# Pontifícia Universidade Católica do Rio Grande do Sul
# Programação Orientada a Objetos
# Prof. Marcelo H. Yamaguti
# 2024/2

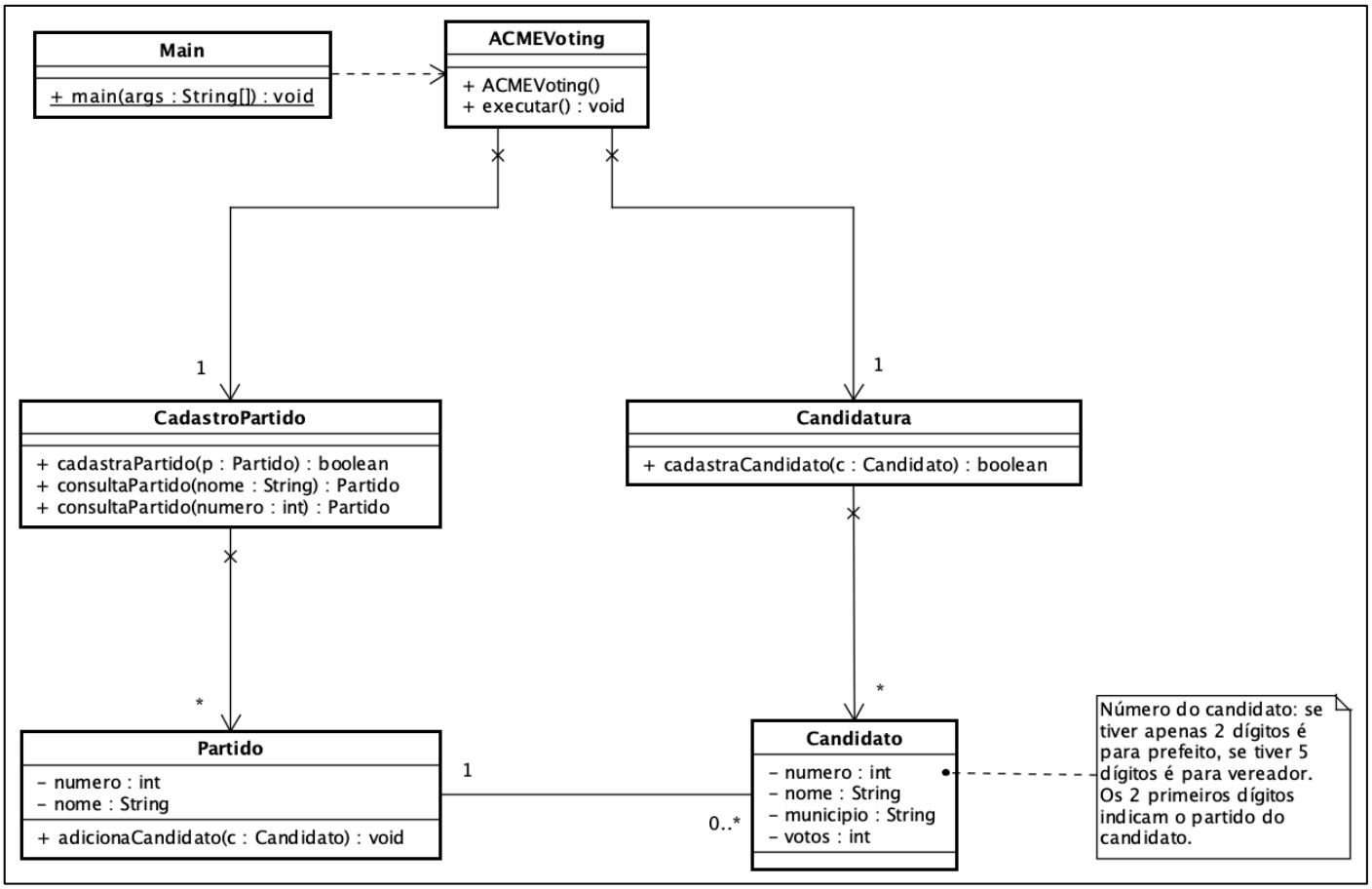
## Exercício de Avaliação 1

1. **Enunciado geral:**

A ACMEVoting é uma empresa independente que acompanha as eleições para prefeitos e vereadores.

Você será responsável pelo desenvolvimento de um aplicativo que acompanhará os votos dos candidatos e seus partidos.

O analista de sistemas identificou as seguintes classes iniciais, com alguns atributos e relacionamentos apresentados a seguir.




O analista identificou operações básicas iniciais das classes:

- **Partido**: classe que representa um partido na eleição:
  - **adicionaCandidato**(Candidato): adiciona um novo candidato ao partido.
- **CadastroPartido**: classe catálogo que gerencia o cadastro de partidos:
  - **cadastraPartido(Partido)**: recebe como parâmetro um novo Partido e o cadastra no sistema. Não pode haver partidos com o mesmo número. Retorna *true* se o cadastro teve sucesso; ou false em caso contrário.
  - **consultaPartido(String)**: retorna o partido com o nome indicado. Se não houver nenhum partido com este nome retorna *null*.

- **consultaPartido(int)**: retorna o partido com o número indicado. Se não houver nenhum partido com este número retorna *null*.

- **Candidatura**: classe catálogo que gerencia o cadastro de candidatos:
  - **cadastraCandidato(Candidato)**: recebe como parâmetro um novo candidato e o cadastra no sistema. Não pode haver candidatos com o mesmo número no mesmo município. Retorna *true* se o cadastro teve sucesso; ou *false* em caso contrário.
- **ACMEVoting**: classe da aplicação:
  - **ACMEVoting()**: construtor da aplicação.
  - **executar()**: executa o funcionamento da aplicação.
- **Main**: classe principal (inicial) do sistema:
  - **main(String[])**: cria um objeto ACMEVoting e depois chama o método executar().

O método executa() da classe ACMEVoting deve realizar a sequência de passos:

1. **Cadastrar partidos**: lê todos os dados de cada partido e, se o número não for repetido, cadastra-o no sistema. Para cada partido cadastrado com sucesso no sistema, mostra os dados do partido no formato: **`1:número,nome`**
2. **Cadastrar candidatos**: lê todos os dados de cada candidato e, se o número não for repetido no município e o partido existir, cadastra o candidato no sistema. Para cada candidato cadastrado com sucesso no sistema, mostra os dados do candidato no formato: **`2:número,nome,município`**
3. **Cadastrar votos de candidatos**: lê os votos de um determinado candidato. Se o número do candidato for válido, adiciona os votos do candidato. Para cada cadastramento com sucesso mostra os dados no formato: **`3:número,município,votos`**
4. **Mostrar os dados de um determinado partido pelo número**: lê o número de um determinado partido. Se não existir um partido com o número indicado, mostra a mensagem de erro: "**`4:Nenhum partido encontrado.`**". Se existir, mostra os dados do partido no formato: **`4:numero,nome`**
5. **Mostrar os dados de um determinado candidato**: lê um número de candidato e o município. Se não existir um candidato com o número indicado no município, mostra a mensagem de erro: "**`5:Nenhum candidato encontrado.`**". Se existir, mostra os dados do candidato no formato: **`5:numero,nome,município,votos`**
6. **Mostrar os votos dos prefeitos de um determinado partido**: lê o nome de um partido. Se não houver nenhum partido com o número indicado, mostra a mensagem de erro: "**`6:Nenhum partido encontrado.`**", caso contrário, mostra os dados de cada um de seus prefeitos no formato: **`6:nomepartido,númeroprefeito,nomeprefeito,municipio,votos`**
7. **Mostrar os dados do partido com mais candidatos**: localiza o partido com maior quantidade de candidatos. Se não houver partidos com candidatos, mostra a mensagem de erro: "**`7:Nenhum partido com candidatos.`**". Caso contrário, mostra os dados do partido e quantidade de candidatos correspondente no formato: **`7:número,nome,quantidade`**
8. **Mostrar os dados do prefeito e do vereador mais votados**: se não houver candidatos cadastrados no sistema, mostra a mensagem de erro: "**`8:Nenhum candidato encontrado.`**" Senão, mostra os dados do prefeito e do vereador mais votados no formato: **`8:número,nome,município,votos`**

## 2. Definição do exercício:

O objetivo do exercício é implementar um sistema que capaz de atender as necessidades da empresa descrita no enunciado geral, e que atenda as restrições a seguir:

- A entrada de dados ocorrerá por leitura de arquivo de texto. Ajuste a classe ACMEVoting para ler e escrever em arquivos: veja na área Moodle da disciplina > Módulo: Materiais de apoio > CÓDIGOS AUXILIARES > Redirecionamento de entrada/saída de dados para arquivos.
- Os dados de entrada estarão no arquivo **'input.txt'** e a saída deverá ser gravada no arquivo **'output.txt'**
    - No passo **1. Cadastrar partidos**: cada linha corresponde ao número e nome de um partido. Quando o número lido for -1, não há mais partidos a serem cadastrados.
    - No passo **2. Cadastrar candidatos**: cada linha corresponde ao número, nome e município de um candidato. Quando o número lido for -1, não há mais candidatos a serem cadastrados.
    - No passo **3. Cadastrar votos de candidatos**: cada linha corresponde ao número de um candidato e a quantidade de votos correspondente. Quando o número lido for -1, não há mais dados a serem cadastrados.
    - As últimas linhas do arquivo de entrada correspondem a:
        - Número do partido para o passo 4.
        - Número de um candidato para o passo 5.
        - Nome de um partido para o passo 6.
- Toda entrada e saída de dados com o usuário deve ocorrer apenas na classe ACMEVoting.
- É permitida a criação de novos métodos, atributos, classes e relacionamentos, mas as informações definidas no diagrama de classes original não podem ser alteradas.
- O diagrama de classes deve ser atualizado conforme as alterações realizadas e deve ser entregue em arquivo Astah ou PDF.

**3. Critérios de avaliação**

O exercício será avaliado conforme os seguintes critérios:

- Diagrama de classes atualizado e coerente com a solução: 1 ponto.
- Implementação correta conforme especificação e diagrama de classes: 4 pontos.
- Execução correta dos passos solicitados: 5 pontos.
    - Haverá comparação do arquivo gerado pela aplicação com o arquivo-teste disponível no Moodle (comando diff no Linux ou comp no Windows)

- *Ponto extra (opcional) de 1 ponto (máximo de 10 pontos): após a execução dos passos indicados:*
    - *Mostrar o partido com mais votos de vereadores: mostra os dado*s dos partidos com mais votos de vereadores no formato: **`9:número,nome,quantidade`**
    - Mostrar o município com maior quantidade de votos: mostra os dados do município que possuir maior quantidade de votos somados para prefeito e para vereadores no formato: **`10:municipio,quantidade`**

4. **Entrega:**
    - A entrega do exercício envolverá:
        - arquivos dos números-fonte do sistema (e demais arquivos necessários para a compilação do sistema).
        - arquivo com o diagrama de classes atualizado.
    - Deverá ser gerado um arquivo compactado (.zip ou .rar), com os itens acima, e entregue na tarefa da área Moodle da disciplina.
    - **Data de entrega**: 11 / 9 / 2024.

5. **Considerações finais:**
    - O exercício deve ser desenvolvido individualmente.

- A implementação deve seguir o Java Code Conventions para nomes de identificadores e estruturas das classes.
- Não será aceito exercício com erros de compilação. Programas que não compilarem corretamente terão nota zerada.
- A cópia parcial ou completa do exercício terá como consequência a atribuição de nota 0 (zero) aos exercícios dos alunos envolvidos. Para análise de similaridade será utilizado o MOSS (https://theory.stanford.edu/~aiken/moss/).