Marcela Carpenter da Paixão e Maria Luiza Ferreira Reis Santos

Prof. João Paulo A. Almeida

Programação Orientada a Objetos 2024/02

28 de fevereiro de 2025

Trabalho 1 de Programação Orientada a Objetos - Sistema Eleitoral

Este trabalho foi desenvolvido com o objetivo de realizar a análise de dados reais de eleições, com disponibilizados em arquivos CSV pelo TSE, para fazer relatórios como os descritos na especificação do trabalho. Tudo isso, feito usando a linguagem de programação Java.

Para processar os dados dos candidatos, partidos, contabilizar os votos e gerar as estatísticas foram utilizados os conhecimentos adquiridos em sala de aula como locales, formatações, leitura de arquivos, comparators, entre outros.

Implementação

Para esse trabalho, foram utilizados nove arquivos .java, sendo eles: Main, Leitor, Relatorio, Votacao, Partido, Candidato, CandidatoComparator, CandidatoPartidoComparator e PartidoComparator.

O programa inicia na classe *Main*, onde são recebidos como argumentos os nomes dos arquivos contendo os dados da eleição, além do código da cidade e da data da eleição. Se os argumentos forem insuficientes, uma mensagem de erro é exibida e a execução é interrompida. Caso contrário, o processo de leitura dos arquivos começa.

A classe *Leitor* é responsável pela leitura e processamento dos arquivos. No método leCandidatos(), os dados dos candidatos são extraídos utilizando FileInputStream, BufferedReader e Scanner. São armazenadas informações como número, nome, partido, gênero e situação eleitoral. Durante esse processo, partidos novos são cadastrados na eleição e associados aos seus respectivos candidatos. Apenas os

candidatos que atendem aos critérios estabelecidos (como código da cidade e cargo correto) são adicionados à *Votacao*, classe responsável pela apuração das informações, e pelo sistema eleitoral de maneira geral.

Já o método leVotacao() processa o arquivo de votação, onde são identificados os votos nominais e os votos de legenda. Cada voto é direcionado ao candidato correspondente ou, no caso de votos de legenda, ao partido. Se a linha processada não corresponder ao código da cidade ou ao cargo esperado, ela é ignorada.

Após a leitura dos dados, a classe *CandidatoComparator*, que implementa a interface Comparator<Candidato>, ordena os candidatos de forma decrescente, sendo o critério de ordenação o número de votos e, em caso de empate, o mais velho tem prioridade.

Em seguida, a classe *Relatorio* gera os relatórios requisitados na especificação do trabalho sobre o desempenho dos candidatos e partidos, apresentando detalhes e estatísticas relevantes sobre eles. Para cada relatório, foram feitos os cálculos, análises e chamadas de funções para auxiliar na obtenção do resultado esperado.

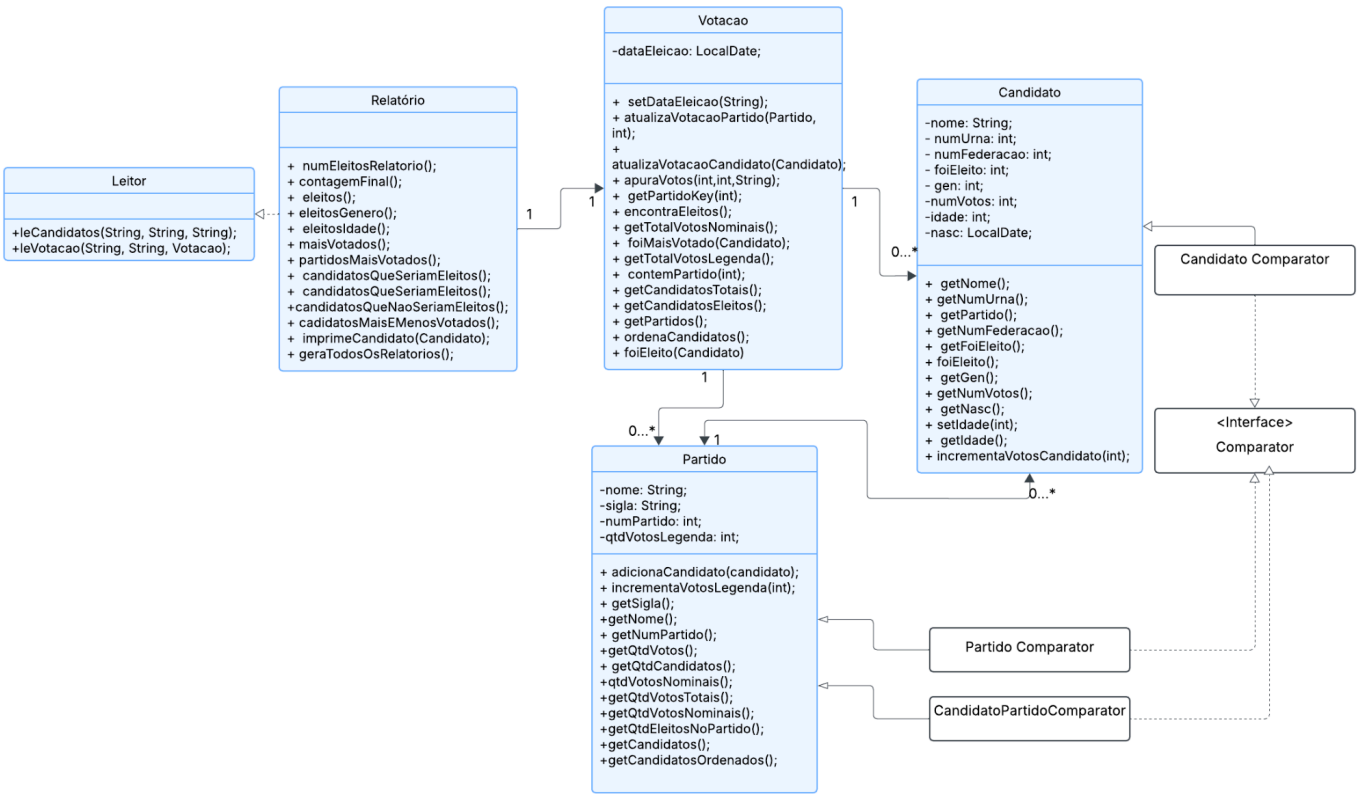
Classes

Implementamos o trabalho usando 5 classes e 3 comparators, os quais estão descritas abaixo:

- **Candidato.java**: reúne todas as informações relevantes do candidato para as gerações dos relatórios. Além disso, faz o incremento dos votos de cada um e calcula a sua idade.
- **Partido.java**: reúne todas as informações relevantes do partido para as gerações dos relatórios. Além disso, faz o incremento dos votos de legenda e agrega a contagem de votos, define a quantidade de eleitos no por partido..
- **Votacao.java:** apura os votos e encaminham eles pros seus devidos partidos e candidatos, retorna os componentes da eleição e as suas quantidades. Traz os candidatos totais e os eleitos em listas ordenadas, e os partidos em hashMaps que tem como chave o seu número.

- **Relatório.java**: pega todos os dados da eleição e manipula eles (em cálculos e faz comparações entre as listas dos candidatos)para eles para gerar os relatórios, além de moldar a formatação da impressão;
- **Leitor.java**: uma classe sem atributos, todavia com funções estáticas que fazem a leitura e armazenamento dos dados contidos em candidatos.csv e votação.csv.
- **Main.java:** pega os dados do terminal de código da cidade, nomes de arquivos e data, redireciona para pegar as informações dos arquivos de votação e candidatos, processa eles e gera os seus relatórios.
- **CandidatoComparator.java**: ordena os candidatos com base no seu número de votos válidos e, caso haja empate, em que é mais velho.
- **PartidoComparator.java**: ordena os partidos com base no seu número de votos válidos e, caso haja empate, no seu número de urna.
- **CandidatoPartidoComparator.java:** compara os partidos com base nos seus candidatos mais votados e, caso haja empate, no número de urna do partido.

O diagrama abaixo mostra as relações deles:



Testes

Inicialmente os testes foram realizados de forma manual, com os casos de teste disponibilizados pelo professor, com destaque para os arquivos menores e mais claros de visualizar, analisar e identificar discrepâncias (como os arquivos da pasta AC1473), especialmente a formatação da saída, que deveria seguir exatamente o modelo esperado. Em seguida, após a correção dos erros iniciais, passamos a utilizar o script de correção, que acusou pequenas diferenças, inéditas até então, mas que logo foram encontradas e corrigidas.

Bugs

Para que o programa funcione corretamente, é essencial garantir que os dados de entrada estejam no formato esperado. Os arquivos devem ser **.csv**, utilizando ponto e vírgula (**;**) como separador e mantendo todas as células entre aspas duplas (**" "**).

Outro ponto fundamental é a execução do código com os argumentos corretos e na ordem estabelecida. A sequência esperada é:

<código_municipio> <caminho_arquivo_candidatos> <caminho_arquivo_votacao> <data>

Apenas valores válidos e compatíveis com o formato especificado devem ser passados como parâmetros. Caso contrário, podem ocorrer falhas na execução.

Embora o programa já trate exceções do tipo IOException, reportando a causa do erro na saída padrão, outros problemas podem não ser capturados automaticamente. Por isso, é indispensável que os arquivos não sofram alterações na estrutura original e que não haja dados ausentes ou incompatíveis.

Seguir essas diretrizes evita erros inesperados e garante que o processamento ocorra de maneira eficiente e confiável.