Universidade Federal de Campina Grande (UFCG)
Graduação em Ciência da Computação
Relatório de Projeto LP2 E-Câmara Organizada (E-CO)
Professora responsável Livia Campos
Monitor Responsável Douglas Pereira de Lima
Equipe:Andre Luis Souza de Andrade Santana (118210116), Caio Jose dos Santos Arruda (118210844), João Vitor Patricio Romão (118211058), Sheila Maria Mendes Paiva (118210186)

Campina Grande, 2019.

Design Geral:

O design geral do nosso projeto foi adotado para proporcionar um melhor desenvolvimento entre as diferentes classes do nosso sistema, elaborando ideias que diminuem o acoplamento. Decidimos usar um controller geral, que desenvolve as atividades para cada classe menor. As classes também apresentam um domínio próprio, para aumentar o nível de coesão do sistema.

Com relação a exceções, utilizamos uma hierarquia que leve em consideração do tipo de erro, instanciando as exceções no controle geral e verificando certas exceções diretamente no próprio método. Assim, temos exceções para atributos inválidos (vazio ou nulo), tanto de criação como de incremento de dados. Vale ressaltar que todas as exceções estão encapsuladas em pacotes menores.

Para a criação de determinados objetos, empregamos, em geral, composição de classes e em outros casos como, por exemplo, pessoa e deputado, adotamos um padrão que define uma interface para criar o objeto, mas permite às classes maiores decidirem qual classe instanciar.

As próximas seções detalham a implementação de cada caso.

É importante, também, evidenciar que procuramos sempre debater juntos antes de implementar cada caso e decidir juntos o design utilizado, de início sentimos dificuldades, mas conseguimos decidir juntos, dividimos cada parte por casos, deixando cada um com uma especificação, e, assim, foi a divisão do nosso projeto… na Parte 3, debatemos e sentimos algumas dificuldades e houve uma divisão diferente, na qual um participante do grupo ficou encarregado dessa parte, por obter tal conhecimento e deixamos os casos de unidade e documentação para os outros, inclusive o relatório também.

Caso 1:

O caso 1 pede que seja criado uma entidade que representa diferentes pessoas que poderão se cadastrar no E-CO, seja para realizar consultas ou para depois se tornarem políticos nesse sistema. Toda pessoa é identificada unicamente por um número chamado de DNI (Documento Nacional de Identificação), têm um nome, estado de origem e, opcionalmente, partido, sendo possível, também, cadastrar uma lista de interesses de uma pessoa. Para representar essa entidade, optamos por criar dois construtores dentro da classe Pessoa e uma "ControlePessoa" para manipular os dados de pessoa, na qual há uma coleção que armazena todos os funcionários. Essa coleção é um mapa, onde a chave é a identificação única (DNI) para cada pessoa. Decidimos criar uma interface (Cargo) para definir o cargo de pessoa dentro do sistema, além da definição do método "cadastrarPessoa(String nome, String dni, String estado, String interesses)" ou "cadastrarPessoa(String nome, String dni, String estado, String interesses, String partido)" que permite a adição de uma pessoa ao sistema, dessa forma, utiliza-se polimorfismo de sobrecarga, pois definimos a assinatura do método que possui implementações diferentes.

Caso 2:

No caso de uso 2 é solicitado o cadastro de um deputado sendo realizado automaticamente a partir dos dados de uma pessoa. Apenas é possível cadastrar deputados a partir de pessoas com partido. Todo deputado tem uma data de início na vida pública e têm uma quantidade de leis aprovadas na câmara. Criamos uma classe Deputado e utilizamos a interface (Cargo) criada, para caso houvesse mais de um cargo a ser desenvolvido, para atualizar as informações de pessoa no sistema. Foi criado um método chamado "cadastrarDeputado(String DNI, String dataDeInicio)" na classe "ControlePessoa", já que deputado é uma pessoa anteriormente adicionada ao E-CO, cadastrando o deputado ao sistema.

Caso 3:

No caso de uso 3, para implementar este caso não foi necessário implementar novas entidades, apenas adicionar funcionalidades nas que foram criadas no caso de uso 1. Uma pessoa têm a representação gerada a partir de seus atributos básicos e se ela foi também

cadastrada como deputada ou não. Foi criado um método chamado “exibirPessoa(String DNI)” para exibir os atributos básicos de uma pessoa ou um deputado dependendo do cadastrado no sistema.

Caso 4:

No caso de uso 4 o sistema deve permitir o cadastro de partidos na base governista e a exibição dos partidos da base governista (em ordem lexicográfica). Para essa implementação, optamos por criar um Controle de Partidos, onde o mesmo possui a responsabilidade de controlar os dados envolvendo partido, no caso, o método de cadastrar partidos e exibir base, para isso, este controle está associado a uma classe Partido, que possui as informações necessárias de um Partido. Implementamos o método “cadastrarPartido()”, para que seja possível armazenar o partido em um ArrayList a partir de sua sigla/nome. Para o método “exibirBase()” no qual devemos retornar uma string representando os partidos, utilizamos o Comparator para que fosse possível fazer as devidas comparações e deixar em ordem lexicográfica.

Caso 5:

No caso de uso 5 o sistema deve permitir o cadastro de uma comissão definindo um interesse (tema da comissão) e uma lista de deputados participantes (códigos do DNI). Para implementar esse caso, decidimos criar a classe “ControleComissao”, responsável por manipular os dados envolvendo comissão. Nessa classe temos um mapa de comissão, onde tem-se o tema da comissão e a lista de políticos relacionados a ela. Na classe Comissão utilizamos dois métodos acessores(get), um para que seja possível o controle de comissão ter acesso ao tema da comissão e o outro para que seja possível acessar a lista de políticos relacionados a uma determinada comissão . Também definimos que uma comissão é igual a outra quando possuem o mesmo interesse .

Caso 6:

No caso de uso 6 o sistema deve permitir cadastrar e exibir proposta legislativa. Existem 3 diferentes tipos de propostas legislativas, existem os projetos de lei (PL), criados na Câmara e no Senado e que estruturam normas e regramentos considerados ordinários (do dia a dia), temos também os projetos de lei complementar (PLP), que são projetos de leis que complementam a constituição e também temos os projetos de emendas constitucionais (PEC) que emendam ou alteram a constituição (exceto as cláusulas pétreas). Para implementarmos essa funcionalidade utilizamos herança, pois esses 3 tipos de projetos possuem características em comum, por isso, temos a classe Projeto como a classe Pai e as classes “Pl”, “Plp” e “Pec” como filhas. Além disso criamos a classe “ControleProjetos”, responsável pela manipulação de dados envolvendo projetos, a partir disso, temos 3 métodos para o cadastro de projetos, no caso “cadastrarPL()”, “cadastrarPLP()” , “cadastrarPEC()”, o cadastro é feito a partir das informações relacionadas ao determinado tipo de projeto e ao efetuar o cadastro, retorna-se o código que representa o determinado projeto. Todo projeto cadastrado, foi armazenado em um único mapa para melhor manipulação dos mesmos nos próximos casos. Também implementamos o método “exibirProjeto()”, que retorna a representação textual de um projeto a partir de seu código(identificador).

Caso 7:

No caso de uso 7 o E-Camara deve simular as votações nas comissões e no plenário de acordo com: os políticos presentes nas comissões, o tipo de projeto e o apoio à proposta. Foi implementada a classe “Votacao” que possui o status e o local de uma determinada votação, criamos métodos acessores e modificadores para a realização de processos posteriores, também nessa classe foi criado um método chamado “realizaVotacao”, que realiza a votação de acordo com o tipo informado (comissão ou PEC). Para este caso de uso também foi criada a classe “ControleVotacao”, responsável por realizar as votações e atribuir status de aprovado ou reprovado para determinada votação, além disso, esta classe guarda as votações realizadas identificadas pelo seu código.

Caso 8:

No caso de uso 8 foi pedido que o sistema exibisse as tramitações (diferentes tipos de resultados de votações em comissões e plenários nas quais a proposta passou) referentes às propostas votadas. Foi implementado o método “exibirTramitacao” na classe

“ControleVotacao” que recebe o código da votação cadastrada, acessa essa votação e formula uma string que representa a tramitação da votação.

Caso 9:

O caso de uso 9 pedia que se fosse implementado um mecanismo de busca capaz de identificar automaticamente a proposta legislativa de maior interesse de uma pessoa, facilitando a atuação em propostas específicas. Para a realização das especificações pedidas foram criados métodos que permitiram selecionar e organizar as estratégias necessárias para cada caso, sendo estes métodos: “configurarEstrategiaPropostaRelacionada:void”, que recebe o DNI do deputado e a estratégia que será adotada, e configura a estratégia desse deputado, também foi criado “contagemProjInteressesDep: ArrayList<String>” que recebe o DNI do deputado, conta os interesses em comum do deputado referente ao dni passado com os projetos em tramitação e aloca esses interesses em um ArrayList. Para ordenar as propostas de um deputado, foi criado o “constitucionalSort: ArrayList<String>”, que recebe a lista de projetos formada no método anterior e o ordena de acordo com a estratégia constitucional. Na mesma lógica de “constitucionalSort” foi criado “conclusaoSort: ArrayList<String>”, que ordena o ArrayList de códigos dos projetos de acordo com a estratégia Conclusão. O método “propostaRelacionada: String” foi criado com a responsabilidade de exibir a proposta mais relacionada com o deputado do DNI que é passado como parâmetro respeitando critérios de desempate. Também foi criado na classe “ControleGeral” o método “pegarPropostaRelacionada: String”, que exibe na forma de String a proposta mais relacionada com o deputado de DNI recebido.