Relatório de implementação:
Trabalho I de programação III

1. Implementação:

Ao me deparar com o problema apresentado, minha primeira percepção foi de estruturar a base do problema, independentemente da solução que ela viria a ter. Assim, comecei com a elaboração do pacote “informacao.pessoal” que poderia, em um grande projeto, ser uma classe já existente e que seria reaproveitada pelo meu programa solução. Assim, esse pacote traz a classe pessoa, que contém informações de uma pessoa física pertinentes para o programa em questão, como ela não foi reaproveitada de outro projeto, apenas continha os atributos e funções necessárias para o trabalho em questão. Um exemplo de integração, que fiz questão de deixar visível nesse projeto, nesse pacote, foi a elaboração de dois construtores. Um deles que recebe todos os atributos diretamente e o que eu utilizaria no meu projeto, que recebe ao invés da idade, a data que é base para o cálculo da mesma. E, como função utilitária, no pacote "utilitário", estruturei a classe data, como o cálculo da idade era simples, optei por essa forma de organização, já que só usaria essa função e a relação de associação nas demais classes do objeto “Data”. Nessas, houve sobrescrita do método toString apenas para testes, no início da implementação.
Agora sim, mais no cerne do problema, comecei a estruturar as peculiaridades, partindo primeiro da análise do arquivo de entrada para estruturar os atributos de dois dos personagens principais do cenário: os candidatos e os partidos.

- Candidatos:

  Começando com os candidatos, que foram os primeiros a serem recebidos pela linha de comando, achei mais prudente colocar a leitura do seu arquivo em sua própria classe, apesar de ser um recebimento formatado particular desse programa. Essa mesma classe poderia ser facilmente reutilizada desconsiderando esse método de leitura mais particular, ou até mesmo, em uma linha de produção de sistema, fornecer em uma documentação mais aprofundada que o programa anterior de cadastro de candidatos formatasse a lista conforme o padrão csv referenciado na especificação do trabalho, isso facilitaria a integração dos projetos na realidade.
  Partindo desse recebimento formatado, a classe faz o uso de construtores adequados e retorna uma lista de candidatos, com as informações pessoais

completas e apenas faltando uma referência para seu partido, que ainda não foi recebido. O método que faz toda essa entrada de dados é o “recebeCandidatos”, que tem como parâmetros o nome do arquivo e a data da eleição para o cálculo da idade dos candidatos.

Uma característica importante da classe é que ela estende a classe pessoa, essa é uma forma de herança que facilita o reuso de código, deixando a disposição das informações mais lógica e sendo possível pelo fato do candidato ser uma pessoa, sendo assim “é um” confirma que a relação é de herança e que o candidato além dos atributos e métodos de sua classe também possui os atributos da classe pessoa.

Nessa classe, houve sobrescrita do método “toString” e “compareTo”. O primeiro otimizou a impressão do candidato, já que muitos relatórios usavam a mesma formatação de impressão do candidato, e o segundo estabeleceu a ordem natural de ordenação por quantidade de votos em ordem decrescente, com o critério de desempate para o que tiver maior idade, que também era o método comum aos relatórios.

- Partidos:

  De forma semelhante e com as mesmas justificativas, o “recebePartidos” foi implementado e colocado nessa classe. Ele recebe o nome do arquivo csv com os dados dos partidos e devolve uma lista com eles alocados. Em um primeiro momento, o construtor recebe apenas os atributos do csv, como nome, número, voto de legenda e sigla, e depois, em um segundo momento, completará os demais atributos, que são sua lista de candidatos, quantidade de eleitos e a quantidade de votos nominais que os candidatos filiados receberam no total. Optei por adicionar tais atributos, pois estão intimamente ligados ao partido e ajudam a melhorar a eficiência do algoritmo quando for necessário exibir os relatórios.

  Nessa classe, houve sobrescrita do método “toString” e “compareTo”. Assim, em relação ao último, a ordem natural de ordenação será por quantidade de votos totais no partido em ordem decrescente, votos nominais somados aos votos de legenda, com o critério de desempate para o que tiver menor número partidário.

Nessas duas classes anteriores houve a conferência de exceções provocadas por erro na abertura de arquivos e/ou de entrada e saída e, na classe principal, que contém o método main, verificou-se a passagem da quantidade correta de argumentos necessários pela especificação.

A lista foi escolhida para todas as formas de armazenamento das coleções necessárias, pois há muita demanda por ordenar esses dados armazenados em conjunto, e ela promove essa ordenação com mais facilidade por conta da API Java.

- Eleições:

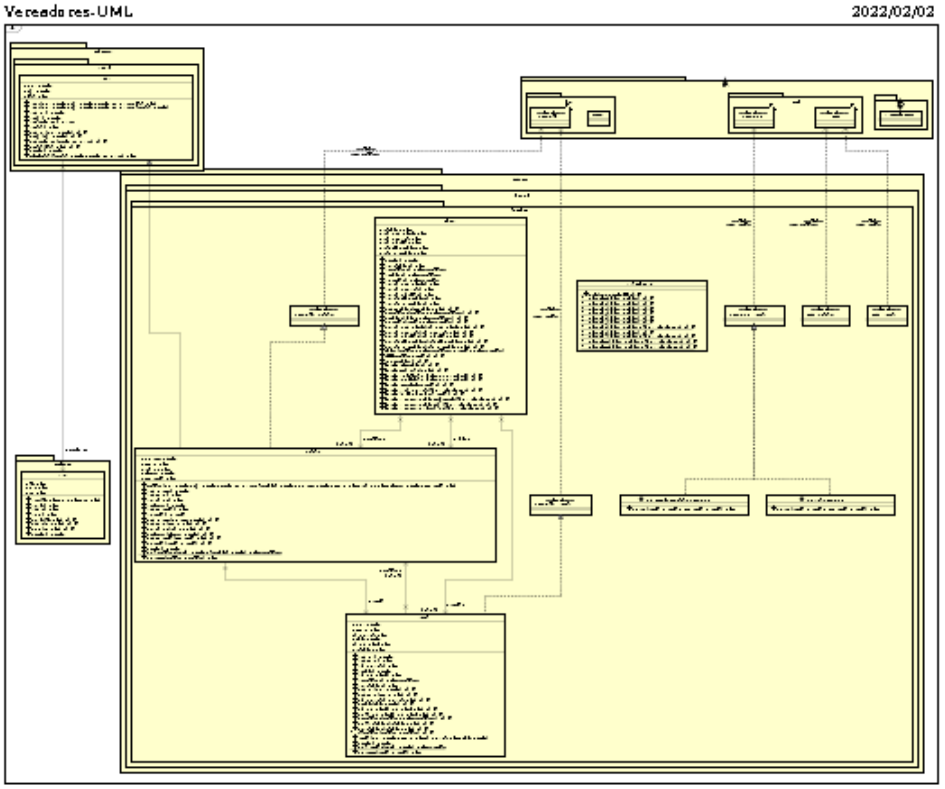
  E, por fim, já que o tema do problema são as eleições, a classe para tal era super importante, visto que uniria partidos e candidatos, teria um resultado, ou seja, uma lista de eleitos, estatísticas sobre a eleição com suas características macros, e por isso era necessária. A elaboração dos relatórios dependiam da eleição ocorrer, e, por mais que, as classes candidatos e partidos contassem com algumas características fruto do resultado como, quantidade de votos, elas têm sua arquitetura mais voltada para criar ou excluir um partido/candidato, mudar o partido de um candidato ou seu nome na urna e afins, logo era preciso e necessário criar um objeto eleições para dar forma ao nosso problema.
  A classe eleições contém atributos do resultado de uma eleição, pois é o foco deste trabalho, mas também poderia conter outras informações como data, quantidade de urnas funcionando, quantidade de mesários, enfim. Nesse caso construímos o objeto “eleições2020” com apenas a lista de candidatos e partidos. E, a partir daí foi preciso desenvolver métodos para:

  1- Preparar a eleição: como a lista de candidatos e partidos foi agregada no nosso objeto a partir do construtor, por ser indispensável para constituir uma eleição, nessa etapa apenas formalizei as filiações por meio do método “afiliaCandidatosPartidos”. Esse conecta e preenche a relação entre os dois protagonistas do processo eleitoral, adicionando o candidato à lista de candidatos do partido dele e estabelecendo referência ao seu partido nos dados do candidato. Vale ressaltar que essa última relação é uma relação de associação, do tipo “tem um”, o candidato tem um partido.

  2- Obter resultado da eleição: consiste em apurar dados mais imediatos sobre o evento, nesse caso, para aumentar a eficiência do programa, computa-se a quantidade de mulheres e homens eleitos, o total de votos válidos, de legenda somado aos nominais e, principalmente, a lista de eleitos e quantidade de vagas. Além de agregar nas informações estatísticas do partido, como a quantidade de votos totais que todos os seus candidatos receberam e a quantidade de eleitos por partido.

Com isso o código já ficou bem modularizado, permitindo por exemplo que mais de uma eleição fosse instanciada, o que seria muito útil se tivesse a pretensão de comparar com eleições passadas. Os objetos também estão bem definidos, permitindo a identificação rápida dos atuantes do programa, cabendo incluir eleitores, mantendo a herança com a classe pessoa, entre outras aplicações. Vale ressaltar que os relatórios, objetivo final dessa solução, foram feitos usando de métodos estáticos apenas para organização da função principal, destarte não fere os princípios de orientação a objetos e poderiam ter sido facilmente implementados na main, o que não foi o caso por uma questão apenas estética. Daí em diante são poucos os comentários a serem feitos, as funções estáticas apenas faziam a formatação da saída e invocavam, através do objeto instanciado “eleições2020” os métodos responsáveis por fazer o “cálculo do relatório”, manipulações dos dados eleitorais. Estes foram feitos por métodos da classe eleição, já que se tratava da quantidade de eleitos, os mais bem e mal colocados em cada partido, os candidatos mais votados, os partidos mais votados, a relação de eleitos e sexo e/ou faixa etária, entre outros no sentido plural, de partidos e candidatos e não apenas de um só, sendo assim, minha decisão de projeto foi localizá-los dessa forma. De forma a concluir, duas coisas importantes nessa produção de relatórios foram, o uso da classe “DecimalFormat” para fazer o arredondamento das casas decimais das porcentagens e imprimir no formato desejado e, ainda mais importante, as duas implementações da interface “Comparator”, “VotoLegendaComparator” e “VotoNominalUmCandidatoComparator” que são dois comparadores extras para ordenar a lista de partidos de forma diferente da implementação da interface “Comparable” da classe citada. O primeiro ordena para formatar a saída do relatório 7 e o segundo para formatar a saída do relatório 8. Optei em todos os casos em fazer a ordenação da saída dentro do método das eleições e não nos métodos dos relatórios, pois em qualquer aplicabilidade desse programa seria útil um critério de ordenação do relatório de saída para a facilitar a comparação, automática ou não.

2. Diagrama de classes UML da implementação:



Obs.: Como o diagrama ficou ilegível nesse formato, anexei o mesmo junto ao arquivo de entrega.

3. Testes:

Não houve nenhum erro ou bug conhecido durante os testes realizados pelo script, tudo ocorreu de acordo com o esperado.