UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

DEPARTAMENTO DE INFORMÁTICA

PROGRAMAÇÃO ORIENTADA A OBJETO

Gabriel Saymon da Conceição

Processamento de dados das eleições de vereadores

Trabalho I - Java

Vitória, ES

2025

Introdução

O projeto consiste no desenvolvimento de relatórios a partir da base de dados das eleições municipais de 2024 para o cargo de vereador. A partir dos arquivos CSV oficiais fornecidos pelo Tribunal Superior Eleitoral, o sistema captura informações de candidatos e de votação, organiza a relação partido–candidato e gera dez relatórios detalhados. A aplicação está dividida em quatro pacotes principais:

- **br.ufes.gabriel.eleicao**: contém a classe App, responsável por orquestrar a execução a partir dos parâmetros de linha de comando (código do município, arquivos CSV, data da eleição e tipo de consulta).
- **br.ufes.gabriel.eleicao.model**: reúne as classes de domínio Candidato e Partido, cada uma com seus atributos (número, nome, data de nascimento, votos, sigla, etc.) e implementações de Comparable para ordenação.
- **br.ufes.gabriel.eleicao.util**: abriga a classe Leitor, que faz a leitura linha a linha dos CSVs via BufferedReader (encoding ISO-8859-1), filtra registros válidos e popula a instância de Eleicao.
- **br.ufes.gabriel.eleicao.service**: concentra Eleicao (estrutura de HashMap para candidatos e List para partidos, além de metadados como data e número de vagas) e Relatorio, onde residem os métodos que processam esses dados e imprimem os dez relatórios na saída padrão.

A classe Relatório reúne toda a lógica de análise: a partir dos dados consolidados em Eleição, ela calcula os eleitos no sistema proporcional, identifica top-N mais votados, casos prejudicados e beneficiados, votação por partido, primeiro e último colocados de cada legenda, faixas etárias, distribuição de gênero e totais de votos.

Implementação

Para estruturar o sistema foram definidas cinco classes principais:

A. *App.java*

Contém o método main(String[] args), que valida os quatro argumentos (código de município — preservando zeros à esquerda —, arquivo de candidatos, arquivo de votos e data da eleição), cria as instâncias de Eleicao, Leitor e Relatorio e dispara, nesta ordem, a leitura dos dados e a geração dos relatórios.

B. *Candidato.java e Partido.java*

– *Candidato* armazena número de urna, nome na urna, nome completo, data de nascimento, gênero, número do partido, sigla do partido, número de federação e total de votos nominais. Implementa Comparable para ordenar por votos (descendente) e, em caso de empate, por idade (mais velho primeiro). Oferece métodos de acesso, addVotos(int) e isEleito().

– *Partido* guarda número, sigla, total de votos de legenda e lista de candidatos.Tem métodos para inserir candidatos, acumular votos de legenda, calcular votos nominais (soma dos candidatos) e total de votos (nominais + legenda), e ordenar sua lista interna de candidatos segundo os mesmos critérios de Candidato.

C. *Eleicao.java*

Responsável por armazenar todo o estado do cálculo: código do município, data da eleição, número de vagas (eleitos), um HashMap<Integer,Candidato> para acesso direto ao candidato pelo número de urna e uma List<Partido> para iteração e ordenação. Expõe métodos de inserção (adicionaCandidato, adicionaPartido), de consulta (getPartidoPorNumero, getNumeroCandidatosEleitos) e getters das coleções.

D. *Relatorio.java*

Reúne as dez funções de relatório, cada uma imprimindo seu bloco de texto formatado em pt-BR e tratando pluralização via if/else (0 ou 1 no singular, >1 no plural). Na ordem:

1. Número de vagas
2. Vereadores eleitos (proporcional, com * para federações)
3. Top-N mais votados nominalmente
4. Prejudicados (top-N não eleitos)
5. Beneficiados (eleitos fora do top-N)
6. Votação por partido (nominais, legenda e total)
7. Primeiro e último de cada partido (empate por idade)
8. Faixa etária dos eleitos
9. Distribuição por gênero
10. Totais de votos válidos, nominais e de legenda

E. *Leitor.java*

– *Leitor* faz a leitura dos dois CSVs com BufferedReader e encoding ISO-8859-1, remove aspas, divide por ponto-e-vírgula, filtra por município (comparando como inteiro), cargo = 13 e situação válida, e insere candidatos e partidos em Eleicao; em seguida processa o arquivo de votação, identifica votos nominais versus legenda e os acumula em Candidato ou Partido.

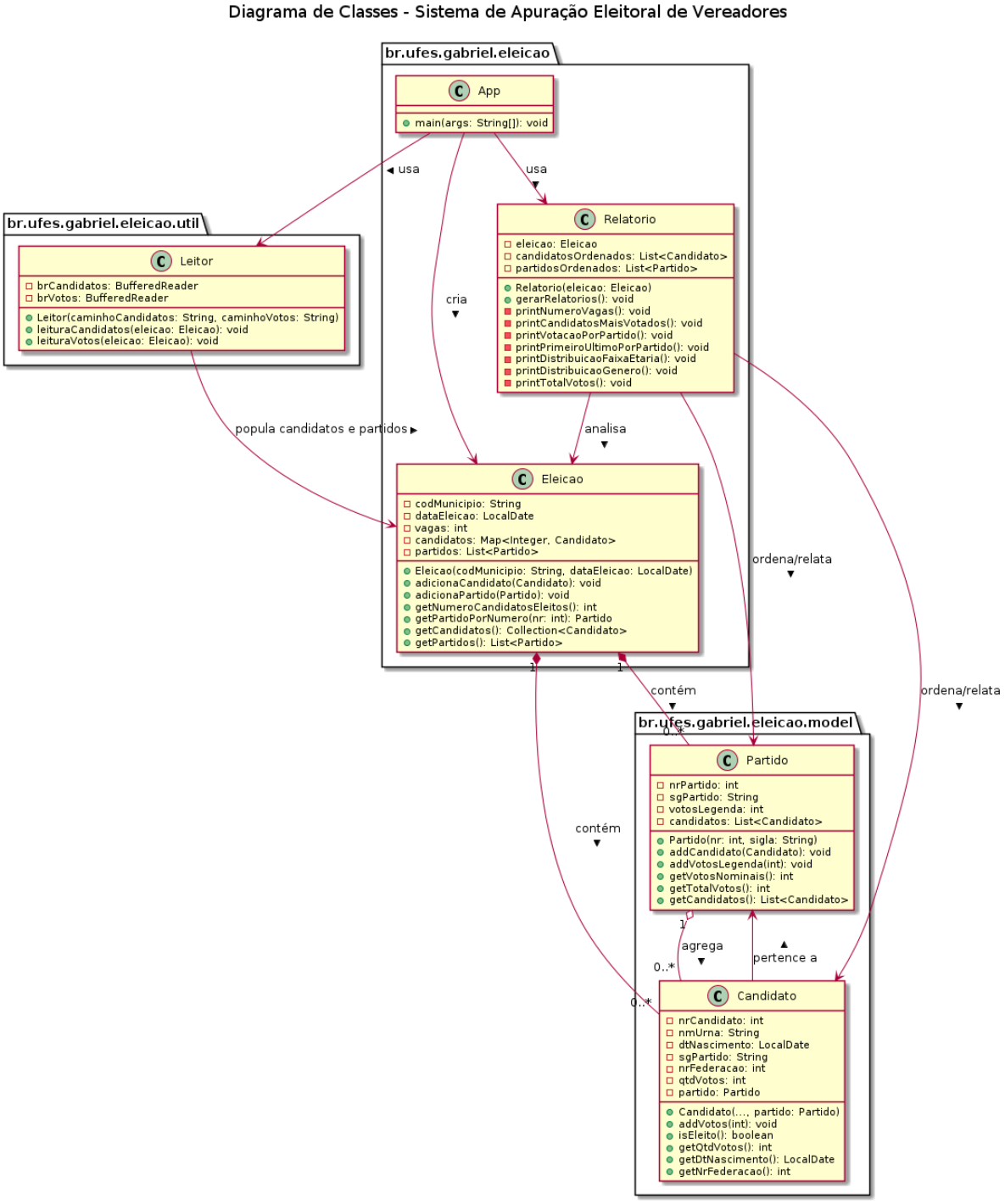

Figura 1 - Diagrama UML da relação de classes do programa.

Descrição dos testes

Durante o desenvolvimento do programa, o código foi testado apenas com os dados do Espírito Santo.

Com o código finalizado, foi utilizado o script de correção disponibilizado pelo professor, no qual o programa é testado para os seguintes estados: AC, ES, PE.

Sendo assim, o script afirmou que as saidas geradas estão de acordo com o esperado, como apresentado na Figura 4.

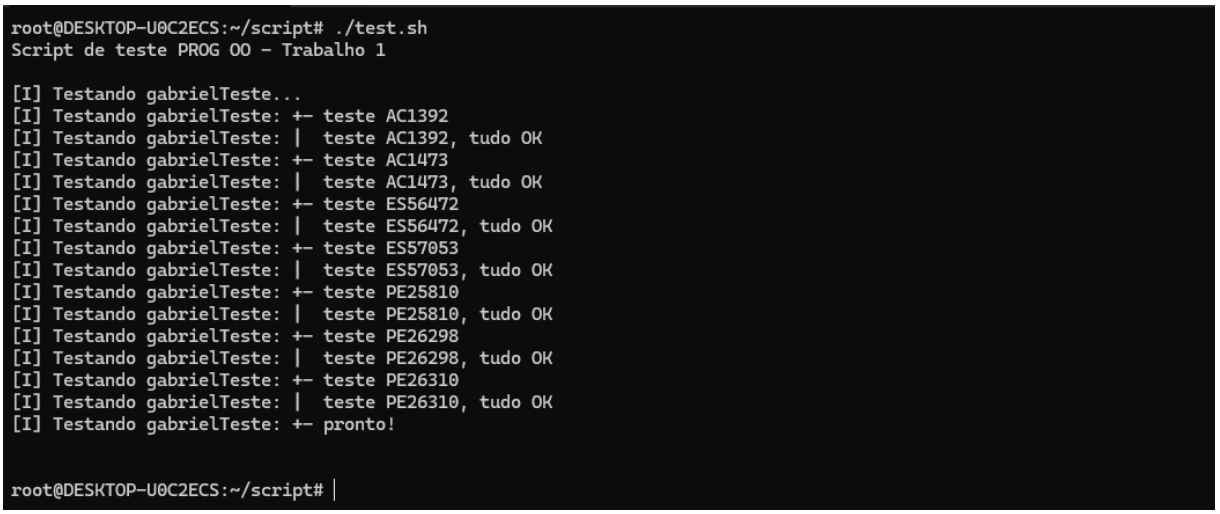

Figura 4 - Testes realizados pelo scrip

Bugs

Ao concluir o projeto nao foi observado nenhuma anormalidade ou inconsistência no que diz respeito ao funcionamento e execução do mesmo.

Conclusão

Em síntese, o desenvolvimento deste sistema reforçou a importância de um planejamento arquitetural sólido antes de iniciar a codificação. A definição clara dos pacotes — leitura, modelo de domínio e geração de relatórios — e das responsabilidades de cada classe permitiu que a implementação evoluísse de maneira coesa e escalável. Além disso, a adoção de estruturas de dados adequadas (HashMap para acesso rápido aos candidatos e listas ordenadas para geração de rankings) e de comparators bem definidos garantiu precisão nos critérios de desempate e na produção dos relatórios. Por fim, a separação de responsabilidades facilitou testes, futuras manutenções e eventuais extensões, assegurando que novas regras de negócio ou tipos de análise possam ser incorporados sem grandes impactos no código existente.