Relatório de projeto LP2 _ E-CO

Design geral: O design geral do nosso projeto foi escolhido com o objetivo de possibilitar uma melhor integração entre as partes do projeto, utilizando métodos que diminuem o acoplamento, e consequentemente, a dependência. Uma das estratégias utilizadas foi o uso de um Controller Geral que delega ações para diferentes Controllers(ControllerComisao, ControllerPartidosGovernistas, ControllerPessoa, ControllerProjetos e ControllerVotacao). Esses possuem alguns métodos que delegam atividades, a fim de aumentar o nível de abstração do sistema. Quando uma entidade precisou de comportamento dinâmico utilizamos o padrão Estrategy. Um exemplo desse tipo de entidade foi a Estrategia de Ordenação. Para lançar exceções utilizamos uma classe com métodos que fazem a avaliação dos dados que são recebidos pelo sistema. Em relação à organização das classes separamos em pacotes (Comparators, Controllers, Entidades, Estratégias, Projetos, Sistema) que reúnem as classes de natureza semelhante.
As próximas seções especificam melhor a implementação de em cada caso.

Caso 1: O caso 1 pede que seja criado uma entidade que representa uma pessoa. Para isso optamos por criar uma classe pública chamada *Pessoa* que irá representar essa pessoa. A pessoa tem nome, dni, estado, interesses e partido como atributos. Como uma pessoa pode ter um partido ou não, tem-se a necessidade de criar dois construtores no qual poderá receber uma pessoa sem partido ou uma pessoa associada a algum partido político. Toda pessoa começa como um civil, desse modo, existe uma entidade *Civil()*, ela implementa uma interface(*Função*). A interface *Função* possui o método *representação(String nome, String dni, String estado, String interesses, String partido)* que é responsável por definir o tipo de representação da pessoa (pessoa civil ou deputado). Como uma pessoa tem um tipo de representação específica, a classe *Pessoa* tem uma composição dessa interface. Para gerar/gerenciar uma pessoa, foi criada uma classe *ControllerPessoa()*. Existe uma coleção nessa classe que armazena as pessoas. Um mapa, que tem como chave um DNI e o valor é uma *Pessoa*.

Caso 2: O caso 2 pede que seja criado um deputado a partir de uma pessoa já existente no sistema. Para isso criamos a entidade *Deputado* que representa um deputado, no qual possui os atributos *quantidades de lei* e *data de início*. A entidade implementa a interface *Função*. Para gerar/gerenciar um deputado, foi criada a classe ControllerDeputado(), que por sua vez possui uma Coleção. Essa coleção é um mapa que tem como chave o DNI dele pois ele nunca deixa de ser uma pessoa e o valor é a *Pessoa*. Porém, para que não ocorra problemas como quebra de expert da informação, optamos por criar um *ControllerCentral()* que será responsável por manipular as informações tanto do *ControllerPessoa()* como do *ControllerDeputado()*. O controlador central possui uma composição do controlador de pessoa e controlador de deputado, sendo assim, é possível manipular/obter informações a respeito de uma pessoa, se ela é deputado ou não, etc.

Caso 3: O caso 3 pede para que seja exibido, uma pessoal ou deputado a partir do seu DNI. Para isso optamos por criar um método no controlador central (*exibePessoa*) que irá

verificar se essa pessoa existe no mapa que está armazenado no controlador de pessoas. Caso exista, a pessoa/deputado é exibida, senão uma exceção é lançada indicando o erro.

Caso 4: O caso 4 pede para que seja cadastrado partidos da base e estes sejam exibidos. Para isso optamos por colocar um Set do tipo String que armazena cada partido. Se a operação de cadastrado for bem sucedida, nada acontece. Caso contrário, uma exceção é lançada indicando o erro. Os partidos governistas são exibidos a partir do método *exibirBase()* que irá exibir os partidos em ordem lexicográfica e separado por vírgula.

Caso 5: O caso 5 pede para que seja cadastra as comissões. Para isso decidimos criar uma classe pública *Comissão* que representa uma comissão. Essa comissão por sua vez possui um tema e uma lista de deputados que irão compor essa comissão. Para cadastrar e manipular os dados de uma comissão, nós decidimos criar um controlador específico para a comissão *ControllerComissão()*, este será responsável por cadastrar uma comissão, exibir uma comissão, além de validar se a comissão existe ou não. Caso a comissão não exista, sua exibição não será realizada e uma exceção é lançada.

Caso 6: O caso 6 pede para que seja cadastrado e exibido um projeto. Para essa unidade, decidimos fazer uma classe abstrata *Projeto,* utilizando de polimorfismo e herança, essa classe tem as filhas *ProjetoEmentaConstitucional (PEC), ProjetoLei (PL) e ProjetoLeiComplementar (PLC).* Todo projeto possui um deputado (responsável), ano de criação, código (identificador único), ementa, interesses, uma url (acesso ao site do projeto) e uma lista de tramitação que contém todos os locais por onde ela passou (se foi aprovado em comissões, plenário etc). Para exibição de um projeto optamos por criar um controlador para isso, *ControllerProjetos*, este por sua vez, é responsável por cadastrar um projeto, exibir (*exibirProjeto*), dentre outras funcionalidades. Como cada projeto possui um *toString()* específico, a impressão de um projeto a partir de seu código não terá problemas de exibição. Vale ressaltar que tanto o cadastro como a exibição de um projeto só é possível se os parâmetros passados forem válidos. Caso contrário, exceções são lançadas.

Caso 7: O caso 7 pede para que seja votado uma proposta legislativa. Para isso, ainda na classe abstrata *Projeto,* como cada projeto (Pec, plc e pl) tem sua prerrogativa de votação, essa classe por sua vez, possui os métodos abstratos *votarComissão(), votarPlenario() e validaQuorum().* Desse modo, cada classe filha implementa as funções supracitadas de sua forma. Para manipulação de votação de um projeto, criamos um controlador de votação que será unicamente responsável pela votação do projeto, *ControllerVotação(),* o controlador possui um mapa de deputados, um mapa de comissões e um mapa de projetos. Para a votação no plenário, o controlador possui o método *votarPlenario()* , este método verifica inicialmente se o status do projeto é governista (a favor), oposição ou livre. Feita a verificação, caso o status seja governista, o método *votarPlenario()* é chamado e a votação é iniciada, retornado um valor boolean, indicando se foi aprovado ou não no plenário. Da mesma forma ocorre caso o status seja oposição, o método é invocado, assim como se o status for livre, o método é invocado. Para votação na comissão, ocorre da mesma forma, é verificado o status governista e após isso, é invocado o método de *votarComissão()* presente na classe projeto, que possui seu próprio meio de votação a partir do status governista que foi definido para o projeto. Para que tenha-se um baixo acoplamento e uma

alta coesão, o Controller*Central()* possui os controladores de votação e projeto, possibilitando assim, que o controlador central possua a informação dos outros dois controladores e possa manipular de forma fácil as ações solicitadas. Vale lembrar que todas as ações que são feitas, antes são validadas através da classe *Validação()* e caso algum dos parâmetros passados seja inválido, uma exceção é lançada.

Caso 8: O caso 8 pede para que sejam exibidas as tramitações de uma proposta. Para isso, optamos por usar um ArrayList de String em cada proposta, armazenar o status de cada tramitação nesse ArrayList, na medida que aconteçam. Desse modo, a todo momento o ArrayList estará atualizado. As tramitações podem ser exibidas separadas por vírgula, através do método exibirTramitacao(), implementado na classe Proposta. As validações requeridas para tal funcionalidade, são feitas através da classe Validação.
Caso 9: O caso 9 pede para que a proposta mais semelhante aos interesses de uma pessoa seja retornada ao ser chamado um método, e pede também para que o método de desempate possa ser escolhido por cada pessoa. Por isso optamos por usar o padrão Strategy, e visando a flexibilidade do código, criamos 3 métodos de desempate, são eles: EstrategiaAprovacao(), EstrategiaConclusao() e EstrategiaConstitucional(), usando seus respectivos comparators. Eles podem ser selecionados através do método configurarEstrategiaPropostaRelacionada(String dni, String estrategia). Com isso, a proposta mais adequada é retornada a partir de um dni de uma pessoa que é passado como parametro. Vale ressaltar que as validações necessárias estão sendo realizadas na classe Validacao().

Caso 10: O caso 10 pede para que sejam implementados métodos de persistência no sistema ECO(Salvar, Carregar e Limpar). Para isso, foi criada a classe GerenciadorArquivos(), que vai gerenciar toda a parte de carregar, limpar e salvar os arquivos .bin das coleções do sistema. Optamos fazer isso para que seja facilitado o gerenciamento de arquivos, jogando a responsabilidade em uma classe só.