Relatório de Projeto LP2

Design geral:

O design geral do nosso projeto foi feito visando uma maior integração entre as diversas partes do sistema, através de um controllerGeral, no qual está ligado à Facade e aos outros controllers específicos, facilitando a manipulação dos dados armazenados por estes. Utilizamos, também, de facilidades localizadas na pasta Util, com o objetivo de padronizar as validações das informações passadas nos métodos.
Além disso, com o intuito de abstrair o código, buscamos manter uma relação de herança entre as entidades, como a observada entre as entidades Pessoa e Deputado, esta sendo uma extensão de Pessoa, e a entre PropostaLegislativa e seus tipos, PL, PLP e PEC.
Ademais, além dessas estruturas, também buscamos utilizar, a partir da parte 2, um controllerVotacao, com o intuito principal de realizar a votação das Propostas.
Em geral, em todas as partes do projeto que envolvia um dinamismo, buscamos optar por métodos que traziam pontos positivos e facilidades no desenvolvimento do projeto e na reutilização do código.

Caso 1:

O caso de uso 1 pede que possa ser possível cadastrar diferentes pessoas no sistema. Para isso, optamos por criar a classe Pessoa, contendo as informações básicas de uma pessoa (Nome, DNI (código de identificação), Estado de origem, Interesses e Partido). Além disso, com o objetivo de criar os dois métodos de cadastrarPessoa(), um com e outro sem o parâmetro “partido”, além de todos os outros atributos básicos de Pessoa. Para gerenciar as ações feitas em Pessoa, foi criada a classe ControllerPessoa, contendo o atributo mapa de pessoas para armazenamento de vários objetos do tipo Pessoa, passando-se como chave identificadora de cada objeto, o seu próprio DNI. Optamos por usar essa coleção por causa da facilidade de identificação do objeto pessoa, acessando-o de forma mais simples e mais legível ao programa. Além disso, lançamos as exceções, utilizando-se das facilidades proporcionadas pelos métodos estáticos da classe auxiliar Validador, para os casos de:
Nomes, DNIs ou estados vazios, DNIs compostas por algo além de números e do hífen(“-”) e o cadastro de pessoas com mesmo DNI, evitando, assim, problemas futuros no código.

Caso 2:

O caso de uso 2, refere-se ao método cadastrarDeputado() que na sua essência, só pode ser uma Pessoa com atributo partido. Para facilitarmos, usamos herança onde a classe Deputado é classe filha da classe Pessoa, logo, se uma Pessoa desejar ser Deputado, o dni continua sendo o código de identificação padronizado da Pessoa, sendo assim, estando nulo, vazio ou ser composto por caracteres além de números e traços, uma exceção é automaticamente lançada de acordo com o defeito encontrado. Caso contrário, verifica-se caso essa Pessoa se encontra cadastrada no mapa de pessoas, senão, lança-se uma outra exceção. Após isso, há a checagem caso tenha-se partido sabendo-se que,

como já mencionado, uma Pessoa só pode ser Deputado se tiver um partido como atributo. Por fim, tendo passado por essas exigências, cadastra-se o Deputado a partir de uma DataDeInicio, em forma de String, que é verificada nos requisitos: ser uma data futura, ser uma String vazia ou nula e se for com composta por caracteres além de números. Vale salientar que todas essas validações, foram feitas durante esse processo de cadastro, foram feitas a partir de métodos estáticos da classe Validador. Após cadastrar o Deputado, o objeto Pessoa com o mesmo dni encontrado no mapaPessoas é removido.

Caso 3:

O caso de uso 3 pede que seja possível representar textualmente uma pessoa, sendo ela um deputado ou não, havendo, assim, duas formas diferentes de representação. Para tal objetivo, criamos, no ControllerPessoa, o método exibirPessoa(), passando como único parâmetro o DNI da pessoa, exibindo o toString de Pessoa. Além disso, lançamos as exceções, utilizando-se das facilidades proporcionadas pelos métodos estáticos da classe auxiliar Validador, para os casos de:
DNIs compostas por algo além de números e do hífen (“-”) e exibir Pessoas que não estão cadastradas no sistema.

Caso 4:

O caso 4 pede que seja possível gerenciar o cadastro de partidos governistas e para isso, tem-se o método cadastraPartido() que recebe unicamente a String com o nome do partido a ser cadastrado e valida-se essa String é vazia ou nula. Caso contrário, a String será adicionada em uma List partidos em prol da armazenamento de todas os partidos que serão cadastrados. Posteriormente, a lista partidos será ordenada lexicograficamente no método exibirBase(), o qual exibirá todas as Strings cadastradas.

Caso 5:

O caso de uso 5 pede que seja possível cadastrar comissões temáticas de modo que possa haver discussões mais aprofundadas sobre determinados temas. Para isso, optamos por criar a entidade Comissão, contendo os atributos tema, o identificador único da comissão, e uma lista de DNIs, sendo uma String, contendo os DNIs de todos os deputados participantes da Comissão. Para criação do método cadastrarComissao(), contendo como parâmetros o tema e esta lista de DNis, criamos, também, o ControllerComissao, contendo o atributo mapa de comissões para o armazenamento de vários objetos do tipo Comissão, passando-se como chave identificadora de cada objeto o seu próprio tema. Optamos por usar essa coleção por causa da facilidade de identificação do objeto comissão, acessando-o de forma mais simples e mais legível ao programa. Além disso, lançamos as exceções, utilizando-se das facilidades proporcionadas pelos métodos estáticos da classe auxiliar Validador, para os casos de:
Nomes, temas ou strings de políticos vazias ou nulas, DNIs compostas por algo além de números e do hífen (“-”), cadastro de tema já cadastrado anteriormente, evitando, assim, problemas na identificação, cadastro de uma pessoa inexistente e o cadastro de uma pessoa que não é política.

Caso 6:

O caso 6 refere-se unicamente no cadastro e exibição das propostas legislativas envolvidas no desenvolvimento do processo. Inicialmente, decidimos optar por usar herança como design dessa US6 já que existem 3 tipos de propostas legislativas envolvidas. As propostas PLP, PEC e PL possuem atributos em comum que são colocados na super classe abstrata Proposta Legislativa, porém há atributos que são particulares de cada proposta legislativa, logo, possuem dessa forma um toString() diferente para cada uma. Para gerenciar as ações envolvidas sobre essas propostas legislativas, foi criado um controlador que possui justamente um mapa que armazenará todas as propostas cadastradas após todas as validações exigidas pela especificação do projeto e em cada processo de cadastro, há a atualização dos contadores que controlam a quantidade de propostas legislativas de cada tipo.

Caso 7:

O caso de uso 7 pede para que seja possível votar uma proposta legislativa, podendo assumir 2 locais de votação, comissão ou plenário. Para ajudar no controle dos dados necessários à votação, tais como: mapas de comissões, deputados, projetos de leis, etc. foi criado o controllerVotacao, no qual está ligado ao controllerGeral, onde contém-se os métodos principais votarComissao(), recebendo como parâmetro o DNI do autor, o status governista(podendo ser governista,oposição ou livre) e o próximo local para onde a votação prosseguirá, e votarPlenario(), recebendo como parâmetro o DNI do autor da proposta de lei, e uma lista, no formato de string, contendo os DNI's dos deputados presentes na votação, além disso, utilizamos de métodos auxiliares nas construções dos principais, tais como: aprovaGoverno(), verificaInteresse(), aprovaPlenarioGovernista(),etc.
Além disso, lançamos as exceções para os casos de:
Strings de código, comissão, próximo local vazios e presentes vazios, DNIs compostas por algo além de números e do traço ("-"), código de um PL inexistente, DNI do deputado presente não cadastrado no sistema, realizar votação de PL conclusivas já votadas em comissões de mérito (qualquer comissão além do CCJC), realizar votação em comissão de PLs não conclusivos, de PLPs e PECs já encaminhados ao plenário, realizar votação no plenário sem presentes, realizar votação no plenário sem quórum mínimo, etc.

Caso 8:

O caso de uso 8 pede que seja possível exibir a tramitação de uma Proposta Legistativa, sendo ela uma PL, PLP ou PEC. Para que seja possível a construção dessa funcionalidade, decidimos utilizar da classe abstrata já criada PropostaLegislativa e de um método, igualmente abstrato, exibirTramitacao(), no qual recebe como parâmetro o código da Proposta, e com ele, cada tipo de PL tem seu próprio padrão de tramitação. Lançamos as exceções para o inserimento de um código vazio ou um código de um PL inexistente.

Caso 9:

O caso de uso 9 tem como objetivo apresentar as propostas legislativas mais relevantes para uma pessoa através de até 3 maneiras de apresentação (“Constitucional, Conclusão e Apresentação”). Foram implementados dois métodos na classe ControllerPLS, o “pegarPropostaRelacionada” que exibe de maneira padrão a proposta mais relevante na pesquisa de forma “CONSTITUCIONAL” e o “configuraEstrategiaProposta” que altera o modelo de pesquisa da proposta para uma maneira escolhida pelo usuário. No entanto, não conseguimos completar o desenvolvimento dessa US. Dessa forma, não apresenta uma lógica completa, falhando em realizar a mudança de modelo de busca, gerando assim erros de pesquisa.

Caso 10:

O caso de uso 10 envolve-se unicamente com a persistência de arquivos, logo, para isso, foi implementado maneiras em cada controller específico para gravar e recuperar as informações da última execução do programa. Ao gravar as informações, o mapa contido no controller é relacionado com o arquivo onde ficará as informações. Já ao recuperar essas informações, esse arquivo é resgatado e o mapa do método é referido as informações já salvas no arquivo. Logo, para iniciar o sistema, ler-se o arquivo com as informações já salvas e para finalizar, no entanto, grava todas as novas informações.

Considerações:

- Optamos por fazer o projeto todo em português em prol da padronização dos métodos e para evitar possíveis dificuldades de entendimento iniciais.
- O uso de um controlador geral foi visto como maneira de facilitar o relacionamento entre os controladores mais específicos. Tendo assim, os demais controladores como atributos.
- Utilizamos as Classes Validador e Conversão para auxílio no ato de validação dos parâmetros e em tratamento de exceções.