Aline Mendonça Mayerhofer Manhães

Prof. João Paulo A. Almeida

Programação Orientada a Objetos 2024/02

28 de fevereiro de 2025

Primeiro Trabalho de Implementação - Sistema Eleitoral

Criação de um sistema eleitoral, com a leitura de dois arquivos: candidatos e votação. No arquivo de candidatos, foram lidas informações acerca dos partidos e dos candidatos, como seus nomes, números, situação eleitoral, nascimento, além de dados sobre o tipo de cargo e da cidade na qual participaram, entre outras informações. No arquivo de votação, foram lidas as quantidades de votos para um determinado candidato (partido - votos de legenda - ou candidato - votos nominais). A partir disso, foram criados 10 relatórios, conforme informado na especificação do trabalho enviada pelo professor. Estes relatórios detalham dados diversos sobre a eleição, como os candidatos eleitos e os mais votados.

Implementação

Foram criados 9 arquivos .java. São estes 1 arquivo principal (App.java), 5 arquivos de classes diversas (Leitor.java, Candidato.java, Partido.java, Eleicao.java e Relatorio.java), além de 3 arquivos de interface Comparator (ComparadorCandidatos.java, ComparadorPartidos.java e ComparadorPartCand.java).

Em App.java, na main, foram recebidos por args os nomes dos arquivos a serem lidos, o código que representa a cidade e a data da eleição. A partir disso, foi criado um objeto da Classe Leitor, que contém os códigos da cidade e do cargo (vereador) a ser lido nos arquivos, além de um objeto Eleicao. Essa eleição é criada no construtor do Leitor, com a data da eleição.

Uma vez que o Leitor foi criado, podemos ler efetivamente os dois arquivos: candidatos e votação, respectivamente. As leituras são feitas com InputStream, BufferedReader e Scanner e são tratadas IOExceptions, exibindo uma mensagem de erro. Na leitura de candidatos, são armazenadas 10 informações dentre as 50 colunas do arquivo. Para cada partido novo que aparece, independente do cargo, é criado um objeto da Classe Partido e este é guardado dentro de eleição. Além disso, caso as exigências de código de cidade, cargo e situação eleitoral sejam

cumpridas, um novo objeto Candidato é criado e armazenado em eleição. Esse processo é repetido enquanto existirem linhas a serem lidas no arquivo de candidatos.

No arquivo de votação, a dinâmica de leitura é semelhante, mas os dados lidos mudam. São lidos a quantidade de votos e o número do candidato a receber esses votos (pode ser Partido ou Candidato). Caso seja um Partido, os votos são adicionados como votos de legenda. Se for um Candidato, esses votos são adicionados em si próprio e aos votos totais do seu Partido, mas são considerados como votos nominais. Votos nominais = votos totais - votos de legenda;

Feito todo o processo de votação, os relatórios são gerados. Para isso, recebemos na main a Eleicao que o Leitor gerou, com todos os dados atualizados do nosso sistema eleitoral. É criado um objeto da classe Relatório, que recebe Eleicao e cria, a partir disso, listas encadeadas para representar os candidatos, os candidatos eleitos e os partidos da eleição, além de guardar a data e o número de vagas disponíveis para o cargo na cidade em questão. São calculadas todas as estatísticas para gerar os 10 relatórios e exibi-los, conforme a especificação do trabalho, mostrando todos os dados na formatação exata de saída.

Classes

Classe App: contém a função main. Cria e chama o Leitor e o Relatorio. Logo, chama as funções de leitura e geração de relatórios.

Classe Leitor: guarda as informações essenciais para a leitura dos arquivos, além de efetivamente ler os dois arquivos e guardar os dados lidos na Eleicao.

Classe Candidato: armazena as informações de um candidato. Também possui funções para cálculo da idade e para aumentar a quantidade de votos de um candidato. Possui acesso ao seu partido.

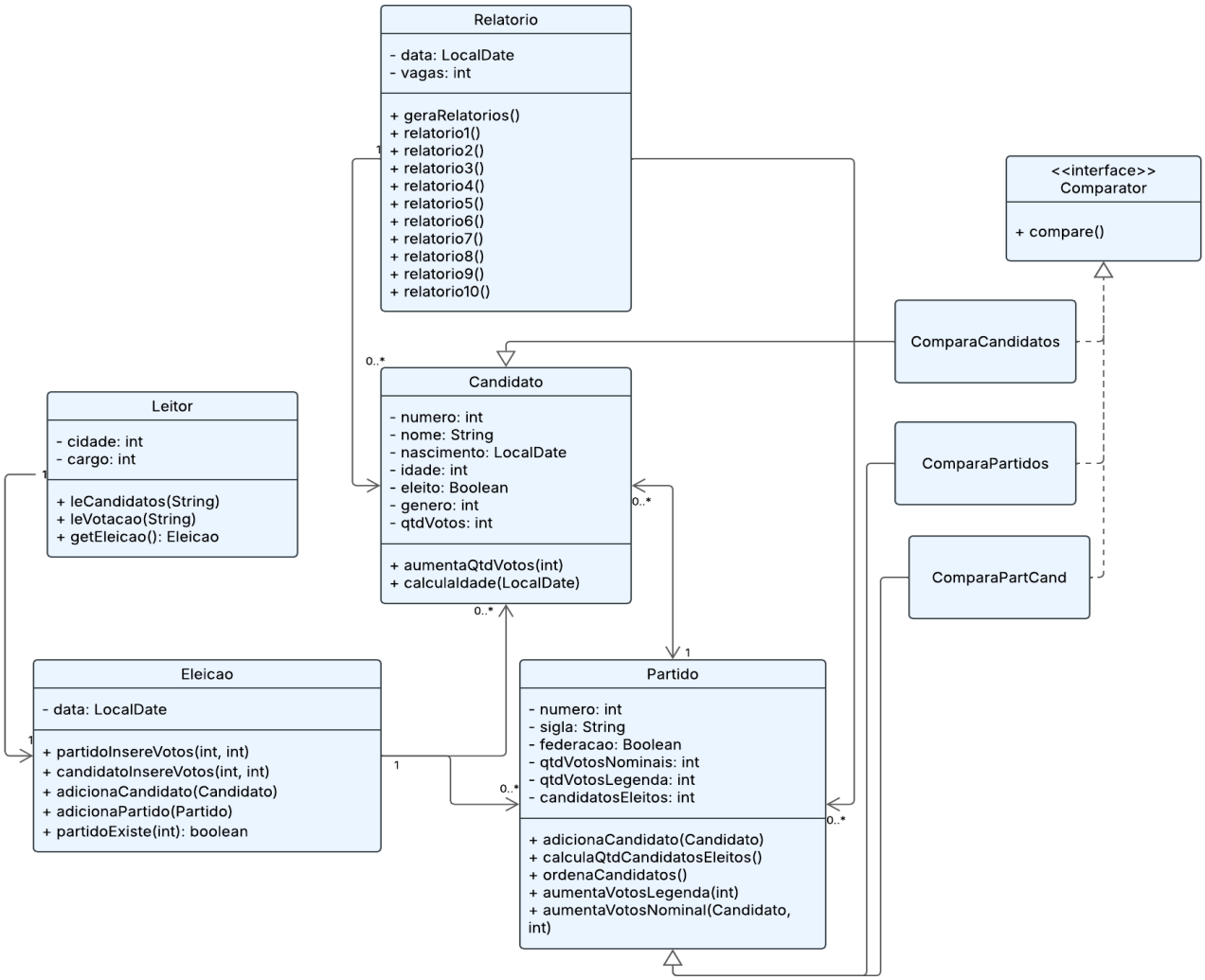
Classe Partido: armazena as informações de um partido. Possui acesso aos seus candidatos, por meio de um HashMap, no qual as chaves são os números dos candidatos. Também possui funções para inserir um candidato em seu HashMap, ordenar seus candidatos e para aumentar a quantidade de votos de um candidato ou do próprio partido.

Classe Eleicao: classe com todos os candidatos e todos os partidos armazenados em HashMaps distintos. Pode conferir se um partido existe, adicionar candidatos ou partidos aos seus HashMaps, contabilizar votos e retornar um candidato específico com base em seu número.

Classe Relatorio: possui listas encadeadas para candidatos, candidatos eleitos e partidos, para calcular todos os dados de relatórios exigidos na especificação. Calcula o número de vagas, os candidatos eleitos, os mais votados, os partidos mais votados, a distribuição dos votos por tipo e de candidatos por idade e gênero.

Classes de Comparator: são 3, cada uma com um propósito.
ComparadorCandidatos ordena os candidatos usando o critério de número de votos válidos, e em último caso, a idade deles.
ComparadorPartidos ordena os partidos a partir de seus votos totais, e em último caso, compara seus números.
ComparadorPartCand ordena os partidos a partir de seus candidatos mais votados (usando ComparadorCandidatos), e em caso de empate, compara os números dos partidos.

A implementação das Classes segue o diagrama abaixo:



Testes

Uma vez que o código estava pronto, para testá-lo, eu peguei 3 candidatos aleatórios em uma planilha de candidatos e os coloquei em um arquivo .csv de teste. Depois, escolhi um deles e criei um arquivo de votação .csv de teste, adicionando 2 votos a esse candidato escolhido. Assim, pude testar a minha leitura de arquivos. Posteriormente, fui alterando algumas variáveis, como a do código da cidade, da situação eleitoral, de federação, entre outras, e fui verificando se a minha saída nos relatórios também era alterada conforme o esperado. Após testar a leitura e os relatórios, eu passei a testar com os casos de testes dados pelo professor (2 do Acre, 2 do Espírito Santo e 3 de Pernambuco). Ajustei alguns erros e cheguei no meu código final.

Bugs

É fundamental que todas as pré-condições definidas na especificação do trabalho sejam cumpridas. São elas:
1- Arquivos do tipo .csv, com valores separados por ponto-vírgula (;) e todas as células com os seus valores entre aspas duplas (“ “). Além disso, o tipo do valor deve ser respeitado. Por exemplo, para ler o número de um candidato, espera-se um número (inteiro) e não uma String.
2- Rodar o código da maneira correta, passando todos os argumentos necessários para o funcionamento do programa. A sequência correta de argumentos é:

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

3- Passar por parâmetro, apenas argumentos válidos e que correspondam ao tipo que é especificado acima. Portanto, seguir a ordem é essencial.

O programa já trata IOException e informa a sua causa na saída padrão. Porém, o mesmo não ocorre para todos os possíveis erros. Logo, é necessário que não existam entradas inválidas, dados faltantes ou de tipo diferente do especificado, alterações na estrutura original dos arquivos de leitura. Isso deve ser seguido a fim de evitar comportamentos inesperados por parte do programa. Assim, as pré-condições devem ser seguidas.