Relatório de Projeto LP2 2019.2

Design geral:
O design geral do nosso sistema foi pensado de forma a permitir, além das funcionalidades em si, expansões com o menor atrito possível e a legibilidade do código por outros programadores. Tivemos como base os padrões de responsabilidade Grasp e Strategy. Dessa forma, optamos, por um controller geral, que cuida das associações entre as diferentes entidades do sistema. Por fim, buscamos extrair o máximo potencial da API de Collections de Java, procurando sempre fazermos uso das estruturas de dados mais adequadas em cada situação.

Caso 1:
O caso de uso 1 pede para criarmos uma Pesquisa com descrição e campo de interesse, no qual elas podem ser desativadas e ativadas, alteradas quando ativas, exibidas e ver se uma determinada pesquisa é ativa ou não. Sendo assim, optamos por criar uma classe Pesquisa, que guarda as informações dela, e um controller de Pesquisas, em que toda vez que é cadastrado uma nova pesquisa, é gerado um Id que identifica a Pesquisa, e é colocado num Map de Pesquisas, como sendo a chave e é associado a um objeto Pesquisa que é criado simultaneamente, com as características passadas da Pesquisa. Assim, pode se chamar os outros métodos através desse ID, desativando, ativando, alterando, exibindo ou vendo se aquela determinada pesquisa é ativa ou não.

Caso 2:
O caso de uso 2 descreve como devemos lidar com pesquisadores em nosso sistema. Inicialmente podemos cadastrar, alterar, desativar/ativar, exibir e consultar o status de ativação. Para o cumprimento da especificação, utilizamos de uma classe Pesquisador, com o intuito de encapsular atributos - nome, função, biografia, email e foto - e métodos que trabalham em conjunto. Além disso, criamos uma entidade PesquisadorController, a fim de armazenar e gerenciar vários Pesquisadores.

Caso 3:
O caso de uso 3 descreve como os problemas e os objetivos de uma pesquisa devem ser. Foi criado uma classe Problema encapsulando os atributos código, descrição e viabilidade. Enquanto que para os objetivos foi criado a classe Objetivo, possuindo os atributos código, tipo (podendo ser geral ou específico), descrição e valor. O sistema deverá permitir cadastrar, apagar e exibir problemas e objetivos. Para o cadastro foram criados os métodos cadastrarProblema() e cadastrarObjetivo() que recebem os atributos básicos das duas entidades. Para apagar foram criados os métodos apagarProblema() e apagarObjetivo() que recebem como parâmetro o código do problema e do objetivo. Para exibir as

entidades foram criados os métodos exibirProblema() e exibirObjetivo() que também recebem como parâmetro o código do problema e do objetivo. E por fim, foram criadas as classes ProblemaController e ObjetivoController, com o objetivo de, como o próprio nome já diz, controlar e armazenar as diversas instâncias das classes Problema e Objetivo.

Caso 4:

O caso de uso 4, para o auxílio com objetivos da pesquisa, descreve e planeja atividades para alcançar determinados resultados.

Cada atividade possui uma descrição, uma duração, um risco para sua atuação (BAIXO, MÉDIO ou ALTO) e uma descrição que determina o risco da atividade.

Os resultados esperados da atividade são compostos por uma coleção de itens (ArrayList) - em que podem ser considerados como “REALIZADO” ou “PENDENTE”. Os itens possuem uma ordem de cadastro, e os resultados esperados não possuem itens repetidos, logo, foi feito uma verificação se a atividade já possui o item antes de cadastrá-lo na coleção.

Quando a atividade é cadastrada é gerado um código automaticamente, sendo este: “A” + quantidade de atividades que já foram cadastrados.

Quando uma atividade é excluída, nada é alterado nos próximos códigos de atividades cadastradas.

A representação de uma atividade é dada por sua descrição, nível de risco, descrição de risco, e seus itens - onde pode-se verificar quantos itens pendentes ou realizados os resultados esperados possuem.

Caso 5:

Pesquisas necessitam podem possuir um problema e vários objetivos para resolver determinada problemática, logo, o caso de uso 5 associa por meio dos id’s do problema e dos objetivos uma pesquisa com um único problema e vários objetivos. Um problema pode estar presente em várias pesquisas e um objetivo só pode estar em uma única pesquisa. A pesquisa possui a informação de seu problema associado, também possui uma coleção de objetivos (LinkedHashMap) à ser realizados. Podemos associar ou desassociar problema e objetivos de uma pesquisa. Podemos listar as pesquisas de três diferentes formas (PROBLEMA, OBJETIVOS, PESQUISA), logo, foi implementado três diferentes classes implementando um comparator<Pesquisa> para serem instanciadas quando for necessário ordenar para exibição.

A ordenação PROBLEMA é feita pelo maior id do problema da pesquisa, se a pesquisa não possuir problema a ordenação passa a ser pelo maior id da pesquisa.

A ordenação OBJETIVOS é feita pela maior quantidade de objetivos associados, caso empate, a ordenação é feita pelo objetivo de maior id, pesquisas sem objetivos são ordenadas pelo maior id.

A ordenação PESQUISA é feita pelo maior id da pesquisa.

Caso 6:
No caso de uso 6, foi pedido para que se pudesse associar um pesquisador a uma pesquisa, e para que fosse possível cadastrar detalhes da especificação(Professor, ALuno ou Externo) dos pesquisadores. Assim, nós escolhemos utilizar a estratégia do Strategy, no qual nós criamos um atributo que é uma classe chamada Especificação, essa classe Especificação tem outras três classes ligadas a ela através da estratégia de Herança, que são as classes: Professor, Aluno e Externo, dessa forma sempre que for cadastrar os detalhes do Pesquisador como um professor, aluno, é criado dentro desse atributo Especificação um novo professor ou um novo aluno, cada um com suas características particulares e seus métodos particulares, como por exemplo o método toString() que para cada um é diferente, e toda vez que se quiser acessar esses dados particulares, é só entrar dentro do atributo especificação para pegá-los.

Caso 7:
O caso de uso 7 especifica que as pesquisas podem ter atividades associadas a ela, para isso, foi criado na entidade Pesquisa um atributo do tipo HashMap para armazenar as chaves, sendo cada chave o código da AtividadeMetodologica associada, e os valores, sendo cada valor um objeto da classe AtividadeMetodologica. Também é pedido que as atividades possam ser executadas, e para isso elas devem ser associadas a pelo menos uma pesquisa. Para isso varremos todas as pesquisas até encontrar a primeira ocorrência da atividade a ser executada, caso não ache nenhuma, é lançada a exceção “Atividade sem associações com pesquisas”. Para executar uma atividade, devemos executar um item cadastrado nela (passado sua posição por parâmetro) e torná-lo realizado, além de registrar a duração da execução daquele item. Para isso, buscamos o item no ArrayList de itens no objeto de AtividadeMetodologica pela posição passada por parâmetro e utilizamos um método da classe Item para mudar a situação do atributo estado para “REALIZADO”, além de alterar sua duração para a duração informada por parâmetro. É solicitado a criação de, exclusão e listagem de resultados nas atividades. Para isso criamos a entidade Resultado para representar um resultado no sistema. A entidade possui o atributo descrição e seus objetos são armazenados em um ArrayList na entidade AtividadeMetodologica. Para o cadastro de resultados foi necessário criar um método cadastraResultado() que recebe o código da atividade em que o resultado será criado, e a descrição do resultado. Para a exclusão foi criado o método removeResultado() que recebe o código da atividade em que o resultado está registrado e a posição do resultado. E para a listagem foi criado o método listarResultados() que recebe o código da atividade que estão os resultados a serem listados e retorna uma String contendo as representações dos resultados. E por fim, é pedido para informar a duração total de execução de uma atividade, para isso foi criado um método getDuracao() que recebe como parâmetro

o código da atividade a ser consultada a duração e retorna a soma da duração de todos os itens da atividade. Para isso foi necessário percorrer o ArrayList de itens e consultar sua duração, somando numa variável e retornando a soma.

Caso 8:

No caso de uso 8, nos é pedido para criar uma forma de buscar e exibir entidades cadastradas em nosso sistema, baseada em um uma palavra-chave que deve estar presente em atributos específicos destas entidades; exibir uma entidade associada a um índice - na ordem: Pesquisa, Pesquisador, Problema, Objetivo e Atividade -, e contar quantos resultados a busca obteve.

Nesse sentido, criamos duas interfaces: Buscador e Buscavel, visando, respectivamente, encapsular as entidades que fariam a busca no sistema e encapsular as entidades que poderiam ser encontradas pelos buscadores. Por fim, a entidade Busca, gerencia, armazena, ordena (usando a entidade BuscaComparator), gera a representação esperada nos métodos de busca fazendo reaproveitamento de tipo.

Caso 9:

No caso de uso 9, é pedido para que se possa definir uma ordem sugestiva das atividades, associando uma atividade a uma outra atividade subsequente, e esta por sua vez associada a outra subsequente e assim por diante, dessa forma, criando uma ordem de atividades, porém sem ser permitido criação de loops. Com isso, se tem outras funcionalidades, como retirar um subsequente de uma atividade, dependendo da ordem, dividindo ela em outras duas diferentes. Além disso, têm outras funcionalidades como contar quantas atividades vem depois daquela, pegar o Id de determinada atividade na ordem das atividades, ou pegar o Id da pesquisa com o maior risco na ordem, depois de determinada atividade. Assim, para o método de pegar o próximo e na validação se criava loop ao cadastrar uma atividades subsequente, optamos por utilizar recursividade, já que era a estratégia mais adequada visto que toda vez que se entrava numa atividade para pegar seu subsequente, tinha que entrar nesse subsequente para realizar a mesma coisa e assim por diante, até chegar na condição de parada. E nos outros dois métodos nós optamos por usar um While. Para implementar este caso não foi necessário implementar novas entidades, apenas adicionar funcionalidades nas que foram criadas no caso de uso 7.

Caso 10:

Almejando dar inteligência ao usuário final, a especificação do caso de uso 10 requer que o sistema seja capaz de fazer recomendações de próximas atividades a serem realizadas em uma pesquisa e configurar a estratégia de sugestão que será usada. Tendo isso em vista, a entidade Pesquisa ordena suas atividades que possuem pendências com base na estratégia escolhida pelo usuário, e retorna a

mais indicada. Para isso, Pesquisa usa os seguintes comparadores de Atividade: AtividadeMaiorDuracao, AtividadeMaiorRisco, AtividadeMaisAntiga e AtividadeMenosPendencias.

Caso 11:

Usuários externos podem precisar de um arquivo de texto com as informações de determinada pesquisa, logo, o caso de uso 11 possibilita a exibição das informações em ordem de cadastro como um resumo ou somente como resultados, onde no resumo é exibido: nome da pesquisa, pesquisadores, problema, objetivos e atividades (Codigo da pesquisa.txt) e nos resultados é exibido: nome da pesquisa e os resultados com descrição e itens (Codigo da pesquisa-Resultados.txt).
A cada execução que se pede para gravar um resumo ou resultados em um arquivo o mesmo é sobrescrito, portanto, todas as informações da pesquisa que antes foi gerada, é apagada e a nova representação da pesquisa entra no lugar.
Para a implementação foi necessário “perguntar” à todas as entidades relacionadas a pesquisa sua informação, assim, armazenando tudo em uma representação textual e no final sendo repassado à um arquivo com o nome do código da pesquisa.

Caso 12:

No caso de uso 12, foi exigido a persistência das entidades básicas do sistema: Pesquisa, Pesquisadores, Objetivos, Problemas e Atividades. Para salvar os dados, serializamos os atributos dos controllers em que os objetos dessas entidades estavam armazenados, e escrevemos em um arquivo .txt, em que cada controller salvou um arquivo para cada atributo seu. Cada controller tem uma pasta somente sua para salvar seus arquivos, assim separando os dados dos controllers e mantendo a organização. Para carregar os dados, desserializamos cada arquivo .txt de cada pasta dos controllers, recuperando assim as informações de cada atributo dos controllers, e logo em seguida atribuindo essas informações novamente para cada atributo, recuperando o estado dos controllers.