Relatório de Projeto L P2 2019.2

Design geral:

O design de nosso projeto foi pensado com intuito de ser extremamente funcional, possuir o acoplamento apenas quando necessário, entre entidades, e possuir camadas de abstração que sejam úteis para criar soluções aos problemas da especificação do projeto, podemos exemplificar com a entidade AssociaEmPesquisa, que atua por composição na classe Pesquisa, e pode associar objetivo,problema,pesquisador à pesquisa, podemos citar também como exemplo a classe OrdemAtividade, que também atua por composição em Atividade, e ao receber uma informação do usuário, pode acrescentar/remover/ atividades a uma determinada ordem, e contar quantas atividades faltam ser executadas na ordem desejada pelo usuário.
Quando nos deparamos com a situação de atributos imutáveis, decidimos utilizar Enumerators, com a intenção de deixar o código mais elegante, o uso de tais entidades fica visível nas classes StatusItem, NivelRisco, entre outras classes de nosso sistema.
Quando observamos que determinada entidade de nosso sistema, começou a apresentar um padrão dinâmico, adotamos o padrão Strategy, como exemplo tem a classe Pesquisador, quando um pesquisador necessita ser especializado, para resolvermos optamos pela criação de uma Interface denominada Especialidade, que possui o método de especializar um pesquisador, criando assim, um Aluno ou Estudante, e que por sua vez, está contida como objeto dentro da entidade Pesquisador.
Com relação as exceptions, optamos pela criação de uma entidade chamada de Validação,onde todas as classes que são necessárias se fazer algum tipo de verificação, possuem um objeto de Validação, que por sua vez, possui todos os métodos necessários para lançar exceções no sistema.
As próximas seções detalham mais especificadamente como se comportou nosso sistema de acordo com as necessidades de cada unidade.

Caso 1:
O caso 1 trata de criar uma entidade que represente as pesquisas no sistema. Cada pesquisa é composta por uma descrição, pelos campos de nteresse e por um código gerado a partir das três primeiras letras do campo de interesse seguidas de um inteiro. Para gerar esse código foi criado um método chamado geraCodigoDePesquisa(código: String, i: int): String, que recebe as 3 primeiras letras do campo de interesse e um inteiro (começando com 1), a partir desses dados, é verificado se já existe alguma pesquisa com o código gerado, se existir o método se invoca de forma recursiva com o i tomando valor de i+1, e segue até encontrar um código válido. O gerenciamento das pesquisas é feito por uma classe chamada ControllerPesquisa. Nesta classe, há uma coleção que armazena todas as pesquisa. Essa coleção é um mapa, no qual as chaves são os códigos, únicos para cada pesquisas.

Caso 2:

O caso de uso 2 se trata da criação das entidades que representam pesquisadores no sistema Psquiza. Cada pesquisador possui nome, função, biografia, email, URL de foto e especialidade. Para que fosse possível tratar todas as especialidades (Professor, Aluno ou Externo) de uma forma parecida e em uma mesma estrutura de dados, foi utilizada uma interface Especialidade. A partir dela, é possível implementar Professor ou Aluno, que possuem métodos e atributos únicos para cada tipo. Para gerenciar os pesquisadores, foi criada a classe ControllerPesquisador, responsável por criar, armazenar e editar os pesquisadores. Nela, foi utilizado um mapa para guardar os objetos pesquisadores criados, através de uma chave que representa seu email e o identifica unicamente.

Caso 3:

Para o caso 3, foi solicitado pela especificação à capacidade do sistema cadastrar, apagar, e exibir, problemas e objetivos, partindo desta necessidade em nosso sistema, optamos pela criação de quatro novas entidades, sendo elas as classes Problema e Objetivo que são respectivamente representações de um problema e de um objetivo, onde um problema possui descrição, viabilidade e um id que é gerado automaticamente seguindo a ordem de cadastro P+1, P+2, P+3... E um objetivo possui além das características de problema, um valor para a aderência e um tipo, que varia entre geral ou especifico. As duas outras entidades são os Controllers de problema e objetivo, que possuem uma coleção (HashMap), que armazena todos os problemas do sistema para o Controller de problema ou todos os objetivos do sistema para o Controller de objetivo, e também, são responsáveis por cadastrar novos, apagar e exibir problemas/objetivos de acordo com a informação passada pelo usuário.

Caso 4:
O caso de uso 4 informa que para atingir um objetivo do caso de uso 3, deve-se descrever e planejar atividades metodológicas. Cada atividade deve ter uma descrição, uma duração, resultados esperados e um risco associado. Implementamos um controller que se encarrega de fazer a comunicação entre a Facade e a Atividade, assim como entidades que estão ligadas a Atividade. Optamos por armazenar a descrição em uma String, e criar entidades para Resultado, Risco e Item, pois essas últimas têm atributos específicos de cada uma. Resultado possui como atributos descrição, uma String, e id, um inteiro. Cada objeto Atividade tem seu mapa de resultados, sendo assim, também contém métodos que trabalham com atributos dessa entidade. Risco contém sua descrição, em uma String, e um atributo que define o seu nível, para isso utilizamos uma classe Enum NivelRisco. Item possui id e duração, armazenados como inteiros,o nome, armazenado em uma String e o seu status, sendo necessário a implementação de uma classe Enum StatusItem, que define se ele está realizado ou pendente.

Caso 5:
O caso de uso 5 consiste em associar um Problema e Objetivos a uma Pesquisa. Para isso, fizemos com que o ControllerPesquisa tenha os mapas que outras entidades do sistema são armazenadas, com isso, tal controller acessa o Problema ou o Objetivo a ser associado

e passa o objeto para a classe Pesquisa. Na classe Pesquisa criamos uma composição responsável por armazenar as associações e efetuá-las. Todos os métodos sobre associação em Pesquisa estão nessa composição. As associações são armazenadas com o próprio objeto a ser associado, no caso a composição da associação em Pesquisa tem um atributo do tipo Problema, pois cada Pesquisa pode ter apenas um problema associado e um ArrayList do tipo Objetivo, pois é permitido uma pesquisa ter vários objetivos.

Caso 6:

No caso de uso 6, surgiu a necessidade de associarmos/desassociarmos um pesquisador a determinada pesquisa, com a condição de ambos estarem cadastrados no sistema. Para realizar tal solicitação, optamos por passar para pesquisa a Collection detentora das informações sobre os pesquisadores, que por sua vez passou a possuir informações sobre Pesquisador e Pesquisa, podendo assim realizar a associação. Também se fez necessário a especialização de um Pesquisador, seja ela como estudante ou professor, para realizar tal tarefa, optou-se pela criação de uma interface, denominada Especialidade e por criar duas entidades chamadas de Estudante e Professor respectivamente, que implementam a interface Estudante contém o semestre em que o aluno está cursando e seu IEA, e Professor contém a formação,unidade e data. Ambos os objetos das entidades, passam a existir no sistema por meio de uma composição, ou seja, na entidade Pesquisador, temos um objeto de Especialidade, que quando invocado, cria um objeto de Aluno ou Professor, dependendo das informações passadas pelo usuário

Caso 7:

Para o caso 7, foi implementada a função de associar e desassociar atividades a pesquisas. Para tal, utilizamos os mapas presentes em ControllerPesquisa para que pudéssemos acessar as atividades e pesquisas cadastradas simultaneamente. Para realizar a associação, é feito o uso de um objeto do tipo AssociacaoEmPesquisa dentro da classe Pesquisa, que realiza tarefas de associação/desassociação. Além, disso, dentro dela há uma estrutura de dados (ArrayList) que armazena as atividades associadas a ela, não podendo ter atividades repetidas em uma mesma pesquisa.

.

Caso 8:
Para o caso de uso 8 foi pedido que o sistema tivesse a possibilidade de realizar busca de um termo, nas principais entidades do sistema, de forma mais específica a busca deve ser feita nas descrições e campos de interesse das pesquisas, nas biografias dos pesquisadores, nas descrições dos problemas e dos objetivos e nas descrições e descrições de risco das atividades. Além de listar todos os resultados também foi necessário implementar métodos que possibilitam contar a quantidade de resultados encontrados e retornar apenas um resultado a partir da sua posição. Para poder ordenar a

listagem de busca foi necessário implementar um comparator, que ordena a partir dos códigos das entidades de forma anti lexicográfica .

Caso 9:

Para o caso de uso 9, notamos que a necessidade do sistema em geral, era de ordenar a execução das atividades , porém também percebemos que a entidade que detinha as informações sobre atividade, já executava diversificadas funções em nosso sistema, por isso optamos pela criação de uma nova entidade chamada de OrdemAtividade, que possui uma Collection (HashMap) com as atividades cadastradas no sistema, sendo assim o expert, que poderá definir a ordem das atividades que vão ser executadas, podendo adicionar uma atividade como precedente ou subseqüente de outra atividade, remover, contar quantas atividades faltam ser executadas a partir de uma atividade fornecida pelo usuário, e informar a atividade com maior risco de execução ao usuário.

Caso 10:
No caso 10 foi pedido a implementação de um sistema de sugestão de próxima atividade a ser realizada em uma atividade. Para isso, foram criados comparators de Atividade (ComparatorAtividadePorDuracao, ComparatorAtividadePorPendencias e ComparatorAtividadePorRisco) para que fosse possível verificar qual de duas atividades possuíam mais/menos atributos de acordo com a estratégia utilizada. Além disso, por padrão, o compareTo(a1,a2) de Atividade é configurado para verificar se a1 possui itens pendentes mais antigos que a2. As estratégias de sugestão de próxima Atividade são todas armazenadas na Enum SugestaoProximaAtividade.

Caso 11:
No caso de uso 11 foi pedida a funcionalidade de exportar um resumo e os resultados de uma pesquisa, para um arquivo de texto que deve ficar na raiz do projeto. Por se tratar de uma lógica muito específica e complexa, foi feito composição na classe Pesquisa. Tanto para gravar o resumo, quanto para gravar os resultados foi utilizado um File que representa o resumo ou os resultados, com o nome do arquivo sendo “_CodigoDaPesquisa.txt” para os resumos e “CodigoDaPesquisa-Resultados.txt” para os resultados e um FilleWritter. Com isso teremos um stream de texto, no qual a sua fonte é o sistema, o destino é o arquivo na raiz do projeto. Para ordenar todos os objetivos e as atividades foi preciso fazer nas duas classes métodos compareTo que usam a ordem de cadastro para ordenar. Foram criados métodos em Facade, ControllerPesquisa e em Pesquisa, que apenas delegam para o método criado em RelatorioPesquisa. Este último, então, escreve o resumo ou os resultados, caso já exista algum arquivo com esse nome, o programa deve sobrescrever.

Caso 12:
O caso de uso 12 consiste em fazer com que o sistema possa ser encerrado sem perder os dados salvos. Primeiramente foi necessário implementar a interface serializable em todas as classes que precisavam ser armazenadas para a persistência do sistema. Também foi

criada a pasta “/sistema_serializado” que armazena todos os arquivos .txt nos quais dados do sistema são gravados para serem recuperados depois. Para isso, utilizamos o ControllerPersistencia, classe responsável por receber todos os outros controllers que tenham informações salvas. Nessa classe, temos métodos para salvar os dados e depois carregá-los. Os métodos de salvar e carregar funcionam de forma parecida. Os responsáveis por salvar utilizam uma stream do tipo ObjectOutputStream, onde a fonte é o sistema e o arquivo .txt é o destino, inserindo dados do tipo object em tal arquivo, através do comando .writeObject(). Ao carregar o sistema, utilizamos outros métodos também deste controller, a fim de ler os arquivos .txt. Nesses métodos utilizamos uma stream do tipo ObjectInputStream que lê o que estava armazenado nos arquivos da pasta “/sistema_serializado” e passa para o sistema. Para que o sistema faça uso dessas informações optamos por fazer o cast desse objeto do tipo ObjectInputStream para o tipo do controller que foi salvo na operação salvar. Após isso, os métodos de carregar retornam esses controllers para que, na Facade, os controllers que antes foram instanciados como novos, recebam os dados gravados anteriormente.

Link para o repositório no GitHub:
https://github.com/CaetanoAlbuquerque/projetoLp2