Relatório de projeto LP2

Caso 1:
Para o caso 1, criamos a entidade PessoaCivil que encapsula uma Pessoa, contendo informações como nome, dni, estado, interesses e partido. Para armazená-los, criamos a classe PessoaController. Ela possui um HashMap de Pessoa onde a chave é o DNI, bem como métodos de cadastro de pessoas no sistema.

Caso 2:
No caso 2, pede-se que seja feita a funcionalidade de cadastrar deputados. Para isso, criamos a entidade Deputado que encapsula um Deputado, contendo informações como data de início e quantidade de leis aprovadas. Como um deputado é criado a partir do DNI de uma pessoa, criamos a interface PessoaInterface e trocamos o tipo no HashMap definido no caso 1 para PessoaInterface. Dessa forma, a classe PessoaCivil e Deputado implementam a interface PessoaInterface, permitindo, assim, que Pessoa e Deputado possam ser armazenados na mesma coleção. A fim de evitar a perda da referência Pessoa no momento da criação do Deputado, um deputado possui além dos atributos dataDeInicio e leisAprovadas um atributo pessoa do tipo PessoaCivil que é responsável por guardar a referência ao Objeto Pessoa que foi usado como base para sua criação.

Caso 3:
No caso 3, pede-se que seja feita a funcionalidade de exibição de Pessoa/Deputado. Para isso, foi feito o método toString() nas entidades PessoaCivil e Deputado. Foi criado o método exibirPessoa na classe PessoaController e este retorna o toString de uma Pessoa a partir do seu DNI. O uso de composição com interface permitiu que o método se comporte de forma polimórfica, pois a decisão do toString chamado será tomada em tempo de execução a partir do tipo mais específico ao qual àquele DNI pertence. Não foi necessário criar novas entidades, apenas adicionar funcionalidades nas que foram criadas nos casos de uso 1 e 2.

Caso 4:
No caso 4, pede-se que seja feita a funcionalidade de cadastrar partidos governistas.
Para isso, foi criada a entidade PartidoController que possui um TreeSet de Strings, onde a String representa o nome do partido, bem como métodos de cadastro de partido e exibição de base. A decisão de armazenar os partidos em um TreeSet foi tomada com base na funcionalidade de exibirBase, pois a exibição é feita em ordem lexicográfica. Dessa forma, não foi preciso fazer uma nova ordenação, pois no momento de cadastro um partido já é ordenadamente adicionado.

Caso 5:
No caso 5, pede-se que seja feita a funcionalidade de cadastrar comissão. Para isso, foi criada a entidade Comissao que armazena um HashSet com os deputados que fazem parte dela. Para o gerenciamento das comissões cadastradas, foi criada a Classe CamaraController que possui um HashMap de Comissao, bem como o método de cadastro de comissão no sistema. Para se fazer as validações no cadastro de comissões, era necessário que se tivesse acesso a entidade PessoaController. A fim de reduzir

acoplamento, foi criada a classe SystemController que é responsável por gerenciar os controladores mais internos do sistema. Dessa forma, o SystemController recebe requisições da Facade, realiza validação em métodos que fazem uso de dados presentes em mais de um controlador e delega os métodos para os controladores mais internos.

Caso 6:
No caso 6, pede-se que seja feita as funcionalidades de cadastrar e exibir proposta legislativa.
As propostas podem ser PL(Projeto de lei), PLP(Projeto de Lei Complementar) e PEC(Proposta de Emenda Constitucional). Para representar esses diferentes tipos, foi criada a classe abstrata ProjetoDeLei que contém os atributos e métodos comuns a todos os tipos. Para armazenar e gerenciar as leis cadastradas no sistema, criamos a classe ProjetosDeLeiController que possui um HashMap de ProjetoDeLei, onde a chave é a String composta por um código sequencial gerado pelo sistema com o ano de criação da lei. Para a exibição de diferentes projetos, apenas foi necessário definir o toString em cada tipo.

Dessa forma, o uso de Herança permitiu além de reúso de código e tipo o uso de polimorfismo, pois o método exibirProjeto presente na classe ProjetosDeLeiController retorna o toString do código de um projeto passado como parâmetro. Este, por sua vez, se comporta de forma polimórfica, pois a decisão do toString chamado será feita em tempo de execução a partir do tipo mais específico do objeto. A entidade ProjetosDeLeiController também é gerenciada pelo SystemController.

Caso 7:
No caso 7, pede-se que seja feita funcionalidade de votar proposta legislativa. Para isso, foram criadas três enumerações na classe abstrata ProjetoDeLei, sendo elas: TipoDeLei, StatusDaLei e StatusPlenario, com o objetivo de facilitar as verificações nos métodos de votação. A fim de encapsular a lógica de votação de uma proposta legislativa, decidimos dividi-la em duas partes: Para votações de projetos de lei em comissões, criamos o método votarComissao na classe Comissao. Esse método recebe um projeto de lei, realiza a votação e retorna um boolean que identifica se a lei foi aprovada ou rejeitada (true se aprovada e false se rejeitada) para uma determinada comissão. Já para a votação no Plenario, foi criada a classe Plenario que armazena um HashSet com os deputados cadastrados no sistema. A entidade Plenario também possui o método votarPlenario que é responsável por realizar a votação de um projeto de lei no plenario. Se o projeto de lei for aprovado, o método retorna true, caso rejeitado, retorna false. Para gerenciar e encapsular as Comissoes e o Plenario, foi criada a classe a classe CamaraController. A entidade recebe requisições do SystemController e delega os métodos de votação para as entidades correspondentes. No SystemController também foi adicionado o método votarPlenario, que é responsável por fazer validações por delegar a responsabilidade de votação dos projetos de lei.

Caso 8:
No caso 8, pede-se que seja feita a exibição da tramitação de um projeto de lei. Para isso, decidimos adicionar um ArrayList de String na classe abstrata ProjetoDeLei. Pelo uso da Herança apenas foi preciso que esse atributo fosse definido na classe abstrata. Dessa forma, sempre que um projeto de lei avança para um novo local de votação, adicionamos o local em questão no ArrayList de tramitação. Feito isso, implementamos o método exibirTramitacao na classe ProjetoDeLei. No SystemController há o exibirTramitacao, que delega a responsabilidade para a entidade ProjetosDeLeiController. Este, por sua vez, procura o código do projeto retornando sua tramitação.

Caso 9:
No caso 9, pede-se que seja retornada a proposta legislativa de maior interesse de uma pessoa. Para isso, adotamos o uso do padrão Strategy, criando a interface EstrategiaOrdenacao. Existem três estratégias de ordenação possíveis, sendo elas por Conclusao, Aprovacao e Constitucional. Para isso, criamos as classes OrdenacaoAprovacao, OrdenacaoConclusao e OrdenacaoConstitucional que implementam a interface EstrategiaOrdenacao. Além disso, também criamos a classe OrdenacaoIdade que, em casos de empate, onde duas leis são igualmente interessantes à pessoa, fica responsável por decidir qual projeto de lei será retornado ao usuário com base no seu ano de cadastro. Em caso de um novo empate, o código do primeiro projeto de lei cadastrada no sistema é retornado. Alteramos a entidade Pessoa que passou a ter um atributo EstrategiaOrdenacao do tipo EstrategiaOrdenacao. Dessa forma, no momento de cadastro de uma pessoa, a estratégia padrão de ordenação definida é a OrdenacaoConstitucional.

A fim de facilitar o uso dos comparators, criamos a classe LeiComparator, que encapsula os atributos codigo, qntdInteresses, aprovacoes, TipoDeLei, tramitacao e dataDeCadastro. Sempre que há pelo menos um interesse em comum da pessoa em questão com um projeto de lei, um nova lei do tipo LeiComparator é criada. Com o objetivo de encapsular a lógica de ordenação, o SystemController recebe as requisições da Facade e delega para a classe ProjetosDeLeiController a responsabilidade de retornar a lei de maior interesse a uma pessoa. Na classe ProjetosDeLeiController, criamos o método pegarPropostaRelacionada que retorna o código da lei mais interessante à uma pessoa.

Uma pessoa pode mudar a estrategia de ordenação. Para isso, criamos o método configurarEstrategiaPropostaRelacionada no PessoaController que recebe o DNI de uma pessoa e a nome da nova estratégia de ordenação. A partir do nome passado, a nova estratégia é criada e passada para a pessoa em questão com o método setEstrategia.

Caso 10:
No caso 10, pede-se que seja feita persistência de dados do sistema em arquivos. Para isso, criamos a classe Dados que é responsável por encapsular a lógica de salvamento de arquivos, além de armazenar todas as coleções do sistema salvando-as na pasta (“dados”). Cada coleção é salva em um arquivo diferente no formato txt. Para fazer o uso desses dados, foi definido o atributo dados do tipo Dados nos controladores gerenciadores de entidades, onde todas as operações de cadastro no sistema, bem como consultas são feitos

na entidade Dados. A classe Dados possui, também, métodos que operam nas coleções, bem como os métodos limparSistema, salvarSistema e carregarSistema. Para o funcionamento desses métodos, foi necessário implementar a interface Serializable em todas as entidades do sistema. Para o método limparSistema, as coleções são novamente instanciadas e conteúdo dos arquivos de dados é apagado. Para carregar o sistema, utilizamos o método carregarObjeto que tenta carregar um Objeto a partir de um arquivo. Se o arquivo estiver vazio, o método retorna Null e a coleção é iniciada sem dados. Caso contrário, o objeto é retornado e a coleção iniciada com os dados presentes no arquivo. Para se fazer uso dessa classe, definimos o atributo dados do tipo Dados no SystemController. No construtor de SystemController um novo objeto do tipo Dados é instanciado e passado para os controladores mais internos.

O que foi feito por cada um:

Augusto: Implementação dos casos de uso 1, 5 e 10. Relatório do projeto, documentação em geral e testes de unidade.

Renildo: Implementação dos casos de uso 2, 3, 7 e 8. Diagrama de classes, documentação em geral e testes de unidade.

Wander: Implementação dos casos de uso 4 e 6. Javadoc, documentação em geral e testes de unidade.

O caso de uso 9 foi feito em equipe, pois demandava uma atenção a mais na decisão de como seria feita a implementação.