Design geral:

O design geral do nosso projeto foi escolhido com o objetivo de otimizar a integração entre as classes que fazem parte do sistema, para diminuir o acoplamento criamos um controller geral, que delega atividades para as camadas mais inferiores, que por sua vez também possuem seus próprios controllers.

Para a criação de novos objetos foram utilizadas factories. Para reaproveitamento de tipo utilizamos o conceito de herança com a criação de classes abstratas e de classes que estendem as classes abstratas.

As próximas seções detalham a implementação em cada caso.

Caso 1:

O caso 1 pede que seja possível cadastrar e armazenar pessoas no sistema, para isso criamos a classe Pessoa que contem todos os atributos necessários de uma pessoa, sendo opcional a filiação da pessoa a um partido, para isso foi utilizada a sobrecarga na classe sendo criados dois construtores de pessoa, um para pessoas filiadas a partidos e outra para pessoas que não estejam filiadas.

Para armazenar as pessoas criamos a classe ControllerPessoa que contém um mapa que tem como chave o DNI(numero de identificação) da pessoa e armazena objetos do tipo pessoa.

Caso 2:

O caso 2 pede para que seja possível cadastrar um deputado a partir de uma pessoa existente, para isso foi criada a classe Deputado que contem os atributos data de inicio e leis aprovadas. Dentro da classe pessoa foi criado um atributo “deputado” do tipo Deputado que inicia como Null, caso a pessoa se torne um deputado, é passada a data de inicio para o método virouDeputado() que cria um novo objeto do tipo Deputado e atribui a “deputado”.

Caso 3:

O caso 3 pede para que seja possível exibir a representação de uma pessoa gerada a partir de seus atributos básicos, para isso criamos o método exibirPessoa() que monta uma String com os atributos de uma pessoa, o método verifica se a pessoa é ou não um deputado e exibe a representação de acordo com esta verificação.

Caso 4:

Ocaso 4 pede para que seja possível cadastrar partidos na base governista e exibir os partidos da base, para isso criamos o ControllerComissoes que controla as comissões e os partidos,

nele está contido uma lista que armazena os partidos que são cadastrados pelo método cadastrarPartido(String partido) que recebe o nome do partido e adiciona-o a lista.

Para a exibição dos partidos da base, criamos o método exibirBase() que ordena a lista de partidos e retorna uma String contendo todos os partidos separados por virgula e em ordem lexicografia.

Caso 5:

O caso 5 pede para que seja possível cadastrar comissões no sistema, para isso criamos a classe Comissao, que constrói uma nova comissão a partir de um tema e um String com os dni dos políticos participantes da comissão. Para armazenar as comissões cadastradas foi criado um mapa de comissões que tem como chave o tema de cada

comissão e armazena objetos do tipo Comissao, e o método cadastraComissao(String tema, String políticos), que cria o objeto Comissao e adiciona ao mapa de comissões.

Caso 6:

O caso 6 pede para que seja possível cadastrar e exibir propostas legislativas, sendo estas propostas de diferentes tipos mas com atributos em comum, para isso utilizamos a herança, criando a classe abstrata Projeto que contem atributos e métodos em comum a todas as propostas, para as propostas do tipo PL foi criada a classe PL que herda da classe Projeto, como as propostas do tipo PLP e PEC são consideradas constitucionais, foi criada a classe abstrata ProjetosConstitucionais que herda da classe Projeto e criamos as classes PLP e PEC que herdam da classe ProjetosConstitucionais.

Para armazenar os projetos cadastrados e controlar as ações executadas sobre eles, criamos a classe ControllerProjeto, que contem um mapa de projetos que tem como chave uma String que representa o projeto e armazena objetos do tipo Projeto utilizando-se do polimorfismo para armazenar objetos do tipo PL, PLP e PEC que são extensões da classe Projeto, os projetos são adicionados ao mapa a partir dos métodos cadastrarPL, cadastrarPLP e cadastrarPEC que criam objetos de seus respectivos tipos e adicionam os mesmos ao mapa de projetos.

Para exibir as propostas,criamos um método toString() que mosta uma String com as informações do projeto dentro de cada tipo de projeto e criamos o método exibirProjeto(String codigo) dentro do ControllerProjeto que chama o método toString() de cada projeto.

Caso 7:

O caso 7 pede para que seja possível que um projeto de lei seja votado em comissões ou no plenário, para isso criamos os métodos votarPlenario dentro de PL, PLP e PEC e o método votarComissao dentro de PL e de ProjetosConstitucionais, esses métodos retornam um booleano, true caso o projeto seja aprovado na votação ou false caso seja rejeitado. Os votos são contabilizados pelo método contaVotos, criado na classe projeto. Devido às verificações de exceções que necessitavam de dados existentes nos outros controllers, criamos estes mesmos métodos dentro das classes controllerProjeto e controllerGeral, para evitar a quebra do encapsulamento.

Caso 8:

O caso 8 pede para que seja possível exibir a tramitação de um projeto, para isso criamos o método exibirTramitacao na classe controllerProjeto que acessa o tributo “tramitacao” de um projeto. O atributo tramitacao é atualizado sempre que o projeto de lei é votado em comissão ou no plenário.

Caso 9:

O caso 9 pede para que seja possível exibir uma proposta de lei, ainda não finalizada, que tenho relação com os interesses da pessoa. Caso existam mais de um projeto que tenha essa relação, o critério adotado para selecionar qual será retornada pode variar. Por padrão a pessoa sempre começa dando prioridade à forma constitucional, mas ela pode ser alterada para a forma conclusão e para a forma a aprovação.

Para implementar esta funcionalidade foi criada uma interface que é implementada por três classes. Cada uma delas vai selecionar de forma diferente qual será o projeto retornado, com o uso do método *min* de *collections.* Para definir qual é o valor mínimo foram criadas outras três classes que selecionam de forma diferente o fator principal de seleção e caso haja empate, o critério de desempate é igual para as três.

Caso 10:

No caso de uso 10 foi requisitado que a equipe adicionasse a funcionalidade de salvar o estado do sistema para persistência de todos os dados após o encerramento. Para isso, foi necessário implementar a interface serializable em todas as classes que tivessem algum dado para armazenar, que no caso foram as classes abaixo dos Controllers. Foi feita a criação da pasta “/arquivos/” para armazenar os arquivos contendo o estado do sistema. Foram criados os métodos gravarSistema, carregarSistema e limparSistema na facade para atender o que foi requisitado.

No método gravarSistema, o controllerGeral delega para os controllers a função de gravar cada dado que estiverem em seus atributos, como no controllerPessoa que possui um mapa de pessoas. No comando .writeObject() o sistema é salvo em arquivos .arq . Caso já existam dados de sistema gravados, os dados são sobrescritos pelos novos.

No método carregarSistema temos um trycatch, onde no escopo do comando é verificado se há algum sistema já carregado na pasta indicada (“/arquivos/”), se não houver então o sistema é iniciado sem nenhum dado salvo. Caso haja um arquivo de sistema para ser carregado, cada Controller carrega os dados em seus atributos pelo comando .readObject().

No método limparSistema, é usado o comando .clear nos atributos que contém os dados do sistema para limpar as Collections mesmo que não existam dados nas Collections, logo após temos também um trycatch para verificar se existem dados do sistema para limpar, caso existam, o comando .delete é usado em cada arquivo para deleta-los, caso não existam o método para por ai.