UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CARLOS DANIEL ALBERTINO VIEIRA

DAVID CRISTIAN MOTTA PROPATO

SISTEMA ELEITORAL BRASILEIRO

Vitória

2022

CARLOS DANIEL ALBERTINO VIEIRA
DAVID CRISTIAN MOTTA PROPATO

SISTEMA ELEITORAL BRASILEIRO

Relatório apresentado à disciplina de Programação Orientada a Objetos como requisito parcial para obtenção de nota.
Professor João Paulo Andrade Almeida.
Matrículas: 2020100867 e 2020101300

Vitória
2022

Sumário

1 INTRODUÇÃO

O sistema eleitoral brasileiro apresenta dois modelos de eleições na escolha dos representantes políticos: majoritário e proporcional. Cargos como presidente, governador, senador e prefeito são escolhidos sob o regime da eleição majoritária, na qual os candidatos mais votados são eleitos, enquanto os demais cargos são decididos pelo modelo da eleição proporcional, onde a eleição dos candidatos depende também do número de votos em seus partidos.

Todavia, o funcionamento do sistema eleitoral brasileiro continua uma incógnita para muitos cidadãos, que em geral desconhecem sua operação. Essa desinformação serve de obstáculo no entendimento tanto do meio como se tece o cenário político atual, como da cidadania de modo geral.

Em vista disso, desmistificar o funcionamento das eleições se torna uma tarefa essencial na efetivação da cidadania. Nessa ótica, ferramentas que disponibilizam informações capazes de diminuir o nível de desentendimento sobre as eleições ganham relevância, o que abre espaço para contribuições provenientes da programação orientada a objetos no fomento de sistemas de análise e exposição de dados sobre tal temática.

1.1 OBJETIVOS

Implementar um sistema em Java capaz de processar dados obtidos da Justiça Eleitoral referentes à votação de vereadores, de forma a levantar informações e gerar relatórios sobre as eleições de 2020 em um município, a fim de aplicar e desenvolver os conhecimentos adquiridos em aula.

2 IMPLEMENTAÇÃO

Para a saída dos dados pedidos na saída padrão, optou-se por usar PrintStream em vez do simples System.out, preferência cunhada no desenvolvimento inicial do código, quando os métodos geravam relatórios em um arquivo de texto separado (por um equívoco na interpretação das instruções do trabalho).

Além disso, o locale pt Br foi inicializado como default no início do programa, a fim de padronizar a leitura e escrita dos dados em diferentes máquinas segundo a convenção brasileira. Da mesma forma, ao longo do programa são utilizadas formatações típicas para decimais no Brasil.

Ainda, foram criados packages para candidatos (com as classes Candidato e DadosCandidatos) e partidos (com as classes Partido e DadosPartidos), com intuito de tornar mais organizado, lógico e prático a relação entre certas classes.

2.1 CLASSES PRINCIPAIS

O programa contou com a criação de classes específicas para a realização das tarefas propostas, nas quais foram empregadas tipos nativos e de referência que apresentam métodos úteis no trato da informação armazenada. Assim, ao todo foram criadas 7 classes: Leitura, Candidato, Partido, DadosCandidato, DadosPartido, comparadorVotosNominais e comparadorVotosLegenda.

2.1.1 Leitura

Essa classe é responsável pela leitura dos arquivos de candidatos e vereadores.

Os métodos LerCandidatos() e LerPartidos() leem todas as linhas do arquivo de entrada e criam uma variável de Candidato e Partido para cada uma dessas através dos métodos LerCandidato() e LerPartido(), armazenando esses em ArrayLists de Candidatos e de Partidos.

2.1.2 Candidato

Foi criada com o intuito de reunir as informações relacionadas a um candidato (disponíveis no arquivo de leitura) e também métodos específicos relacionados à impressão na saída padrão e comparação de objetos desse tipo.

O ponto de destaque para as variáveis dessa classe está na escolha do tipo LocalDate para armazenar a data de nascimento dos candidatos; consiste num tipo próprio para o armazenamento e tratamento de datas, de forma a oferecer um método de comparação que posteriormente seria útil na ordenação da lista de candidatos segundo o critério de prioridade para o mais velho/novo. Além disso, também foi adicionado uma variável da classe Partido, a fim de relacionar cada candidato diretamente com seu partido e facilitar a construção de métodos necessários para a geração de alguns relatórios.

Sobre os métodos implementados, foram criados, ainda, getters e setters de acordo com a necessidade do projeto. Desse modo, esses não foram definidos para todas variáveis, a exemplo do único setter construído para o partido do candidato, visto que esse campo, diferentemente dos demais, apenas poderia ser preenchido após lido o arquivo de entrada dos partidos.

Por fim, nessa classe foi implementado um comparable para ser utilizado posteriormente na ordenação da lista de candidatos segundo critérios específicos. A escolha por um comparable em vez de um comparator residiu no fato de se tratar de uma forma relativamente mais simples de implementar e que atingiria o mesmo objetivo.

2.1.3 Partido

Foi criada com o intuito de reunir as informações relacionadas a um partido (disponíveis no arquivo de leitura) e também métodos específicos relacionados aos objetos desse tipo.

Para as variáveis, o destaque foi na criação de uma variável do tipo ArrayList<Candidato> para armazenar a lista de candidatos de um partido. A escolha pela utilização de uma lista implementada sobre Array se deu em razão dessa

classe oferecer métodos de acesso e setagem relativamente mais rápidos do que as demais opções, o que seria vantajoso em diversos métodos, onde seriam realizadas buscas, iterações, comparações e outros procedimentos.

Sobre os métodos implementados, foram criados getters e setters de acordo com a necessidade no projeto. Desse modo, esses não foram definidos para todas variáveis, a exemplo do único setter construído para a adição de um elemento na lista de candidatos, visto que essa variável, diferentemente das demais, apenas poderia ser preenchida após a leitura do arquivo de entrada dos candidatos. Além disso, também foram implementados métodos que poderiam ser úteis na contabilização de votos e candidatos.

Por fim, nessa classe foram implementados dois comparators (como classes estáticas), para que pudessem ser utilizados posteriormente na ordenação da lista de partidos segundo critérios específicos.

2.1.4 DadosCandidatos

Foi criada com o intuito de analisar e processar as informações relacionadas a uma lista implementada em Array de Candidato.

Essa classe se destaca através do método AssociarPartidosCandidatos(), que, ao receber listas de candidatos e partidos, conecta cada partido com seus devidos candidatos por meio do número do partido, variável do tipo int comum às classes (Candidato e Partido).

Os métodos implementados nessa classe focam nos relatórios relativos aos candidatos.

2.1.5 DadosPartidos

Foi criada com o intuito de analisar e processar as informações relacionadas a uma lista implementada em Array de Partido.

Os métodos implementados nessa classe focam nos relatórios relativos aos partidos.

2.2 RELACIONAMENTO ENTRE CLASSES

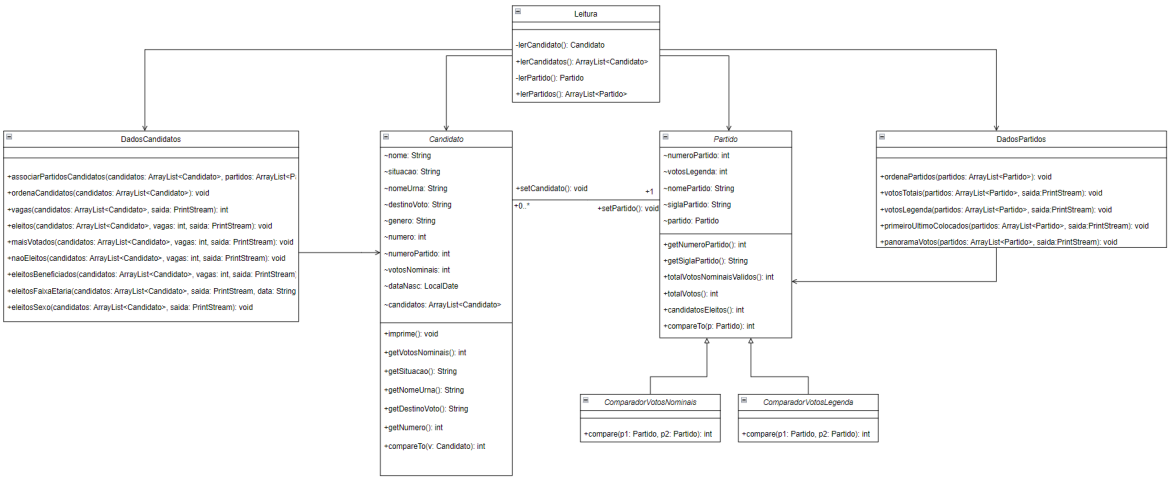

Figura 1 - Diagrama UML de classes do programa. Link para melhor visualização embutido na imagem.

2.3 TRATAMENTO DE EXCEÇÕES

Por meio da documentação da Oracle, que apresenta descrições das classes próprias do Java, foi possível identificar e compreender quais seriam possíveis exceções que o código geraria para cada método chamado.

2.3.1 NumberFormatException

Exceção gerada ao falhar na conversão de uma String para uma formatação válida do tipo NumberFormat ou de um tipo nativo da linguagem. Tal exceção poderia ser gerada pelo programa ao tentar converter strings numéricas lidas do arquivo de entrada para o tipo inteiro. Como a tarefa de leitura ocorre em tempo de execução e é rotineira, não há como ignorar o tratamento dessa exceção.

2.3.2 DateTimeParseException

Exceção gerada ao falhar na conversão de uma String para uma formatação válida do tipo LocalDate. Tal exceção poderia ser gerada pelo programa ao tentar realizar o parsing da data fornecida na linha de comando para o tipo LocalDate. Por

se tratar de um argumento indispensável para o funcionamento do programa e que é passível de erros de digitação por parte do cliente, é necessário cobrir a exceção gerada nesse caso .

2.3.3 FileNotFoundException

Exceção gerada ao falhar na abertura de um arquivo inexistente. Tal exceção ocorreria naturalmente no programa caso o arquivo csv não existisse no caminho fornecido na linha de comando.

2.3.4 SecurityException

Exceção gerada caso ocorram erros na abertura de um arquivo relacionados a permissões de acesso. Embora as chances sejam baixas, ainda poderia ocorrer a depender da forma como os arquivos de leitura foram modificados pelo cliente, portanto é interessante tratar esse caso e identificá-lo.

2.3.5 NoSuchElementException

No caso do programa em questão, surgiria decorrente da estratégia de leitura dos arquivos de entrada, na qual lê-se linha a linha desses e são extraídos os campos específicos dos candidatos e partidos por meio de tokenização. Sob uma possível falha na obtenção de um desses campos/tokens, há a possibilidade de não haver elemento lido, o que desencadearia essa exceção.

2.3.6 IOException

Exceção tipicamente ligada a erros na leitura ou escrita de arquivos de entrada ou saída. Como no trabalho são manipulados inputs e outputs na linha de comando, urge a necessidade de tratar tal exceção, o que é normalmente mandatório no manejo de entradas e saídas em Java e outras linguagens.

3 TESTES

O programa foi submetido a uma bateria de testes disponibilizada pelo professor João Paulo Andrade, a qual contém 7 casos de testes ao todo. As cidades que compõem a bateria de testes são: Belo Horizonte, Cariacica, Rio de Janeiro, São Paulo, Serra, Vila Velha e Vitória.

Além de apresentar os inputs e outputs relativos a cada uma dessas cidades, a bateria de testes continha também scripts que automatizam o processo de teste, de forma a rodar sequencialmente os testes para cada uma das cidades e apontar erros entre as saídas construídas e as esperadas do programa.

Assim sendo, o script de teste fornecido apontou congruência em todos os casos de teste, sem quaisquer diferenças até onde se notou.

```
carlos@carlos-Inspiron-5420:~/Projetos/java/trab1/Trabalho-1-Prog3/script$ ./test.sh
Script de teste PROG3 - Trabalho 1

[I] Testando trabalho...
[I] Testando trabalho: teste belo-horizonte
[I] Testando trabalho: teste belo-horizonte, tudo OK em output.txt
[I] Testando trabalho: teste cariacica
[I] Testando trabalho: teste cariacica, tudo OK em output.txt
[I] Testando trabalho: teste rio-de-janeiro
[I] Testando trabalho: teste rio-de-janeiro, tudo OK em output.txt
[I] Testando trabalho: teste são-paulo
[I] Testando trabalho: teste são-paulo, tudo OK em output.txt
[I] Testando trabalho: teste serra
[I] Testando trabalho: teste serra, tudo OK em output.txt
[I] Testando trabalho: teste vila-velha
[I] Testando trabalho: teste vila-velha, tudo OK em output.txt
[I] Testando trabalho: teste vitoria
[I] Testando trabalho: teste vitoria, tudo OK em output.txt
[I] Testando trabalho: pronto!
```

Figura 2 - Resultados dos casos de teste fornecidos pelo professor.

4 BUGS

Apesar do tratamento de exceções implementado em código, notou-se que caso uma exceção seja capturada durante a execução do programa, o repasse dessa entre os contextos do código pode não encerrar a execução do programa devidamente, podendo estender o tempo de execução e aumentar o risco da ocorrência de bugs. Assim, a função em questão retornará ao fluxo do código e as demais serão executadas corretamente, desde que não apresentem erro similar ou sejam afetadas pelo erro da função em que a exceção foi captada; por exemplo, caso ocorra uma exceção nas funções de leitura, não serão gerados os vetores com os candidatos e vereadores, sendo assim as funções seguintes apresentarão erros em sua execução.

5 DISCUSSÕES

A dupla acredita ter entendido e aplicado bem seus conhecimentos adquiridos ao longo do período, não tendo encontrado grande dificuldade na execução deste trabalho. Percebemos que ao longo do desenvolvimento do trabalho entendemos mais e melhoramos nossas habilidades na programação em Java, trabalhando também no Git e GitHub, estudando e usando eles na colaboração dos códigos desenvolvidos.

6 CONCLUSÕES

Em conclusão, é possível apontar que o programa apresentado se exibe capaz de processar com êxito dados obtidos da Justiça Eleitoral referentes à votação de vereadores, de forma a levantar informações e gerar relatórios sobre as eleições de 2020 em todos os municípios testados.

A implementação em Java do sistema em questão permitiu entender o potencial dessa linguagem no tratamento de dados, de forma a confirmar a utilização da Programação Orientada a Objetos na resolução de problemas reais do mundo contemporâneo.

REFERÊNCIAS

WHICH implementation of list to use. Stackoverflow, 18 jan. 2022. Disponível em: https://stackoverflow.com/questions/3830694/which-implementation-of-list-to-use. Acesso em: 19 jan. 2022.

BLOCOS try catch. Devmedia. Disponível em: https://www.devmedia.com.br/blocos-try-catch/7339. Acesso em: 19 jan. 2022.

JAVA ArrayList. W3schools. Disponível em: https://www.w3schools.com/java/java_arraylist.asp. Acesso em: 19 jan. 2022.

PASCAL Case e camel Case – O que é e como usar? Celsokitamura, 22 set. 2017. Disponível em: https://celsokitamura.com.br/pascal-case-e-camel-case/. Acesso em: 23 jan. 2022.

O QUE é um diagrama de classe UML?. LucidChart, [2018?]. Disponível em: https://www.lucidchart.com/pages/pt/o-que-e-diagrama-de-classe-uml. Acesso em: 4 fev. 2022.

USEDELIMITER. Oracle, [2017?]. Disponível em: https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html#useDelimiter-java.lang.String-. Acesso em: 19 jan. 2022.

PARSEINT. Oracle, [2017?]. Disponível em: https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html#parseInt-java.lang.String-. Acesso em: 20 jan. 2022.

FILEINPUTSTREAM. Oracle, [2017?]. Disponível em: https://docs.oracle.com/javase/8/docs/api/java/io/FileInputStream.html#FileInputStream-java.io.File-. Acesso em: 22 jan. 2022.

INPUTSTREAMREADER. Oracle, [2017?]. Disponível em: https://docs.oracle.com/javase/8/docs/api/java/io/InputStreamReader.html. Acesso em: 22 jan. 2022.

READLINE. Oracle, [2017?]. Disponível em: https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html#readLine–. Acesso em: 22 jan. 2022.

LOCALDATE. Oracle, [2017?]. Disponível em: https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html. Acesso em: 21 jan. 2022.

DATEFORMAT. Oracle, [2017?]. Disponível em: https://docs.oracle.com/javase/7/docs/api/java/text/DateFormat.html. Acesso em: 21 jan. 2022.

DATETIMEFORMAT. Oracle, [2017?]. Disponível em: https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html. Acesso em: 26 jan. 2022.

NUMBERFORMAT. Oracle, [2017?]. Disponível em: https://docs.oracle.com/javase/7/docs/api/java/text/NumberFormat.html. Acesso em: 27 jan. 2022.

COMPARATOR. Oracle, [2017?]. Disponível em: https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html. Acesso em: 26 jan. 2022.

LOCALE. Oracle, [2017?]. Disponível em: https://docs.oracle.com/javase/8/docs/api/java/util/Locale.html. Acesso em: 26 jan. 2022.