UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA
VITÓRIA, 2023

PROGRAMAÇÃO ORIENTADA A OBJETO - TRABALHO 1

Estatísticas sobre as eleições brasileiras de 2022

Ana Tereza Ribeiro Soares Pereira e Vitor Dadalto Câmara Gomes

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA
VITÓRIA, 2023

Sumário

Ana Tereza Ribeiro Soares Pereira e Vitor Dadalto Câmara Gomes

Introdução

O projeto consiste na análise estatística das eleições brasileiras de 2022 para deputados estaduais e deputados federais. Para isso, utilizamos os dados oficiais das urnas e dos candidatos, disponibilizados no site do Tribunal Superior Eleitoral, e desenvolvemos um programa em Java capaz de gerar as seguintes estatísticas:

I. Número de vagas;
II. Candidatos eleitos;
III. Candidatos mais votados dentre a quantidade de vagas;
IV. Candidatos que não foram, mas seriam eleitos caso a eleição fosse majoritária;
V. Candidatos eleitos no sistema proporcional vigente, e que não seriam eleitos se a votação fosse majoritária;
VI. Votos totalizados por partido e número de candidatos eleitos;
VII. Primeiro e último colocados de cada partido;
VIII. Distribuição de eleitos por faixa etária, considerando a idade do candidato no dia da eleição;
IX. Distribuição de eleitos por sexo;
X. Total de votos;
XI. Total de votos nominais;
XII. Total de votos de legenda.

A seguir, serão detalhados a estruturação do código, a separação das classes, o funcionamento do programa, relatos sobre o período de desenvolvimento e possíveis bugs.

Implementação

O projeto é composto pelas seguintes classes:

1. App.java

   A classe “App” foi criada exclusivamente para armazenar a ‘main’ do programa. Assim, ela é responsável por processar as informações da linha de comando e chamar as funções das demais classes (leitura dos arquivos e geração de relatórios), organizando-as.

2. CSVReader.java:

   A classe “CSVReader” tem como foco a leitura dos arquivos passados na linha de comando, uma vez que reúne todas as funções responsáveis pela leitura e pela manipulação do que foi lido e faz a chamada das funções de armazenamento dos dados lidos. Os atributos dessa classe são os caminhos dos dois arquivos a serem lidos.

3. Election.java:

   A classe “Election” armazena os atributos de uma eleição, sendo eles: um mapa da classe “Candidate“ com todos os candidatos válidos, um mapa da classe “Party“ com os partidos, a data da eleição, a votação a ser analisada (estadual ou federal), o número total de vagas, o número total de votos de legenda, o número total de votos nominais e o número total de

votos. Suas funções consistem basicamente no armazenamento dos dados lidos, em getters dos atributos e na atualização da contagem de votos.

4. Candidate.java:

    Na classe “Candidate” estão os atributos referentes a um candidato, sendo eles: cargo (deputado estadual ou federal), número do candidato, nome na urna, número do partido, acrônimo do partido, número da federação, data de nascimento, gênero, status da eleição (eleito ou não), destinação dos votos (nominal ou de legenda), condição da candidatura, número de votos nominais e idade. As funções dessa classe se resumem a getters dos atributos privados e a funções de armazenamento dos dados nas estruturas, de atualização da quantidade de votos, cálculo da idade e de ordenação.

5. Party.java:

    A classe “Party” contém os atributos de um partido, sendo eles: número, acrônimo, nome, número de candidatos pertencentes ao partido, número de votos de legenda, número total de votos nominais, número de votos totais desse partido e 2 mapas de candidatos pertencentes ao partido, um com os candidatos válidos e outro com os candidatos inválidos. Os candidatos inválidos foram armazenados pois, se seus votos forem destinados para legenda, mesmo que ele tenha sido indeferido, serão contabilizados. As funções dessa classe se resumem a getters dos atributos privados e a funções de armazenamento dos dados nas estruturas, de atualização da quantidade de votos e de ordenação.

6. Report.java:

Na classe “Report” estão as funções responsáveis por gerar os relatórios, tanto para padronizar a saída quanto para fazer as operações necessárias para gerá-los.

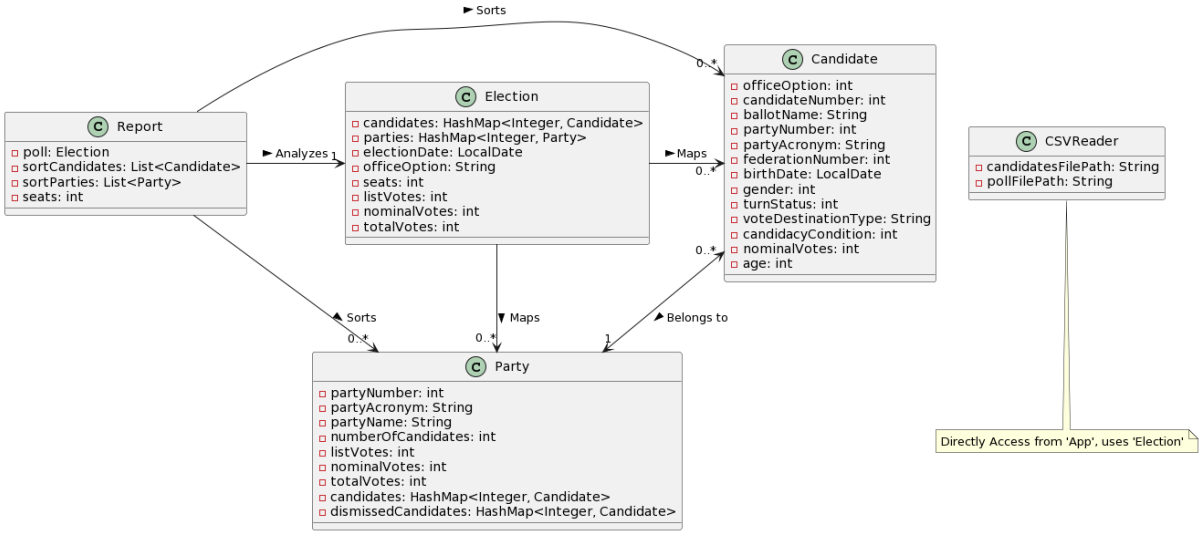

Figura 1 - Diagrama UML do programa desenvolvido

A organização interna do código foi feita na intenção de impedir que o usuário faça alterações nas estruturas de dados. Para isso, todos os atributos das classes são privados, acessíveis apenas mediante getters que retornam valores ou cópias de vetores e mapas, assim deixando intacta a organização feita durante a leitura. Além disso, nossa ‘main’ foi desenvolvida chamando apenas funções das classes “CSVReader”, “Election” (que reúne os atributos gerais de uma eleição) e “Report”, de modo que o usuário não consiga utilizar funções exclusivas de “Candidate” e “Party”.

A única possibilidade de interferência do usuário no código seria por meio da linha de comando, que poderia conter algum caminho incorreto, uma data inválida ou estar digitada de forma despadronizada. Para isso, foi feito um tratamento de exceções que verifica o formato da data inserida e a linha de comando completa e, caso não estejam no formato apropriado, o programa encerra sua execução. Além disso, se a data estiver no formato certo mas, quando for transformar de String para LocalDate houver algum erro, lança o método printStackTrace(), que imprime detalhes da exceção e é uma ferramenta muito útil para diagnosticar o problema. O mesmo tipo de verificação foi feito para a conversão do tipo String para int.

O tratamento descrito acima foi necessário pois desenvolvemos a leitura através do armazenamento de cada linha do arquivo csv como String, seccionando-a pelas vírgulas (e assim identificando as diferentes colunas), transformando e armazenando nas estruturas apenas os dados necessários para a execução do programa, como data de nascimento, número do candidato, quantidade de votos, entre outros, e, por fim, substituindo a String lida pela próxima linha.

Desenvolvimento

A partir do diagrama UML pronto, iniciou-se o desenvolvimento do código pela leitura, utilizando o arquivo do Espírito Santo como único teste nesse período. Nesse momento, todos os candidatos eram armazenados em um mapa geral de candidatos, usando seu número como chave, independente da situação de sua candidatura.

Analisando a quantidade lida de candidatos e a quantidade armazenada, observou-se uma discrepância e, após uma inspeção do arquivo de candidatos, notou-se que o número de um candidato indeferido de um partido pode ser reutilizado para um candidato válido, assim gerando uma sobreposição no mapa e eliminando candidatos

válidos e seus votos. A solução criada para isso foi utilizar dois mapas diferentes, um com candidatos cuja candidatura estava válida (2 ou 16) e outro para os candidatos inválidos cujos votos são destinados para a legenda, que precisam ser contabilizados. Assim, o problema foi reparado e seguimos com o desenvolvimento do programa. Os candidatos identificados com os mesmos números no Espírito Santo foram:

--estadual (4): 530 x 534 (número total de candidatos em nossa estrutura de dados x número total de candidatos no arquivo lido)
SARGENTO RAFAELA (10888): REPUBLICANOS (10)
TENENTE ANDRESA DOS BOMBEIROS (10888): REPUBLICANOS (10)

LOCUTOR FERNANDO LOPES (10345): REPUBLICANOS (10)
NASCIMENTO (10345): REPUBLICANOS (10)
ZANATA (10345): REPUBLICANOS (10);

FABRÍCIA SOARES (35356): PMB (35)
PEDRO TURINI (35356): PMB (35)

--federal (1): 200 x 201 (número total de candidatos em nossa estrutura de dados x número total de candidatos no arquivo lido)
GILDA (1066): REPUBLICANOS (10)
LÍDIA ALVARINO DA SAÚDE (1066): REPUBLICANOS (10)

Após a conclusão da primeira versão da leitura, avançou-se para a organização interna das classes e para o armazenamento dos dados nas estruturas e, posteriormente, para o desenvolvimento dos relatórios, que utilizaram os dados armazenados para calcular as estatísticas e gerar a saída desejada de maneira padronizada. Por fim, foram testados os arquivos das votações dos estados Acre, Alagoas, Minas Gerais, Pernambuco e Rio Grande do Sul por meio de um script de correção.

Em seguida, notou-se que o tempo de execução do programa estava muito alto. Visando otimizar o código e diminuir esse tempo, substituímos o método de leitura de Scanner para BufferedReader. Essa alteração não comprometeu o resultado final do programa e não gerou erros, mas sim reduziu drasticamente o tempo de execução, como pode ser observado na Figura 2.

```
vitordcgomes@DESKTOP-1GKT484:/mnt/c/Users/vitor/PROG/Java/brazilian-election-analysis/trabalho1-poo-script$ ./test.sh
Script de teste PROG OO - Trabalho 1

[I] Testando project...
[I] Testando project: +- teste AC
[I] Testando project: |  teste AC, tudo OK (estadual)
[I] Testando project: |  teste AC, tudo OK (federal)
[I] Testando project: +- teste AL
[I] Testando project: |  teste AL, tudo OK (estadual)
[I] Testando project: |  teste AL, tudo OK (federal)
[I] Testando project: +- teste MG
[I] Testando project: |  teste MG, tudo OK (estadual)
[I] Testando project: |  teste MG, tudo OK (federal)
[I] Testando project: +- teste PE
[I] Testando project: |  teste PE, tudo OK (estadual)
[I] Testando project: |  teste PE, tudo OK (federal)
[I] Testando project: +- teste RS
[I] Testando project: |  teste RS, tudo OK (estadual)
[I] Testando project: |  teste RS, tudo OK (federal)
[I] Testando project: +- pronto!
```

Figura 2 - Saída do script de correção

Bugs

O código foi desenvolvido tentando cobrir o máximo de possíveis erros, mas não conseguimos garantir que todos foram tratados. Um possível bug é que, se for inserida uma data inválida na entrada, mas no formato certo, como 31/02/2022, o código não vai identificá-la como uma data inexistente, assim contabilizando normalmente a idade do candidato com um possível erro neste cálculo.

Conclusão

A partir do desenvolvimento do diagrama UML do nosso projeto, a programação foi desenvolvida organizadamente e seguindo as estruturas pré definidas. Como relatado anteriormente, foram feitas algumas alterações no diagrama conforme o programa foi desenvolvido, mas nenhuma impactou amplamente a estrutura original definida.