Relatório de Projeto LP2 2019.2 - Psquiza

**Design geral:**
O design geral do sistema foi sendo pensado e modificado no decorrer do projeto. Optamos por desenvolver uma Facade, responsável pela comunicação entre a interface com o usuário e o restante do sistema. A Facade se comunica diretamente com cinco controllers, responsáveis por funcionalidades específicas, relacionadas entidades mais básicas da Psquiza.
Também foram utilizados conceitos de Orientação a Objetos no decorrer do projeto, como por exemplo os conceitos de Interface, Composição e Strategy.
Para lidarmos com validações de entradas, criamos uma classe específica que lança as exceções necessárias em caso de atributos inválidos. Os métodos dessa classe são todos estáticos para que não seja preciso instanciar o objeto do Validador antes de invocar a validação.
Os pacotes do projeto estão divididos por funcionalidade, sendo oito no total.
As próximas seções detalham a implementação em cada caso.

**Caso de Uso 1:**

O caso de uso 1 pede que o sistema possa cadastrar e administrar pesquisas que possuem uma descrição e um campo de interesse. Essas pesquisas podem ser encerradas, alteradas, ativadas e exibidas. Para isto, foi criada a classe pesquisa e a classe controller de pesquisa. A classe pesquisa possui, além dos atributos propostos, um atributo referente ao seu status. Já a classe controller, armazena as pesquisas cadastradas em um map, no qual a chave de cada pesquisa é gerada no próprio controller a partir das tres primeiras letras de seu campo de interesse e de um número que identifica quantas pesquisas com esse código já estão cadastradas. Métodos de validação também foram utilizados para verificar se as entradas estavam de acordo com o que era pedido na especificação.

**Caso de Uso 2:**
No caso de uso 2 é relatado a importância do pesquisador para a pesquisa, sendo assim é necessário que o sistema realize o cadastro do mesmo. Assim é criada a entidade Pesquisador, que após seu cadastro pode ter suas informações alteradas posteriormente e exibidas no sistema. Além disso, o pesquisador pode ou não estar ativo, ou seja, ele está disponível para trabalhar em alguma pesquisa, para verificar sua disponibilidade foi criado o método pesquisadorEhAtivo() que recebe o e-mail do pesquisador e retorna sua disponibilidade em verdadeiro ou falso. A entidade pesquisador ficou contida em um mapa de pesquisador no PesquisadorController de Psquiza devido a seu identificador único, o email.

**Caso de Uso 3:**
O caso de uso 3 apresenta a problemática da criação de Problemas e Objetivos de uma pesquisa acadêmica no sistema. Por serem duas entidades bastante relacionadas, resolveu-se mantê-las controladas por uma mesma classe chamada de ProblemaObjetivoController. Por terem identificadores únicos no sistema, Problema e Objetivo foram armazenados em dois mapas, cujas chaves são os códigos dos problemas e objetivos cadastrados no Psquiza.

**Caso de Uso 4:**
O caso de uso 4 retrata a importância da descrição e planejamento de atividades relacionadas à pesquisa, cada atividade é composta por itens que podem ser marcados como feitos ou não.

Assim, foi implementada as entidades Atividade e Item. Considerando que as atividades possuem identificadores únicos, optamos por contê-las em um mapa, cujas chaves são códigos gerados através do método generateCodeAtividade, cada atividade possui uma lista de itens a serem realizados. Para controlar as atividades, e por consequência os itens, foi criado um classe chamada AtividadeController.

.

**Caso de Uso 5:**

O caso de uso 5 foi responsável por gerar a primeira mudança brusca de design do projeto, sendo o primeiro a apresentar a necessidade de ligação entre classes completamente distintas, pois, passou a ser necessário a associação de problemas e objetivos a uma pesquisa.

Foi criado métodos de associação dentro do pesquisaController que através do código da Pesquisa e do código de um problema/objetivo guarda o objeto da associação dentro de uma pesquisa específica. também é possível desassociar um problema ou um objetivo de determinada pesquisa e listar todas as pesquisas do sistema.

**Caso de Uso 6:**

O caso de uso 6 trouxe bastante discussão de design entre a equipe, os monitores e professores. A problemática surgiu da necessidade que havia de se cadastrar especialidades (aluno e professor) para os pesquisadores. Estas especialidades são opcionais e afetam atributos e comportamentos dos Pesquisadores cadastrados no sistema.

A solução encontrada foi criar uma Interface Especialidade que é implementada pelas classes Aluno e Professor. Desta forma, além de guardar uma String com a função do Pesquisador, a classe também armazena um atributo do tipo Especialidade que no momento da criação do Pesquisador é configurado como null. Caso o Pesquisador tenha a função de Professor, esta especialidade poderá ser cadastrada para o tipo Professor, da mesma forma para a função de Aluno. Se o pesquisador possui a função do tipo Externo, esse atributo permanecerá nulo para sempre.

As associações de pesquisador com pesquisa foram feitas armazenando os pesquisadores em um mapa, cujas chaves são os emails e valores são os objetos do tipo Pesquisador, dentro da classe Pesquisa, uma vez que uma pesquisa pode conter vários pesquisadores associados.

**Caso de Uso 7:**

Surgiu na Psquiza a necessidade de após tantos cadastros de entidades separadas algumas serem associadas, no caso de uso 7 então começou a associação de atividades às pesquisas usando apenas os códigos de cada e guardar as atividades dentro das pesquisas a quais elas foram associadas. Além disso, é possível executar as atividades, cadastrar os resultados das mesmas e listá-los depois. Por fim, a duração total de uma atividade é mostrada pelo método getDuracao().

**Caso de Uso 8:**

A partir desse caso de uso o sistema exige a possibilidade de que sejam realizadas buscas em suas entidades, retornando os objetos que possuem o termo requisitado. Para isto, cada classe implementou seu método de busca, avaliando os atributos adequados para elas. Além disso as classes também implementaram a interface Comparable e métodos CompareTo() para que fosse possível comparar termos presentes nesses atributos e assim retornar os objetos adequados. Um controller de busca foi criado, para encapsular a lógica envolvida no processo e nele eram aglomerados os resultados encontrados em todas as entidades, os quais deveriam ser retornados na chamada do método dentro da facade.

**Caso de Uso 9:**
A especificação do caso de uso 9 pedia que a criação de cadeias de atividades, que consistem em sugestões de ordens a serem seguidas na realização das atividades de uma pesquisa. A estratégia utilizada para isto foi a utilização de estruturas recursivas. Uma Atividade pode apontar para uma próxima Atividade como sugestão de atividade a ser realizada em seguida desta. Desta forma, toda Atividade tem o atributo proximaAtividade do tipo Atividade, inicialmente configurado como nulo.
Todos os cinco métodos deste caso de uso precisaram utilizar também chamadas recursivas, direta ou indiretamente.

**Caso de Uso 10:**
Nesse caso de uso, o sistema agora deve oferecer a sugestão da próxima atividade a ser realizada dependendo da estratégia utilizada na Psquiza.
Para isso foi criado o atributo Estratégia e o método configuraEstrategia() que permite que o projeto possa trabalhar com uma dentre as 4 estratégias disponíveis, são elas: mais antiga (estratégia padrão do sistema), maior risco, maior duração e menos pendências. Após isso,usando o padrão strategy foi criada a interface Estratégia que contém o método proximaAtividade() que permitia que cada estratégia, sugerisse da sua forma a próxima atividade.

**Caso de Uso 11:**
Neste caso de uso o sistema deve ser capaz de exportar os dados de uma pesquisa (resumo geral da pesquisa e seus resultados) para um arquivo de texto. Para implementar tal funcionalidade foi criado os métodos gravarResumo e gravarResultados que faz uso das classes de manipulação de arquivos FileWriter e PrintWriter, gravando tais conteúdos em arquivos .txt na raiz do sistema. Além desses métodos, foram implementados métodos públicos e privados auxiliares para aumentar a coesão e respeitar os demais padrões de desenvolvimento.
**Caso de Uso 12:**
Este caso de uso pedia para que o sistema garantisse a persistência de estado entre execuções. Para tal, os métodos salvar() e carregar() deveriam ser criados para salvar em arquivo o estado do sistema e recuperar tal estado, respectivamente. Dessa forma, uma classe contendo esses métodos foi criada e cada controller do sistema chama o método salvar() para salvar as atividades realizadas em um arquivo de texto.Este método recebia um objeto que seria salvo e o nome do arquivo em que seria salvo. Esses controllers também chamavam o método carregar() individualmente para que o sistema retornasse ao estado salvo nos arquivos de texto. Este método recebia o nome dos arquivos que haviam sido salvos e retornava objetos restaurados a partir disso.

**Considerações:**
Por fim, mesmo com eventuais dificuldades, o projeto pode ser considerado pela equipe um sucesso. E gostaríamos de agradecer a todos que contribuíram seja por apoio ou explicando conceitos que ainda não tinhamos domínio.

Link para o repositório no GitHub:
https://github.com/ummatias/Psquiza20192