

Relatório de Projeto LP2 E-Câmara Organizada(E-CO)

Design Geral:

O design geral do nosso projeto foi desenvolvido de forma que o sistema seja facilmente extensível, pois utilizamos de mecanismos polimórficos como herança e interface para deixar o sistema o mais abstrato possível. Além do uso de um controller geral para delegar as atividades para os controllers mais internos e eles por sua vez delegam para as classes mais internas do sistema, diminuindo o acoplamento. Nos casos em que a entidade precisava de um comportamento dinâmico nós utilizamos o padrão strategy para possibilitar esse dinamismo. Quanto às validações, nós decidimos criar uma classe Validador que continha os métodos de validação necessários ao sistema e que era instanciada em todas as classes que iriam necessitar de validações. Os próximos tópicos detalham o que foi feito em cada caso.

Caso 1:

No caso de uso 1 é requerido que seja possível cadastrar pessoas que possuem um Documento Nacional de Identificação (D.N.I - documento que identifica a pessoa) nome, estado de origem (sigla de 2 letras), uma lista de interesses e, opcionalmente, partido. Assim sendo, viu-se a necessidade de se criar a classe Pessoa, que encapsula todos esses atributos e métodos como equals() e hashCode(), usados para comparar objetos do tipo Pessoa e toString(), que gera uma representação textual de Pessoa.

Dessa forma, sobrecarrega-se o método +cadastrarPessoa() na classe DeputadoController, de modo que ele possa receber todos esses parâmetros supracitados mais um partido ou não como parâmetro. Todos os parâmetros desse método são do tipo String, assim, checa-se se cada um deles são nulos ou vazios, e se forem, exceções do tipo NullPointerException e IllegalArgumentException serão lançadas, respectivamente. Ademais, checa-se se o DNI é válido, no formato "XXXXXXXX-X", no qual X deve ser um inteiro, se não for, lança-se um IllegalArgumentException. A classe ControllerGeral delega toda a operação para a classe DeputadoController, que por sua vez, armazena as pessoas cadastradas em um Map<String, Pessoa>, sendo o DNI a chave de cada pessoa cadastrada, já que pessoa é identificada por seu DNI.

Caso 2:

No caso de uso 2 pedido que seja possível cadastrar deputados a partir dos dados de uma determinada pessoa. Apenas é possível cadastrar deputados a partir de pessoas com partido. Com isso, é chamado o método +cadastrarDeputado() no deputadoController que por sua vez irá resgatar uma pessoa do tipo Pessoa no mapa de pessoas (Map<String, Pessoa>) e chamar o método +viraDeputado da classe Pessoa. Assim, uma pessoa comum passará a ter uma função deputado.

No entanto, para que a pessoa venha a se tornar um deputado, todos os dados passados como parâmetros devem ser válidos. Ou seja, se não forem, serão lançadas exceções adequadas para o tipo do erro.

Caso 3:

No caso de uso 3, requer a representação textual de uma pessoa, recuperada a partir do dni, a representação gerada a partir de seus atributos básicos e que a mesma seja diferente, para pessoas que são políticas e as que não são. No formato para pessoas, que não deputados: Nome - DNI (Estado) [- PARTIDO] [- Interesses], e para as que são, POL: Nome - DNI (Estado) [- PARTIDO] [- Interesses] - DATA - N Leis. Para atender isso, optamos por criar um método que é repassado para classe deputadoController, em que é verificado se a pessoa existe no mapa pessoas (Map<String, Pessoa>), caso ela não exista é lançada uma exceção, caso contrário é retornado o toString() de Pessoa.

O toString() da classe Pessoa, cria uma String que representa uma pessoa e se ela for política, a String formada em Pessoa é passada para a classe Deputado, a qual Pessoa compõe, concatenando assim com as informações do deputado e retornado para o toString de Pessoa.

Caso 4:

No caso de uso 4 é pedido que seja possível cadastrar e exibir partidos aliados do Governo no sistema. Foi criado um novo atributo chamado Base que representa uma coleção de partidos (HashMap) e um novo método chamado cadastraPartido() que recebe o partido a ser cadastrado que é do tipo String, então checa-se se o partido passado é vazio ou nulo, podendo lançar as exceções IllegalArgumentException e NullPointerException, respectivamente. Além disso, foi criado um método chamado exibirBase() que retorna uma representação de todos os partidos da Base. Para isso não foi necessária a criação de novas entidades, apenas foram adicionados novos atributos e métodos nas entidades já existentes.

Caso 5:

No caso de uso 5 é solicitado que seja possível cadastro de comissões definindo um interesse (tema da comissão) e uma lista de deputados participantes (códigos do DNI). A partir disso, criamos o método na classe ControllerGeral que verifica se a comissão já está cadastrada no mapa comissoes(Map<String, Comissao>), caso ele já exista é lançada uma exceção, caso contrário é salvo no mapa, com a chave sendo o tema da comissão e o valor do mapa, sendo um objeto da classe Comissao, ou seja, para o caso de uso 5, também criamos uma classe para guardar todos os atributos e métodos referente a Comissao, mantendo assim o encapsulamento.

Para criar o objeto da classe Comissao, criamos um método privado, para gerar o array de DNIs válidos, em que verifica se a pessoa está cadastrada no sistema, através de um método no DeputadoController, o qual verifica no mapa de pessoas, citado no caso de usos acima, se uma pessoa está cadastrada no sistema, caso não esteja é retornado false, caso esteja é retornado true. Além disso, criamos um método, também no DeputadoController, o qual verifica se uma pessoa é deputado, caso seja é retornado true, caso contrário é retornado false. Se um dos métodos retornarem false, é lançada uma exceção e caso não, é retornado o array de DNIs.

Caso 6:

No caso de uso 6 é requerido que seja possível cadastrar três tipos diferentes de lei:

1. Projeto de Lei: constituído de dni do seu autor, ementa, interesses, url, todos do tipo String, ano do tipo int e conclusivo do tipo boolean.
2. Projeto de Lei Complementar: constituído de dni do seu autor, ementa, interesses, url e artigos, todos do tipo String, e ano do tipo int.
3. Projeto de Emenda Constitucional: constituído de dni do seu autor, ementa, interesses, url e artigos, todos do tipo String, e ano do tipo int.

Sendo assim, por meio de polimorfismo por subtipo (herança), abstrai-se a classe `ProposicaoAbstract`, que agrega atributos (código da lei, dni do seu autor, ementa, interesses, url, ano e situação) e métodos (o construtor, `+equals(): boolean`, `+hashCode(): int`, `+toString(): str`) comuns aos três tipos de projetos de leis definidos pelo caso de uso 6, que por sua vez, carregam seus atributos específicos além dos que elas herdaram.

Os métodos `cadastrarPL`, `cadastrarPLP` e `cadastrarPEC` na classe `ProposicaoController` são responsáveis, respectivamente, por cadastrar Projetos de Lei, Projetos de Lei Complementar e Projetos de Emenda Constitucional. No momento de cadastro de cada tipo de projeto de lei, checka-se se cada um dos parâmetros são nulos ou vazios, e se forem, exceções do tipo `NullPointerException` e `IllegalArgumentException` serão lançadas, respectivamente. Ademais, verifica-se se o dni é de uma pessoa que existe no sistema e é deputado, mas se não for lança-se `NullPointerException`. Por conseguinte, valida-se o ano de cadastro da lei, que não pode ser anterior a 1988 e posterior ao ano atual.

Para se ter controle sobre a quantidade de um determinado tipo de leis cadastradas para um ano, viu-se a necessidade de criar uma classe que realizasse esse comportamento de contagem, que é a classe `Contador`, cujo atributo `contagem` por ser recuperado por meio de um `get()` e seu valor é utilizado para compor o código de lei a ser cadastrado.

Os contadores são armazenados em um `HashMap<String, Contador>`, no qual a chave é a concatenação da String referente ao tipo de lei e o ano de cadastro, exemplo: "PLP2016". Já os projetos de leis cadastrados, são armazenados em uma `HashMap<String, ProposicaoAbstract>` no qual a chave é o código da lei, exemplo: "PLP 5/2014".

Por fim, é requerido que seja possível exibir um projeto de lei, recuperável pelo seu código de lei nos seguintes formatos:

PL: Projeto de Lei - Código - DNI - Ementa - Conclusivo - Situação

PLP: Projeto de Lei Complementar - Código - DNI - Ementa - Artigos - Situação

PEC: Projeto de Emenda Constitucional - Código - DNI - Ementa - Artigos - Situação

Portanto, por meio do método `exibirProjeto()`, que pesquisa (por meio do código da lei passado como parâmetro) no mapa que armazena os projetos de lei e retorna a representação textual da lei, se existir, ou lança `NullPointerException` se não existir.

Caso 7:

No caso de uso 7 é requerido que seja possível realizar votações de leis nas comissões e no plenário.

Para realizar a votação no plenário e nas comissões foi necessário criar alguns atributos auxiliares na classe `ProposicaoAbstract`, são eles: `passouNoPlenario` (um boolean que indica se uma determinada lei já foi votada pelo menos uma vez no plenário) e `passouNaCCJC` (um boolean que indica se uma determinada lei já foi votada na CCJC).

Nas comissões, uma proposta é considerada aprovada quando a metade dos deputados mais um vota favorável. Há duas formas de votar uma comissão. Se o PL for conclusivo, ele é imediatamente APROVADO ou REJEITADO. Se o PL não for conclusivo, ele pode continuar sendo votado até chegar ao Plenário mesmo que haja pedidos de rejeição pelas comissões.

Além disso, para votação no plenário foi necessário definir o quórum mínimo (mínimo de deputados presentes para realizar a votação) e chão (quantidade mínima de votos para aprovar determinada lei). Foi criado o método `votarPlenario()` que recebe como parâmetros o código de uma lei, o `statusGovernista` da lei (GOVERNISTA, OPOSICAO ou LIVRE) e uma String com os DNIs de todos os deputados presentes, separados por vírgula. Além de verificar o quórum mínimo, todos esses parâmetros são verificados e caso sejam inválidos as devidas exceções são lançadas. A forma de votação no plenário varia para cada tipo de lei:

1. PL não conclusiva: Só pode ser votada uma vez no plenário e seu quórum mínimo é definido da seguinte forma: $(\text{total deputados} / 2) + 1$ e o seu chão é definido da seguinte forma: $(\text{deputados presentes} / 2) + 1$.
2. PLP: Pode ser votada duas vezes no plenário (caso seja aprovada na primeira votação, caso contrário é arquivada), seu quórum mínimo e chão são definidos da seguinte forma: $(\text{total deputados} / 2) + 1$.
3. PEC: Pode ser votada duas vezes no plenário (caso seja aprovada na primeira votação, caso contrário é arquivada), seu quórum mínimo e chão são definidos da seguinte forma: $((\%) * \text{total deputados}) + 1$.

Caso 8:

No caso de uso 8 é requerido que seja possível exibir a tramitação de uma tramitação de lei, pesquisada no mapa que armazena os projetos de lei por meio de seu código de lei. Com isso, na classe `ProposicaoAbstract` criou-se o atributo `tramitacao`, que consiste em um `ArrayList` que armazena todas as situações pelas quais o projeto de lei passou.

Há ainda, o método `atualizaTramitacaoLei`, que vai adicionando ao atributo `tramitacao` cada situação e parecer de uma determinada votação para uma determinada lei. Por fim, invoca-se o método `join()` de String no método `exibirTramitacao()` para separar cada situação presente no `ArrayList` e gerar uma String final da tramitação da lei que será retornada.

Se ao pesquisar a lei no mapa e ela não existir, lança-se Nullpointer Exception.

Caso 9:

No caso de uso 9 é pedido que seja possível buscar a proposta mais relacionada com os interesses de uma pessoa, além disso, requer que o usuário possa configurar a estratégia de desempate da busca, ou seja, escolher entre essas três estratégias: CONSTITUCIONAL, CONCLUSAO, APROVACAO, se empatar novamente deve ser retornada a mais antiga e caso não exista proposta que tenha, pelo menos, um interesse em comum com o usuário, uma string vazia deve ser retornada. Com base nisso, criamos no ControllerGeral os métodos configurarEstrategiaPropostaRelacionada, a qual recebe um dni e uma estratégia, ambos Strings, o método valida e passa para o PessoaController, que verifica se a pessoa existe e caso não exista é lançada exceção e caso contrário, é repassado para classe Pessoa, em que utilizamos o padrão Strategy, ou seja, dependendo da estratégia passada, um atributo que consiste de uma interface de estratégias de desempate recebe um objeto da estratégia desejada, assim permitindo o dinamismo do sistema, simplificação da lógica e permitindo códigos mais flexíveis e reutilizáveis.

Além do método acima, criamos no ControllerGeral o método pegarPropostaRelacionada, o qual recebe como parâmetro uma String, que representa o dni de uma pessoa, para fazer o método foi necessário criar um getInteresses da pessoa e um getProposicoesDeLei, onde comparamos para ver qual proposta é mais similar com a pessoa, representada pelo dni, se acontecer de empatar, utilizamos um getEstrategiaBuscaProposta, em que pega o objeto da estratégia da pessoa e dela é repassada para uma das estratégias uma lista das estratégias empatadas, caso chegue a empatar novamente, dentro das estratégias é chamado o ComparadorPropostaCodigo, o qual implementa Comparator, retornando assim a proposta mais relacionada com pessoa.

Caso 10:

No caso de uso 10, é requerido que seja possível realizar operações de carregar, salvar e limpar sistema. Para isso, foi necessário implementar a interface serializable em todas as classes do pacote entidades e util, isso permite salvar o estado atual dos objetos em arquivos em formato binário para um arquivo, no nosso caso, arquivos do tipo '.dat', sendo assim, esse estado poderá ser recuperado posteriormente recriando o objeto em memória assim como ele estava no momento da sua serialização. Além disso, criamos a classe Persistencia, a qual gerencia os arquivos do projeto, nela há três métodos, salvar() sistema, que cria um ObjectOutputStream, escrevendo os dados dos objetos em um arquivo, também há o carregar(), que cria um ObjectInputStream, lendo os dados de um arquivo para um objeto, e o último método é o limpar(), que cria/sobrescreve arquivos em branco.

No ControllerGeral foi necessário criar três métodos. O primeiro método foi o limparSistema(), o qual cria/sobrescreve os arquivos do tipo '.dat', com arquivos vazios, das coleções do ControllerGeral no diretório files/ do projeto, e chama o método limparSistema() dos demais controllers, o qual faz o mesmo procedimento, porém com suas respectivas coleções. O segundo método foi o salvarSistema(), que tem a responsabilidade de

armazenar as informações do projeto, ou seja ele grava os dados das coleções do ControllerGeral em um arquivo do tipo '.dat' e chama o método salvarSistema() dos demais controllers, o qual também faz o mesmo procedimento, porém com suas respectivas coleções. Por fim, o terceiro método foi o carregarSistema(), o qual lê todos os objetos salvos nos métodos anteriores e adiciona os valores nas variáveis compatíveis, depois de uma operação de casting para o verdadeiro tipo delas, pois todas são lidas como Object. Como os métodos anteriores, também é chamado o método carregarSistema() dos demais controllers, o qual faz o mesmo procedimento, porém com suas respectivas coleções.

Considerações:

Organizamos e controlamos o projeto através de Issues, ficando assim mais fácil a correção do problema e a obtenção das informações para simulação dos problemas.

Eric ficou responsável pelos casos de uso 1, 6 e 8. E testar 4, 7 e 9.

Gabriel ficou responsável pelos casos de uso 3, 5, 9 e 10. E testar 2, 7, e 8.

Victor ficou responsável pelos casos de uso 4, 7, 9. E testar 1, 6 e 8. E refatoração.

Wellisson ficou responsável pelos casos de uso 2, 7. E testar 3, 5, 9. E refatoração.

Membro(s):

Eric Diego Matozo Goncalves - 118210349 (<https://github.com/ericdmg>)

Gabriel Mareco Batista de Souto - 118210258 (<https://github.com/gabrielmbs>)

Victor Brandão de Andrade - 118210406 (<https://github.com/victorbrandaoa>)

Wellisson Gomes P. B. Cacho - 118210873 (<https://github.com/wellissongomes>)

Monitor:

Jadson Luan (<https://github.com/jadsonluan>)

Link para o repositório:

<https://github.com/gabrielmbs/E-CO>