é a classe que possui o método main. Essa classe foi criada a fim de realizar as chamadas dos métodos de leitura dos arquivos de entrada e de geração dos relatórios, imprimindo-os na saída padrão.

2.4 Comparadores

Como é pedido mais de uma forma de ordenação nos partidos políticos, foi necessária a criação de três classes de comparadores, cada uma implementando a interface *Compatarator*, a fim de auxiliar na ordenação de partidos: *CompareNVotes* , *CompareLVotes* e *CompareTVotes*. Devido ao fato dessas classes auxiliarem a ordenação de partidos políticos, a implementação das mesmas foi feita no mesmo arquivo da classe PoliticParty.

2.4.1 CompareTVotes

Esta classe cuida do critério de ordenação de partidos políticos conforme o total de votos (nominais e de legenda) do partido. Os critérios de desempate, são, respectivamente, os votos nominais de cada partida e o número partidário. Para ordenar desta maneira, deve-se utilizar o seguinte comando:

Collections.sort(partys, new CompareTVotes());

2.4.2 CompareLVotes

Esta classe cuida do critério de ordenação de partidos políticos conforme a quantidade de votos de legenda do partido. Os critérios de desempate, são, respectivamente, os votos nominais de cada partido e o número partidário. Para ordenar desta maneira, deve-se utilizar o seguinte comando:

Collections.sort(partys, new CompareLVotes());

2.4.3 CompareNVotes

Esta classe cuida do critério de ordenação de partidos políticos conforme a quantidade de votos de nominais do cantidato mais votado do partido. Os critérios de desempate, são, respectivamente, os votos nominais de cada partido e o número partidário. Para ordenar desta maneira, deve-se utilizar o seguinte comando:

Collections.sort(partys, new CompareNVotes());

2.5 Exceções

Há dois métodos que podem lançar exceções neste trabalho, ambos na leitura dos arquivos de entrada:

- readPoliticParty;
- readCandidate;

Eles recebem como parâmetro o nome do arquivo a ser lido, portanto caso o arquivo não exista ou não esteja no diretório correto, a exceção *FileNotFoundException* será lançada e a execução do programa será interrompida, imprimindo a seguinte mensagem no terminal:

"An Error has ocurred! FileNotFoundException"

Esta ferramenta disponibilizada pelo java nos ajudou bastante no momento de resolver problemas no desenvolvimento do código fonte.

2.6 Testes realizados

Foi utilizado o script de testes fornecido no AVA, executando testes das eleições dos municípios de Belo Horizonte, Cariacica, Rio de Janeiro, Serra, São Paulo, Vila Velha e Vitória. Nenhuma diferença na saída foi apontada pelo mesmo. Segue abaixo foto ao executar o script:

```
elaine@PcDaElaine:~/elaine16/ufes/script$ ./test.sh
Script de teste PROG3 - Trabalho 1

[I] Testando TP1...
[I] Testando TP1: teste belo-horizonte
[I] Testando TP1: teste belo-horizonte, tudo OK em output.txt
[I] Testando TP1: teste cariacica
[I] Testando TP1: teste cariacica, tudo OK em output.txt
[I] Testando TP1: teste rio-de-janeiro
[I] Testando TP1: teste rio-de-janeiro, tudo OK em output.txt
[I] Testando TP1: teste serra
[I] Testando TP1: teste serra, tudo OK em output.txt
[I] Testando TP1: teste s├úo-paulo
[I] Testando TP1: teste s├úo-paulo, tudo OK em output.txt
[I] Testando TP1: teste vila-velha
[I] Testando TP1: teste vila-velha, tudo OK em output.txt
[I] Testando TP1: teste vit├┤ria
[I] Testando TP1: teste vit├┤ria, tudo OK em output.txt
[I] Testando TP1: pronto!
```

Figura 2: Resultado do script de testes.

3 Conclusão

Os relatórios gerados pelo trabalho estão de acordo com a saída esperada. Ademais, em toda a implementação do trabalho utilizamos bastante os conceitos de programação orientada a objetos, aprimorando e aperfeiçoando os conceitos ensinados em aula. Assim sendo, o objetivo do trabalho foi concluído.

4 Referências

Slides disponibilizados no AVA.