UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

GABRIEL FERRARI CIPRIANO

KELVIN STEINER SANTOS

TRABALHO — VEREADORES — JAVA
RELATÓRIO

PROGRAMAÇÃO III — 2018/2

VITÓRIA

2018

1. INTRODUÇÃO

Este relatório descreve o desenvolvimento do código produzido para o trabalho de Programação 3 referente à implementação de um programa em Java capaz de processar dados de uma eleição para vereador e emitir um relatório com uma série de informações sobre o resultado da eleição

O programa tem como entrada um arquivo CSV com uma listagem de candidatos com dados como nome, número de votos, partido, coligação, percentagem de votos válidos, se o candidato foi eleito ou se teve seus votos invalidados. A partir desses dados é produzido um relatório com a listagem de vereadores eleitos, os mais votados, os que foram beneficiados pelo sistema proporcional, os que foram prejudicados, e o número total de votos e candidatos eleitos de cada partido e de cada coligação.

O problema foi modelado utilizando Orientação a Objetos e implementado através de alguns *design patterns* e funcionalidades da linguagem Java que serão mostrados no decorrer da descrição da implementação que será feita na próxima seção.

1.1. Entrada

Os arquivos de entrada são os resultados da votação nominal de vereadores de determinadas cidades nas eleições de 2016, no formato .csv *(Comma-separated values),* isso é, os valores são delimitados por Ponto e vírgula, assumindo uma estrutura de colunas. O arquivo é composto pelas colunas: “Seq”, um número identificador que pode ser precedido por asterisco (‘*’) quando o candidato foi eleito, e por cerquilha (‘#’) se o candidato não teve seus votos validados devido à sua situação jurídica; “Num”, que é o número do candidato; “Candidato”, que é o nome do candidato; “Partido/Coligacao”; “Votação”, que é a quantidade de votos de um candidato; e “% validos”, que é o percentual dos votos válidos que o candidato possui.

Uma coligação é uma reunião temporária de partidos políticos para disputar uma eleição. Funciona, a grosso modo, como se fosse um só partido. No nosso arquivo de entrada, na coluna “Partido/Coligacao” é informado o partido do candidato, e caso tenha uma coligação, é seguido de “ - ” mais os partidos da coligação, separados por “/ “.

No sistema proporcional, diferentemente do sistema majoritário os votos são contabilizados primeiramente para as coligações e partidos únicos, e a partir do cálculo do Quociente Eleitoral (total de votos válidos / Qtd vagas), apenas partidos isolados ou coligações que possuam um saldo de votos acima do QE terão direito a alguma vaga. Para os partidos isolados ou coligações que possuem direito à vagas, é calculado o Quociente Partidário (total de votos do partido / QE), que informa a quantidade de vagas que o partido/coligação tem direito. Uma vez que a quantidade de vagas que um partido/coligação foi definida, é selecionado os vereadores do partido/coligação de acordo com sua quantidade total de votos. Caso ainda sobrem vagas após o cálculo do QP de todos partidos únicos e coligações, divide-se o número de votos do partido/coligação pelo número de vagas mais um. Quem alcançar o maior resultado assume a vaga restante.

2. IMPLEMENTAÇÃO

Logo de início ficou clara a necessidade da implementação de uma classe `Candidato` para representar um candidato na eleição, capaz de armazenar seus dados correspondentes como nome e número de votos. Essa classe é bastante simples e é constituída de alguns campos para armazenar os dados dos candidatos, assim como os respectivos getters e setters:

```
private String nome;
private int pos;
private int numVotos;
private float votosValidos;
private Situacao situacao;
```

O tipo `Situacao` nada mais é que uma enumeração[1] que permite os três valores `ELEITO`, `INVALIDO`, `VALIDO`, para indicar, respectivamente: se o candidato foi eleito, se teve os votos invalidados, ou se tem os votos válidos porém não foi eleito.

Partiu-se então para a elaboração das classes `Partido` e `Coligacao`, e percebeu-se que ambas deveriam apresentar um comportamento em comum: a capacidade de armazenar uma coleção de candidatos. Para isso, foi implementada uma classe abstrata[2] `GrupoCandidatos` com as operações de adicionar um `Candidato` e retornar a coleção de candidatos armazenadas, assim como operações para computar o total de votos e de candidatos eleitos.

Às classes concretas `Partido` e `Coligacao` restaria então implementar apenas a relação elas, isto é, um partido está em uma coligação, e uma coligação é um conjunto de partidos. O que nos dá o seguinte diagrama:

[1] https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html
[2] https://docs.oracle.com/javase/tutorial/java/landl/abstract.html

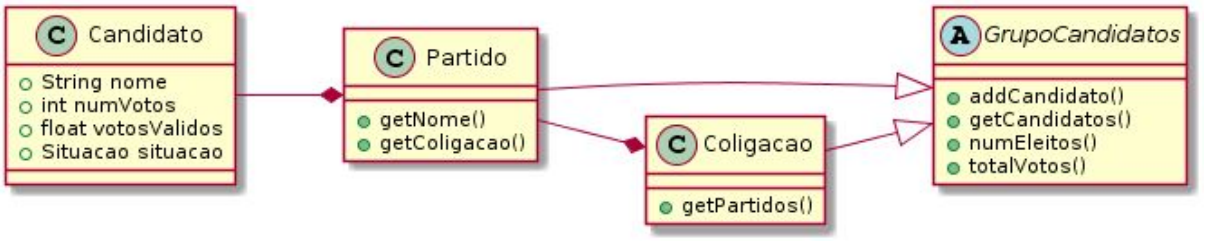

Figura — Diagrama de classes para `Candidato`, `Partido` e `Coligacao`

No caso da classe `Partido`, o nome é utilizado para identificar um objeto, e portanto não pode ser modificado. Da mesma maneira acontece para os objetos de `Coligacao`, que serão identificados pelo conjunto ordenado de nomes dos partidos, portando as suas coleções de partidos não devem ser modificadas.

Para gerenciar os objetos criados e manter corretamente a associação entre eles, foi adotado um *factory pattern* em uma nova classe `Eleicao`. Ela tem a responsabilidade de criar e indexar os partidos e coligações por nome, garantir que todos os partidos tenham referência para a `Coligacao` que os contém quando esta é criada e que os partidos não estejam em mais de uma coligação ao mesmo tempo. Isso é feito pelas operações `getOrCreatePartido`, `getOrCreateColigacao`, que buscam um partido ou coligação nos conjuntos de objetos já existentes através dos identificadores (nomes de partido) ou cria e registra automaticamente os objetos caso não existam.

A classe `Eleicao` também estende `GrupoCandidatos`, já que também armazena a coleção de todos os candidatos participantes na eleição, e expõe a operação `createCandidato`, que cria um candidato a partir dos seus dados o registra adequadamente nos respectivos `Partido`, `Coligacao` e na própria `Eleicao`.

A classe `Main` é a classe principal que contém o ponto de entrada do programa no método estático `main`. Essa classe é responsável por processar os parâmetros de entrada, i.e. o nome do arquivo CSV e codificação, e imprimir os relatórios uma vez que esteja contruída a `Eleicao` com todos os dados. Essa construção da `Eleicao` por sua vez é feita pela classe `Leitor`, que toma uma instância de um `Reader` para o arquivo de entrada, e processa linha por linha fazendo as chamadas adequadas ao objeto `Eleicao`.

Vale notar a utilização de um *decorator pattern* na parte de produção dos relatórios. Nessa parte, é necessária a ordenação de partidos e coligações pelo número de total de votos. Como esse número é obtido varrendo todos os candidatos contidos nesses conjuntos, isso faria a ordenação e impressão desnecessariamente custosas caso utilizassem o cálculo de total de votos diretamente, visto que essa operação seria repetida muitas vezes. Por isso foi criada uma classe chamada `GrupoCandidatos.Contado` capaz de embalar uma instância de Partido ou Coligação, armazenando a quantidade de votos, e também implementando a interface *`Comparable`*[3] para permitir a ordenação baseada no valor armazenado.

Assim, é criada para cada um dos objetos `Coligacao` e `Partido` uma instância de `Contado`, e esses novos objetos que são de fato ordenados e iterados para imprimir as listas de partidos e coligações no relatório.

[3] https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

3. CONCLUSÃO

Para verificar o bom funcionamento do código, foram realizados testes do sistema como um todo, através do script de teste automático fornecido pelo professor, e o programa deu os resultados esperados, com o desempenho esperado, para as entradas referentes às eleições nas cidades do Rio de Janeiro, Vila Velha e Vitória. Um adendo sobre a saída

Através do desenvolvimento do trabalho pudemos exercitar os conhecimentos adquiridos acerca do paradigma de Orientação a Objetos, e da linguagem Java. Foi possível perceber a utilidade de alguns *design patterns*, como o *factory pattern*, sendo interessante notar como alguns desses usos surgiram naturalmente como soluções no decorrer da elaboração do programa.

4. REFERÊNCIAS

Material de aula de Programação III do professor J.P. Almeida. Disponível em: https://nemo.inf.ufes.br/jpalmeida/ensino/2018-02-prog-iii/

The Java™ Tutorials. Disponível em: https://docs.oracle.com/javase/8/docs/api/

Java™ Platform, Standard Edition 8 API Specification. Disponível em: https://docs.oracle.com/javase/8/docs/api/

Eleição majoritária e proporcional. Disponível em: http://www.brasil.gov.br/governo/2012/05/eleicao-majoritaria-e-proporcional

Como funciona o Sistema Proporcional. Disponível em: https://www.eleicoes2018.com/como-funciona-o-sistema-proporcional/