Relatório de Projeto LP2

Design geral:

O design geral escolhido pelo nosso grupo foi escolhido para viabilizar uma melhor integração entre as diferentes entidades do sistema. Sendo assim, com o objetivo de possibilitar um baixo acoplamento e uma alta coesão do programa decidimos criar camadas de abstração por meio do uso de repositórios. Nesse repositórios estão armazenadas todas as estruturas de dados dos sistema. Além disso, optamos por aumentar o nível de abstração criando um controller específico para as camadas menores para que assim, cada controller possa gerenciar a sua respectiva entidade.
Desse modo, quando é necessário utilizar alguma entidade, faz uma relação direta do controller dessa entidade com o seu repositório, por exemplo: PesquisaController se relaciona diretamente com Pesquisas Repositorios
As próximas seções detalham a implementação em cada caso.

Caso 1:
No de uso 1, é pedido que haja um manejo sobre as pesquisas. É necessário poder criar um pesquisa - formada por Descrição e um Campo de Interesse -, alterar uma pesquisa, ativa-la e desativá-la, exibir uma pesquisa e verificar se uma pesquisa está ativa. Para cuidar disso, foi criado um Repositório de Pesquisas (PesquisasRepositorio) , que possui um Map de Pesquisas (uma classe responsável por controlar a entrada e saída de Pesquisas no Map) e um PesquisaController, uma classe que possui um repositório de pesquisas e é responsável por manejar as requisições do usuário no sistema envolvam o uso de pesquisas. Sendo assim, quando o usuário, por exemplo, deseja cadastrar uma nova pesquisa, o controller tratada da requisição, constrói o objeto Pesquisa e a adiciona no PesquisasRepositorio.

Caso 2:

O caso de uso 2 é voltado para a manipulação de Pesquisadores, sendo estes compostos de nome, biografia, email, foto e função (podendo ser professor ou aluno). A partir do ponto que é necessário que sejam criados Pesquisadores que possam ser editados, ativados, desativados, exibidos e ter sua ativação checada, optou-se pela criação de um repositório de Pesquisadores (que possui um map de Pesquisadores) responsável pela entrada de pesquisadores no sistema e acesso a estes ao longo dele. Este repositório (PesqusiadoresRepositorio) terá suas informações manipuladas a partir de requisições do usuário em um controlador de Pesquisadores (PesquisadoresController).

Caso 3:
No caso de uso 3 é pedido que existam problemas e objetivos. Para isso , foi criado a classe “Objetivo” e a classe “Problema” . Entretanto, para o controle dessas duas classes foi optado por um “ObjetivoProblemaController”, isso , pois , objetivos e problemas são classes intrinsecamente ligadas em suas relações. O uso de caso exigia também um cadastro de problema ou de objetivo , o que foi realizado armazenando-os em um HashMap (por

possuírem identificação única) que por sua vez era armazenado num repositório com suas devidas funções padrões (buscar,exibir,apagar,adicionar). Ademais, deveria ser possível apagar um problema ou um objetivo ,assim como exibi-los o que não gerou nenhum problema extra justamente pelo fato de exibir e apagar já fazerem parte da funções padrões do repositório.

Caso 4:
O caso 4 pede que seja possível gerenciar atividades de modo que o sistema possa cadastrá-las, removê-las e exibi-las. Além disso, deve ser possível também cadastrar itens a uma determinada atividade, podendo estes serem pendentes ou realizados . Por fim, o programa pode contar quantos itens pendentes e realizados uma determinada atividade contém. Assim, foram implementadas as entidades Atividade(classe que contém os atributos que uma atividade possui), AtividadeController(responsável por gerenciar todas as funções do sistema relacionado a atividades), AtividadesRepositorio(possui todas as atividades do sistema, sendo estas armazenadas em um mapa) e Item(classe que contém os atributos que um item deve possuir).

Caso 5:
No caso 5 é apresentada a necessidade de associação entre uma pesquisa, um problema e objetivos, bem como a desassociação entre estes. Para cumprir essas requisições criou-se os métodos associaProblema(), desassociaProblema(), associaObjetivo() e desassociaObjetivo(). Ao executar o metodo associaProblema(), é chamado um método de mesmo nome no pesquisaController passando os parametros originais (idPesquisa e idProblema) e o problemasRepositorio para que o problema possam ser acessados no pesquisaController e associados a um pesquisa, um problema só não pode ser associado a uma pesquisa que já está associada a algum problema, nesse caso é necessário que seja feita uma desassociação. Para associar objetivos a uma pesquisa é seguida a mesma lógica de associação, a única diferença é que uma pesquisa pode ter diversos objetivos e estes podem estar associados a apenas uma pesquisa, a desassociação de objetivos também é feita da mesma forma que a desassociação de problemas. O caso em questão também pede para que seja feita a listagem de pesquisas a partir do idPesquisa, idProblema ou número de objetivos associados. Para listar as pesquisas a partir do id delas, foi implementado um comparable na classe Pesquisa que compara os idPesquisa, para listar a partir da quantidade de objetivos foi criado uma classe comparator e para listar a partir do id dos problemas.

Caso 6:
No caso 6 era requisitado que fosse possível adicionar uma especialização a um pesquisador, podendo essa especialização ser do tipo: Professora ou Aluna. Além disso era necessário que fosse possível associar pesquisador a pesquisas, sendo possível colocar pesquisadores dentro de uma pesquisa. Para a cuidar da especialização, foi criado uma composição, onde foi criado uma interface Especialização, que foi implementada pelas Classes Professora e Aluna. Desse modo, a classe Pesquisador passou a possui um atributo da Interface Especialização, assim quando fosse necessário criar um determinada especialização (ou remover a mesma), Professora ou Aluna seria instanciada (ou

removida), permitindo, portanto, a criação e remoção de uma especialização sem ser necessário alterar o tipo do Objeto Pesquisador. Para cuidar da associação (permitir a associação e removê-la) entre Pesquisador e Pesquisador foram criado os métodos associarPesquisador() e removerPesquisador() em PesquisadorController, onde foram passada como pâmetros desse métodos, o Código da Pesquisa, o email do Pesquisador e o Repositório que possui os Pesquisadores. Assim, ambos os métodos funcionam da seguinte forma: quando se deseja associar ou remover uma associação entre Pesquisador e Pesquisa, o PesquisaController cuida da requisição do Usuário, verificando as entradas e buscando a Pesquisa no Repositório de Pesquisas, já que o controller possui uma instância desse repositório da pesquisa. Feito isso, buscasse o Pesquisador no Pesquisadores Repositório que foi passado como parâmetro desse método. Depois, chamasse o método associar (ou desassociar) na Pesquisa - passando como parâmetro o Pesquisador -, o qual cuida da parte final de inserir (ou remover) um Pesquisador da List de Pesquisadores que há em Pesquisa.

Caso 7:

No caso 7 foi pedido que se associe atividades com pesquisas, ou seja, podem ter várias atividades dentro de uma pesquisa. Foi necessário criar algumas validações , como só poder fazer qualquer comando relacionado a uma atividade caso essa esteja relacionada com uma pesquisa existente no HashMap do repositório e caso essa atividade também exista. Para os casos que não seguiam essa validação um erro era lançado. Foi pedido que seja possível associar/desassociar uma atividade a uma pesquisa. Para tanto, foi colocado um ArrayList de atividades dentro de pesquisa. Além disso, era possível executar uma atividade (que para ocorrer um item deve ser executado e um tempo de duração deve ser acrescentado).Adicionar e remover resultados dentro de uma atividade que para tanto foi utilizado um ArrayList dentro de atividades. Assim como pegar informações gerais da atividade: duração e listar resultados.

Caso 8:

No caso 8 o sistema passa a ter a possibilidade de buscar um termo e recuperar os resultados aonde fôra encontrado esse termo nos repositórios do sistema. Além disso, é possível recuperar um resultado específico e também contabilizar quantas vezes um determinado termo foi encontrado no sistema. Portanto, para isso, foi implementado a entidade Busca a qual será responsável por gerenciar e controlar as buscas feitas nos repositórios.

Nessa classe, temos métodos específicos para percorrer em cada repositório e achar um determinado termo. Ao achar um termo, será adicionado na lista de buscas a string do objeto em que foi encontrado o termo. Feito isso, outro método será responsável por organizar a saída de forma que irá imprimir toda lista de acordo com a forma desejada. Ademais, existe também os metodos para retorna um resultado em específico e também um método que irá contabilizar quantas vezes o termo buscado foi encontrado no sistema.

Caso 9:

No caso 9 pede que seja possível definir uma ordem das atividades a qual não necessariamente é a ordem de cadastro. Logo o usuário do sistema iria definir a ordem das atividades conforme ele deseja-se, porém com algumas restrições como exemplo, o sistema

não permite a criação de loops. Além disso, deve ser possível contar quantas atividades existem depois de uma determinada atividade, pegar uma atividade próxima e de forma mais específica, pegar uma atividade próxima com maior risco. Posto isso, as entidades responsáveis por aplicar essas necessidades no programa são as entidades Atividade e AtividadeController. Sendo assim, adaptamos essas duas classes com métodos para definir a próxima atividade,retirar uma determinada atividade próxima, pegar uma atividade próxima, pegar a atividade próxima de uma que tem o maior risco e por fim, um método para contabilizar quantas atividades próximas existem depois de uma atividade.

Caso 10:

No caso 10 é pedido que o programa dê uma sugestão de atividade a ser realizada. Tal tarefa foi reservada para o “PesquisaController”. Os comandos eram simples: escolher a estratégia de recomendação e pedir a recomendação. As recomendações consistem em quatro algoritmos: MAIS_ANTIGA, MENOS_PENDENCIAS, MAIOR_RISCO e MAIOR_DURACAO . Os algoritmos intuitivamente retornam o que o seu nome diz. Para retornar à mais antiga foi utilizado a lógica de retornar o elemento de índice zero dentro do ArrayList de atividades que ficava dentro da pesquisa em questão. Para as demais características a lógica consistiu em percorrer as atividades que estavam dentro da pesquisa e sempre usar um referencial de quem era a recomendada, ou seja, “MAIOR_RISCO” , por exemplo começaria sendo a primeira atividade, em seguida seria comparada com a segunda , a segunda retornaria seu risco e veria se era mais alto que a então “maior risco” , caso fosse, trocaria de referencial e seria a nova “maior risco”, e esse padrão continuaria até a comparação do último elemento.

Caso 11:

No caso de Uso 11, foi requisito que fosse possível gravar, em arquivo txt, um resumo sobre uma Pesquisa, ou os Resultados de Pesquisa. Para isso no PesquisaController, foram criados três métodos, um para gravar um Resumo, que serve para buscar Informações sobre a Pesquisa (Pesquisadores, Problema, Objetivos e Atividades); O segundo serve para gravar os Resultados de uma pesquisa, que serve para buscar informações de uma pesquisa: A descrição dos Resultados e as Informações sobre os Itens concluídas das Atividades que compõem aquela Pesquisa. O terceiro método, que serve de fato para escrever no arquivo é usado dentro dos dois primeiros, nesse método - gravarArquivo() - passa-se como parâmetro o caminho do arquivo que será gravado (caso o arquivo não exista, será criado; caso exista, seŕa sobrescrito) e o texto que se deseja gravar. Para gravar nesse arquivo foi usado o FileWriter() e o BufferedWriter();

Caso 12:

O caso 12 foi responsável por garantir a permanência de estado entre duas execuções do sistema. Isso foi garantido a partir do momento em que os repositórios são salvos em arquivos .txt através da função salvar() e tais arquivos podem ser lidos a partir da função carregar(). Para executar o método salvar() é utilizado a função writeObject(Object) que possibilita o armazenamento de um objeto no formato serializable em um arquivo, para o método carregar() é utilizado o readObject(Object) que permite ler o arquivo serializable e torná-lo uma classe de novo.

Considerações:

Optamos por fazer nosso projeto em português, pois não vemos necessidade de fazê-lo em inglês, uma vez que, este projeto será analisado apenas pelos professores que nos instruiram e não será divulgado para além do meio acadêmico.