Relatório do Projeto Psquiza de Laboratório de Programação II

DESIGN GERAL

O design geral do nosso projeto foi escolhido para possibilitar uma melhor integração entre as diferentes partes do sistema, criando camadas de abstração que diminuem o acoplamento, através do uso de um controller geral, que delega atividades para cada camadas menores. As entidades principais também possuem um controller próprio, para aumentar o nível de abstração do sistema. Tentamos focar quase todo o projeto no padrão GRASP para obter um melhor design. Com relação a exceptions , fizemos uma classe validadora que possui os métodos responsáveis para lançar as exceções em consideração do tipo de erro. As próximas seções detalham a implementação em cada caso.

US 1

Nessa etapa, foi pedido que fosse criado uma classe que representa uma pesquisa. Cada pesquisa possui uma descrição e um campo de interesse, os 2 em string. Com esse campo de interesse é gerado um código que é formado pelos 3 primeiros caracteres do campo de interesse e que serve como chave para o map de pesquisas. Esse código é gerado na classe PesquisaController, assim como os outros métodos pedidos na especificação. A pesquisa também tem um status, true ou false, que define se ela é ativa ou não.

US 2

O caso 2 pede que seja criado uma entidade que representa os pesquisadores no sistema. Os pesquisadores podem ter 3 funções diferentes: Estudante, Professor e Externo, e todo pesquisador possui nome, biografia, e-mail e foto. Para representar essas diferentes funções, criamos uma String que contém a função do pesquisador. O pesquisador pode ativado e desativado através dos métodos ativaPesquisador e desativaPesquisador, além disso, deve ser possível exibir o pesquisador e alterar o pesquisador.
Para gerenciar os pesquisadores, foi criada uma classe chamada PesquisadorController. Nesta classe, há uma coleção que armazena todos os pesquisadores. Essa coleção é um mapa, onde a chave é o e-mail, que é único para cada pesquisador.

US 3

O caso 3 pede que seja criado duas entidades, uma para representar os problemas e outra para representar os objetivos no sistema. Todo problema possui descrição e viabilidade e todo objetivo possui um tipo, que pode ser GERAL ou ESPECÍFICO, uma descrição, aderência e viabilidade. Deve ser possível exibir tanto os problemas quanto os objetivos, e também apagá-los.

Para gerenciar os problemas, foi criada uma classe chamada ProblemaController. Nesta classe, há uma coleção que armazena todos os problemas. Essa coleção é um mapa, onde a chave é o código de identificação, único para cada problema.
Para gerenciar os objetivos, foi criada uma classe chamada ObjetivoController. Nesta classe, há uma coleção que armazena todos os objetivos. Essa coleção é um mapa, onde a chave é o código de identificação, único para cada objetivo.

US4

O caso4, propôs a criação de duas entidades a 1ª chama-se Atividade, que tem como características uma descrição, um nível de risco associado que possui três estágios: BAIXO, MEDIO e ALTO; a descrição do risco e armazena em um mapa os ITEM's que é a 2ª entidade criada, o ITEM possui dois estágios: REALIZADO e PEDENTE e uma String que representa o próprio, as atividades podem ser apagadas e exibidas pelos os métodos apagaAtividade e exibeAtividade que recebem como parâmetro o ID da atividade, os ITEM's pendentes ou realizados podem ser contados pelos métodos contaItensPendentes e contaItensRealizados que recebem como parâmetros o ID da Atividade e retorna um inteiro representando a quantidade.

US5

O caso 5, propôs a criação de métodos que associam ou desassociam Problemas e Objetivos a Pesquisas, a Pesquisa pode estar associada a um Problema, mas o mesmo Problema pode estar associado a várias Pesquisas e a Pesquisa pode ser associada a vários Objetivos, mas cada Objetivo só pode estar associado a uma Pesquisa, foram criados quatro métodos que retornam um valor boolean, e recebem como parâmetros o ID da Pesquisa e o ID do problema ou objetivo que vai ser associado ou desassociado, com exceção do método desassociaProblema que recebe somente o ID da Pesquisa. Também foi proposto criar o método listarPesquisas que tem três critérios de ordenação: 1) Pesquisas com associações a problemas de maior Id aparecem primeiro, e por último as pesquisas sem Problemas associados ordenadas por seus respectivos ID's. 2) Pesquisas com maior número de objetivos associados aparecem primeiro, em caso de empate é comparado seus respectivos ID's. 3) Este último é ordenado da Pesquisa de maior ID para a de menor ID.

US6

Nessa etapa, foi pedido que fosse criado uma associação entre pesquisa e pesquisador, além de adicionar uma especialidade (professor ou aluno) para o pesquisador sem especialização (pesquisador externo). Foi criado uma classe Especialidade e as classes PesquisadorAluno e PesquisadorProfessor, que implementam Especialidade para assim

alterar o toString() que muda de acordo com a especialidade, pois cada uma tem as seus atributos específicos.

US7

O caso 7 pede que seja possível a associação e execução de atividades. Para associar uma atividade a uma pesquisa, optamos por criar um mapa dentro de pesquisa que contém as atividades associadas àquela pesquisa. Além disso, deve ser possível cadastrar e remover resultados em uma pesquisa e listar esses resultados, decidimos guardar os resultados em uma mapa onde eles são identificados pelo número de cadastro. Também é possível obter o tempo de execução dos itens de uma atividade, a partir do método getDuracao.

US8

O caso 8 pede que seja possível buscar por palavras-chave nos elementos cadastrados no sistema, pegar um resultado específico (em ordem) e contabilizar o total de resultados.. Para isso, foi criado a classe ControllerGeral que recebe os controllers de todas as outras entidades no sistema, e assim o método de busca percorre por todos eles em buscando a palavra-chave. Quando é passado uma palavra-chave e um número inteiro, o método busca retorna um resultado específico e o método contaResultadosBusca contabiliza o número total de resultados.

US9

O caso 9 pede que seja possível definir uma ordem de execução para representar uma lógica de execução das atividades. Para isso, optamos por utilizar recursão, e assim toda atividade possuía outra atividade dentro de si própria, onde essa atividade representava a atividade subsequente que é definida no método defineProximaAtividade e essa ordem pode ser quebrada com o método tiraProximaAtividade. Além disso, é possível contar quantas atividades estão ordenadas a partir do método contaProximos e também é possível retornar a enésima atividade ordenada a partir do método pegaProximo. Por fim, é possível pegar a atividade com maior risco partir da ordenação da ordem as atividades com o método pegaMaiorRiscoAtividade.

US10

O caso 10 pede que seja possível oferecer uma sugestão de próxima atividade a ser realizada dentro de uma pesquisa a partir de uma estratégia configurada pelo usuário. Para isso criamos o método configuraEstrategia, que define qual estratégia será utilizada na sugestão da próxima atividade. Por padrão a estratégia configurada de início é a

MAIS_ANTIGA, mas o usuário pode optar também pelas seguintes estratégias: MENOS_PENDENCIAS, MAIOR_DURACAO, MAIOR_RISCO.

US11

Nessa etapa, foi pedido para que fosse criado método para criar um arquivo .txt e para gravar nele o estado atual de uma pesquisa ou o resultado final. No estado atual, o método que fizemos na classe PesquisaController (pois se liga com as outras classes necessárias para realizar o que foi pedido nessa etapa) precisa, através do código da pesquisa, pegar e gravar todos os pesquisadores que estão associados com essa pesquisa, o problema, objetivo e as atividades. Para o resultado final, bastou pegar os resultados da pesquisa de acordo com o código e gravar o resultado dela.

US12

Até a US11 o sistema operava sem garantir a permanência de estado entre as execuções, mas a US12 propôs que fossem criados dois métodos, um deles é o método “salvar” como o próprio nome diz, esse método salva o estado atual do sistema, o segundo método proposto foi o de “carregar” que é dependente do método anterior, pois o carregar restaura o estado do sistema que foi anteriormente salvo pelo o método salvar, ao utilizar o carregar é restaurado as relações das entidades e o que as compõem, inclusive, ordem de armazenamento e a configuração.