Relatório do Trabalho 1 de Programação 3

Danilo Erler Lima e Enzo Baioco Cussuol

Departamento de Informática
Universidade Federal do Espírito Santo
Brasil
Abril 2021

1 Introdução

Este trabalho foi desenvolvido com o intuito de gerenciar e fornecer estatísticas com relação às eleições para vereadores nos municípios brasileiros em 2020. Para tanto, foi utilizada a linguagem Java e a IDE (Ambiente de Desenvolvimento Integrado) Eclipse. Para o gerenciamento de versões e a fim de facilitar o desenvolvimento coletivo, foi utilizado o sistema git por meio da plataforma GitHub (TRABALHO. . . , 2021).

O código foi feito usando o sistema operacional Windows 10, mas tomando-se os cuidados para que este seja funcional em qualquer outro sistema, tal como implica a própria linguagem Java.

2 Implementação

Para implementar o sistema de eleição previamente citado, foram utilizadas 6 classes dentro de um mesmo pacote. Essas classes serão abordadas no decorrer desta seção.

2.1 Leitura

A classe de leitura tem como proposta realizar a coleta dos dados e dessa forma armazenar nas classes de partido e político tudo aquilo que será utilizado posteriormente.

Os dois métodos dessa classe fazem uso do bloco try catch do Java, a fim de tratar exceções na hora de abrir arquivos de entrada. Caso não seja possível realizar a abertura desses arquivos, o bloco catch é invocado, o qual foi implementado de tal forma que a leitura seja bloqueada e seja impressa uma mensagem para o usuário dizendo que o arquivo de leitura não pode ser aberto.

Agora, quanto aos métodos em si, temos:

2.1.1 lePartidos

Basicamente, esse método preenche o conjunto de partidos (armazenados em um HashMap (mais explicações à diante)). Para fazer isso, foi utilizado um scanner sobre um arquivo de input. Esse scanner mais geral lê cada linha do arquivo, e, com a linha em mãos, utiliza-se outro scanner, com o delimitador ",", usado para ler cada campo separado por vírgula, como sugere o uso de arquivos com a extensão csv. Cada campo é então utilizado para criar um novo partido, o qual é inserido no conjunto de partidos.

2.1.2 lePoliticos

Análogo ao método lePoliticos, esse método preenche o conjunto de políticos (armazenados em uma LinkedList). Foi utilizado novamente dois scanners, com a mesma lógica do delimitador. Preenche-se um político com os dados lidos (caso este seja válido) e o insere no conjunto. Contudo, além disso, esse político é inserido no conjunto de partidos, no partido ao qual ele pertence. Por esse motivo, foi escolhido implementar o conjunto de partidos como um HashMap, já que este permite realizar uma busca em tempo constante.

2.2 Politico

A classe político tem como proposta ter as informações referentes a cada candidato presente na eleição, nela estão inclusas os seguintes dados: O nome do candidato, o seu nome

na urna eletrônica, seu número eleitoral e partido, quantidade de votos recebidos, seu sexo e data de nascimento, além de sua situação eleitoral, isso é, se ele foi eleito ou não. Para a organização da classe político, foi implementada uma classe de comparação que toma como critério a quantidade de votos do candidato e em caso de empate utiliza-se a data de nascimento como desempate.

Essa classe possui apenas métodos de getters e setters, então, não convém os listar, uma vez que seria repetitivo.

2.3 Partido

A classe partido tem como proposta ter as informações referentes a cada partido presente na eleição, nela estão inclusas os seguintes dados: O nome e sigla do partido, o número do partido, quantidade de votos de legenda, quantidade total de votos, número de candidatos eleitos, além disso inclui uma lista com todos candidatos do partido, além de seu melhor e pior candidato. Para a organização da classe partido, foram implementadas duas classes de comparação, uma com critério de comparação o número de votos do partido, e outra comparando o número de votos do candidato com mais votos do partido, ambas com critério de desempate o número do partido.

Novamente essa classe possui apenas métodos de getters e setters, então, estes não serão explicitados neste relatório.

2.4 Eleição

A classe eleição possui, como proposta principal, processar informações chaves a respeito das eleições que são de interesse para os relatórios de saída, tais como número de votos dos partidos, número de vagas ocupadas por candidatos eleitos, etc.

Além disso, essa classe também armazena dados como: a data da eleição, o conjunto de partidos e o conjunto de políticos.

O construtor dessa classe segue os seguintes passos:

Primordialmente, é obtida a data da eleição. Para isso, foi utilizada uma classe fornecida pelo Java 8, a classe LocalDate. Para armazenar corretamente essa data, é preciso usar um formatador. Então, foi utilizada a classe DateTimeFormatter que fornece o método ofPattern, o qual recebeu como parâmetro a string "dd/MM/yyyy", que é o estilo de representação utilizado para datas no Brasil.

Posteriormente, o LinkedHashMap e a LinkedList que armazenam os partidos e politicos são preenchidos a partir da leitura dos arquivos de entrada. A escolha pelo HashMap foi explicada na seção de leitura, porém, não foi citado o motivo de ser escolhido um LinkedHashMap ao invés de um HashMap comum. Isso se deve ao fato de que, em um LinkedHashMap, a ordem de inserção é preservada, o que se mostra desejável nessa aplicação, tanto para futuras ordenações quanto para fins de debugging.

Finalmente, são realizados uma série de processamentos a partir dos dados obtidos no passo anterior, tanto para preencher atributos da própria eleição, quanto para partidos e políticos internos.

Aqui, os métodos desenvolvidos foram:

2.4.1 processaNumVagas

Esse método retorna o número de vagas ocupadas por candidatos eleitos. Basicamente, é feito um loop na lista de políticos. Se é encontrado um político com a situação "Eleito",

um contador é incrementado. Após o final do loop, o contador representa o número de vagas.

2.4.2 processaVotosTotaisPartidos

Esse método simplesmente coloca no campo votosTotais de cada partido, o número de votos nominais de cada político desse partido mais o número de votos de legenda desse partido.

2.4.3 processaNumEleitosPartidos

Análogo ao método que processa o número de vagas, esse método contabiliza o número de candidatos com a situação "Eleito", mas, agora, o loop é feito para os candidatos de cada partido. Ao final, é preenchido o campo numEleitos de cada partido.

2.4.4 processaPrimeiroColocadoPartidos

Aqui, é realizado um loop entre os partidos. Para cada partido, todos os políticos que pertencem a esse partido são analisados. O político que tiver maior número de votos nominais será nomeado como o primeiro colocado desse partido. Caso existam mais de um político com número de votos máximo, o primeiro colocado sera escolhido com base na idade, onde o mais velho terá prioridade.

2.4.5 processaUltimoColocadoPartidos

Análogo ao método anterior, mas para obter o último colocado, ou seja, obter o político com o menor número de votos. Também é válida a forma de desempate.

2.4.6 processaTotalVotos

Nesse método, calcula-se o número total de votos contabilizados na eleição. Basicamente é realizado um loop entre os partidos, e um contador soma o número de votos totais de cada partido. Ao final, esse contador guarda o valor desejado, ou seja, o número total de votos contabilizados.

2.4.7 processaVotosLegenda

Análogo ao método anterior, aqui é calculado o número de votos de legenda totais na eleição.

2.4.8 processaVotosNominais

Pode-se dizer que esse método é o mais simples, pois, basicamente retorna a diferença entre o número de votos totais (obtido no método processaTotalVotos) e o número de votos de legenda (obtido no método processaVotosLegenda). Essa diferença será o número de votos nominais contabilizados na eleição.

2.4.9 geraRelatorios

Nesse método, geram-se os relatórios de saída conforme especificado na documentação do trabalho. Para isso, realiza-se uma série de ordenações, utilizando-se a classe Collections (COLLECTIONS..., 2021) com o método sort, seguidas das impressões necessárias explicitadas na classe Escrita. Para ordenar o HashMap, foi necessária uma LinkedList auxiliar, uma vez que o método sort não aceita HashMaps como parâmetro.

2.5 Escrita

A classe escrita tem o objetivo de gerar os resultados esperados de saída, sendo eles principalmente: número de candidatos que foram eleitos, uma lista com todos candidatos eleitos, uma lista com os candidatos com mais votos, uma lista com os candidatos que seriam eleitos numa votação majoritária, e uma lista com os candidatos que foram eleitos, mas não seriam numa votação majoritária. Depois disso, uma lista com os partidos com seus votos e número de candidatos eleitos. Os primeiros e últimos colocados de cada partido. Depois algumas distribuições feitas considerando idade, depois sexo dos candidatos. E por último o número total de votos, votos nominais e de legenda.

Cada método dessa classe gera um relatório dentre os citados acima, por meio de impressões e alguns pequenos processamentos. Não convém os listar explicitamente, pois seria apenas repetitivo e não produtivo.

2.6 Testador

Para o funcionamento do programa foi implementada a classe Testador, que justamente chama os métodos que executam o programa de forma ordenada para que este funcione corretamente.

Primeiro, são criados dois objetos leitura e escrita, que irão realizar o que os nomes sugerem, ler e escrever, respectivamente.

Depois disso, é criado um objeto eleicao, em que o fluxo de execução segue a partir de seu construtor, como já foi citado acima.

Por fim, é chamado o método geraRelatorios do objeto eleicao, o qual funcionamento já foi explicado.

2.7 Diagrama de Classes

Com base nas classes descritas, pode-se elaborar um diagrama UML, que sumariza as informações fornecidas. Os métodos das classes foram omitidos pois são muitos e, muitas vezes, repetitivos, o que atrapalharia a compreensão da relação entre as classes.

A Figura 1 mostra o diagrama, implementado a partir da ferramenta plantUML (PLANTUML..., 2021).

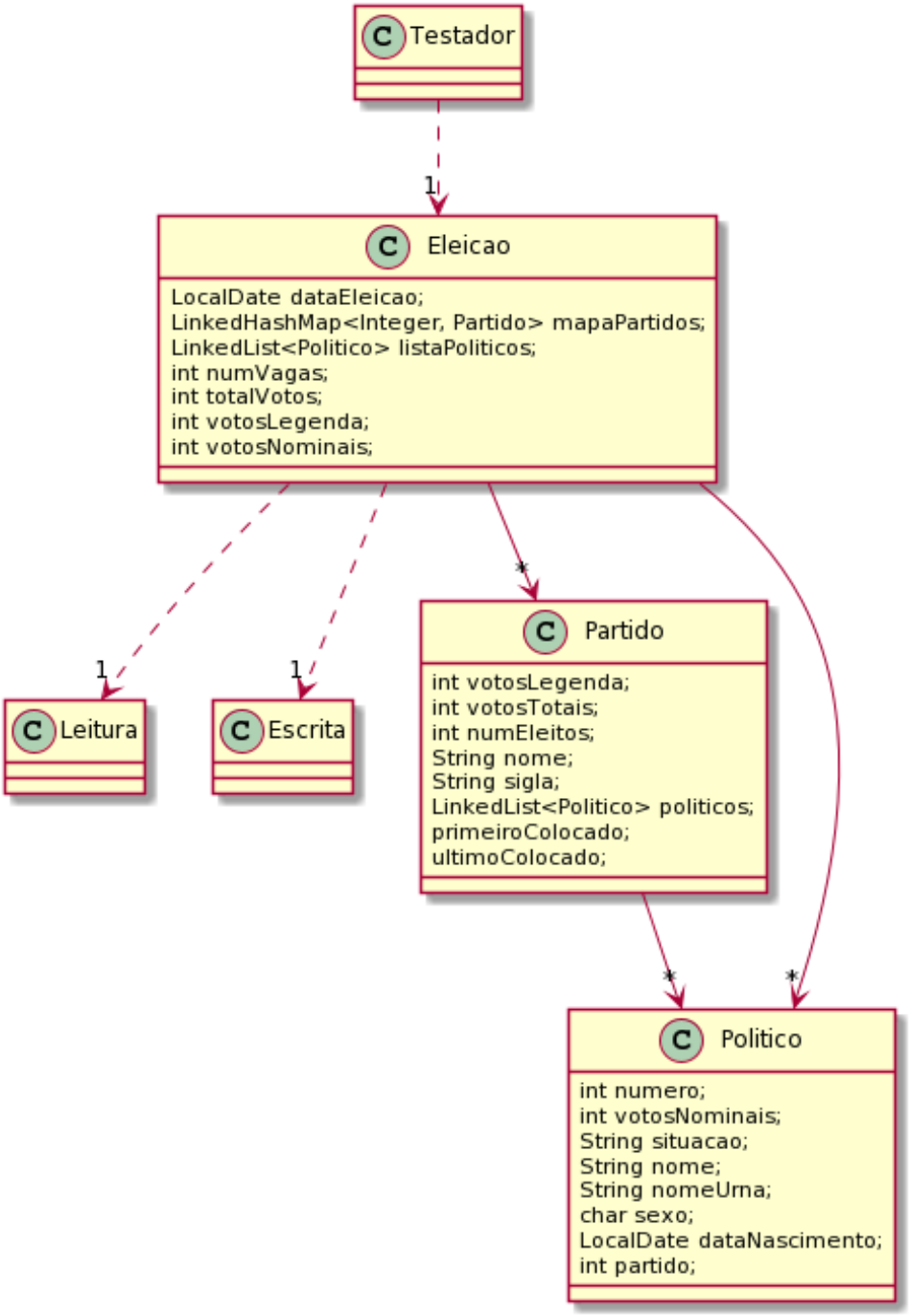

Figura 1: Diagrama de classes UML

3 Testes

Durante o desenvolvimento do trabalho, utilizamos como principal fonte de teste os dados da eleição de vitória de 2020, tendo como parâmetro de saída o exemplo encontrado na própria especificação do trabalho. Contudo, após a divulgação do script de testes, foram encontradas algumas inconsistências no código.

Frente à essas inconsistências, tivemos que consertar algumas partes do programa que estavam erradas. Primeiro, foram adicionadas algumas pequenas lógicas para tratar palavras no plural, por exemplo, nosso código antigo estava imprimindo "1 votos", sendo

que o correto era "1 voto". Além disso, foram adicionadas quebras de linha que estavam faltando.

Por fim, foi adicionado o critério de desempate para primeiros e últimos colocados de cada partido, pois foram observados discrepâncias nos scripts nessa área do código. O critério de desempate, como já citado, foi realizado por idade dos políticos.

4 Bugs

Durante o desenvolvimento um dos problemas encontrados pela dupla, foi o fato de ao realizar a compilação dentro da IDE Eclipse e a compilação realizada a partir do terminal gerarem conflito de versão do Java JDK, dessa forma é fundamental que ao realizar o procedimento de execução realize o passo de ant clean antes, garantindo que evite-se problemas de compatibilidade de versão.

Além disso, foi notado que a IDE Eclipse salva, por padrão, arquivos com a codificação CP1252. Essa codificação estava gerando bugs ao rodar o trabalho em ambientes Linux, uma vez que os caracteres únicos da língua portuguesa não estavam sendo reconhecidos. Então, foi-se necessário alterar a codificação do eclipse, para que este utilize a codificação UTF-8, o que resolveu o problema.

5 Conclusão

A conclusão chegada após a implementação deste trabalho é que este foi fundamental para aprofundar os principais conceitos da linguagem Java e do paradigma de programação orientada à objetos.

Após o término de todo o processo, a dupla se sente mais confiante no que diz respeito ao uso de classes, métodos, construtores, comparadores, exceções, leitura/escrita, e ordenações em Java. Além disso, nos sentimos encorajados para encarar problemas mais complexos, uma vez que agora de fato entendemos a estruturação por trás de uma das linguagens mais famosas de todo o mundo, a linguagem Java.

Referências

ENZO CUSSUOL E DANILO LIMA. **Trabalho 1 de Programação 3**. [S.l.: s.n.], 2021. Disponível em: https://github.com/enzocussuol/Trabalho-1-de-Programacao-3. Acessado em: 07 de Abril de 2021.

ORACLE. **Collections (Java Platform SE 7)**. [S.l.: s.n.], 2021. https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html. Acessado em: 07 de Abril de 2021.

PLANTUML. **PlantUML Web Server**. [S.l.: s.n.], 2021. http://www.plantuml.com/plantuml/uml/. Acessado em: 07 de Abril de 2021.