Relatório Projeto de Programação II - 2019.2
Equipe: Maria Eduarda de Azevedo, Arthur Silva Cavalcante Ferreira, Daniel Dantas Fonseca, Brenda Louisy Morais Alves
Professor Orientador: Matheus Gaudêncio

1. **Design geral do sistema**

O design geral escolhido é composto pela divisão de classes em pacotes de acordo com suas complexidades e funcionalidades. É importante salientar a existência de uma classe de validação da qual todas as classes bases e controllers do sistema são subtipos dessa.
Os pacotes são divididos em:
1.1 Base:
Neste pacote estão inseridas as entidades básicas do sistema, sendo elas:

- Pesquisador
- Pesquisa
- Atividade
- Item
- Objetivo
- Problema
- Aluno
- Professor
- Arquivo

1.2 Controller:
Neste pacote estão as entidades controladoras das classes base responsáveis por armazenar e manipular os objetos de seus respectivos tipos responsáveis. São as classes deste pacote:

- Conector
- ControllerAtividades
- ControllerBuscas
- ControllerObjetivos
- ControllerPesquisador
- ControllerPesquisas
- ControllerProblemas

1.3 Exceções:
Neste pacote estão contidas as entidades que são subtipos de RuntimeException. São as classes deste:

- ActivationException
- AssociationException
- SequenceException

1.4 Comparators:
Neste pacote estão as classes que implementam a interface Comparator e servem para ordenação de acordo com o critério desejado:

- OrdenaAtvdsMaiorDuracao
- OrdenaAtvdsMenosPendencias
- OrdenaPesquisaID

- OrdenaPesquisaObjetivo
- OrdenaPesquisaProblema

1.5 Validação:
Neste pacote está contida a classe de validação geral do sistema, sendo ela Validacao. A escolha por manter a classe separada das demais é devido ao fato desta ter características divergentes das demais classes do sistema.

1.6 Facade:
Neste pacote estão contidos a Facade e o Main. O Main é responsável por executar os testes do EasyAccept e a Facade como interface destinada para o desenvolvedor front-end e mais próxima do usuário final.

Considerações sobre o design
A classe Conector foi codificada com o intuito de mediar as associações entre os controllers, implementando as funcionalidades que exigiam a informação de mais de um controller. Além disso, foi visto como melhor opção por diminuir o acoplamento, uma vez que, todos os controllers conhecem uma única classe ao invés de todos se conhecerem.

2. Casos de Teste:

Caso 1 - Pesquisas
A US1 pede que seja criado uma entidade que represente as pesquisas no sistema, sendo assim foi feita a classe Pesquisa. Para gerenciar as pesquisas do sistema, tendo em vista que cada pesquisa gera um código representativo único para a mesma a partir das suas 3 primeiras letras da descrição e um número inteiro começando por 1, crescendo de acordo com o número de pesquisas que possuem as mesmas 3 letras iniciais iguais da descrição, foi utilizada a coleção TreeMap para armazenar e ordenar as pesquisas. Além disso, foi implementada uma lógica de exceção para ser lançada em casos específicos de uso do sistema, como por exemplo, a não permissão para alterar a descrição ou o campo de interesse da pesquisa, após seu cadastro.

Caso 2 - Pesquisadores
Para este caso de teste foi construída uma entidade que representa um pesquisador, a classe Pesquisador, contida no pacote base. Um pesquisador possui como atributo nome, função, biografia, e-mail e foto. Além disso, possui um atributo de verificação de status, chamado ativada, que fornece a situação de ativo ou inativo de um pesquisador. Para a conexão com o Facade, foi criada uma classe controladora, ControllerPesquisador, que opera diretamente com o objeto Pesquisador e constrói os mesmos. Nele, são armazenados em um mapa todos os pesquisadores cadastrados no sistema, sendo o e-mail o identificador único de cada pesquisador. Assim como a maioria das classes do sistema, essa também

herda da classe Validação, que atua como avaliadora de entradas do sistema e lança exceções caso tais entradas não sejam permitidas.

Caso 3 - Problemas e Objetivos

Para implementar as funcionalidades deste caso de teste foram construídas as entidades Problema e Objetivo, estando ambas contidas no pacote base. Desse modo, um Problema possui como atributo uma descrição e um inteiro correspondente à viabilidade do tratamento deste Problema. Além disso, cada problema tem como atributo um ID que o identifica unicamente no sistema. A decisão por utilizar este ID como atributo desta entidade se deu devido à necessidade de implementar os métodos equals e hashCode, que fariam uso deste ID. Para conexão com a Facade, foi criada uma classe controladora ControllerProblemas (contida no pacote controller) que realiza todas as operações dos objetos do tipo Problema cadastrados no sistema, além de armazenar todos estes objetos em um mapa cuja chave corresponde ao ID destes, sendo este ID gerado automaticamente na classe pela ordem de cadastro destes objetos. Assim, ControllerProblemas é a classe Creator de Problema devido ao fato de que este controller usa, opera e armazena objetos deste tipo, além de ser responsável por gerar os IDs de cada um dos problemas.
Ademais, como já mencionado anteriormente, foi criada a classe Objetivo, que tem como atributo uma String que corresponde ao seu tipo, podendo ser “GERAL” ou “ESPECIFICO”, uma descrição e dois inteiros correspondentes à aderência e viabilidade. Assim como na classe Problema, em Objetivo o ID de um objeto desse tipo no sistema é utilizado como atributo dessa entidade, o que se justifica devido à necessidade de implementar os métodos equals e hashCode, que comparam objetivos a partir dos seus IDs. Para conexão com a Facade, foi criada a classe controladora ControllerObjetivos (contida no pacote controller) que realiza todas as operações com os objetivos cadastrados no sistema, além de armazená-los num mapa cuja chave corresponde ao seu ID, sendo este gerado no próprio controller. Dessa forma, ControllerObjetivos é o Creator de todos os objetos do tipo Objetivo do sistema, sendo, inclusive, esta classe quem gera os IDs destes, que é uma escolha de design que se justifica pelo fato do ID de um Objetivo ser gerado automaticamente no sistema e ser o controller quem utiliza, armazena e opera com estes objetos.

Caso 4 - Atividades Metodológicas

Para este caso de teste foi escolhida a alternativa da construção de uma entidade que representa uma atividade, para tal foi criada a classe Atividade, contida no pacote base. Para representar os itens que cada Atividade contém, foi escolhida a criação de uma outra entidade para sua representação, a classe Item, também contida no pacote base. A escolha da criação de tais entidades foi feita para que o encapsulamento das funcionalidades básicas dessas se mantivessem.
Toda atividade tem associada a ela uma lista de objetos Item. A escolha por essa associação está vinculada ao fato de Item apenas existir em Atividade, o que se faz adequado responsabilizar Atividade pela criação desses objetos no sistema. A escolha da estrutura da lista para o armazenamento foi pensada em conjunto com a necessidade de manter a ordem de cadastro dos itens e ter o índice acrescido

de uma unidade como forma de acessá-los no sistema.
Para a conexão com a Facade, foi criada uma classe controladora, ControllerAtividades, que opera diretamente em objetos Atividade. É também no controller que se armazenam todas as atividades que foram cadastradas no sistema em um mapa, no qual a chave indica seu id e o valor apontado é o objeto Atividade a que o id pertence. A classe ControllerAtividades se porta como construtora de objetos Atividade.

Caso 5 - Associações de Objetivos e Problema

Para a implementação deste caso foram necessárias adições às classes anteriormente criadas no Caso 1.
As associações entre Pesquisa, Objetivo e Problema foram feitas com o auxílio da classe Conector, que, neste caso, delega a classe ControllerPesquisas como sendo a mais adequada para receber os objetos que foram pegos dos outros controllers e associá-los.
A escolha pela melhor adequação desse controller recai sobre a especificação dada, na qual “uma pesquisa está ligada a um problema e também está associada a um, ou mais, objetivos”, o que se deixou interpretar que as informações de uma pesquisa são também compostas pelos seus objetivos e problema associados e seria adequado manter esses objetos dentro de Pesquisa. Além de tal motivo, a necessidade de ordenação das pesquisas com base nos seus objetivos e problemas associados legitima a escolha.
Em pesquisa, o único Problema a ela associado é guardado em um atributo e os objetivos em um mapa, no qual a chave indica o id que aponta diretamente para seu Objetivo correspondente.
Para atender a especificação da ordenação das pesquisas pelos critérios pedidos foram criadas três classes de comparação: OrdenaPesquisaID, OrdenaPesquisaObjetivo, OrdenaPesquisaProblema. Todas essas classes estendem da Interface Comparator.
Para efetuar a ordenação os métodos compare() das classes avaliam respectivamente:

- O toString() dos objetos comparados;
- O toString() dos objetos Objetivo de maior id associados aos objetos Pesquisa passados;
- O toString() dos objetos Problema associados a cada uma das pesquisas comparadas;

Caso 6 - Associação e Especialização da Pesquisadora

Este caso de teste especifica que seja possível especializar um Pesquisador ou para professor ou para aluno. Desse modo, foram criadas as classes Professor e Aluno, sendo ambas subclasses que herdam de Pesquisador e estando ambas contidas no pacote base. A escolha de utilizar herança se mostra válida devido ao fato da especificação explicitar claramente que é necessário especializar um Pesquisador. Assim, como pela definição de herança tem-se que cada subclasse

é uma especialização da superclasse, então conclui-se que o uso deste mecanismo era o mais adequado nesta situação.
Além disso, como cada Pesquisador poderia ser cadastrado no sistema como tendo as funções “externo”, “professor” ou “estudante”, mas apenas estudantes poderiam ser especializados para Aluno e apenas professores poderiam ser especializados para Professor, foi implementado uma verificação a partir dos métodos de Validacao para que seja garantida a implementação desta especificação e que exceções sejam lançadas corretamente.
Ademais, a classe Professor tem como atributo, além dos atributos herdados da superclasse, uma formação, unidade alocada e data de contratação. Da mesma forma, a classe Aluno tem como atributo, além dos atributos herdados da superclasse, um inteiro correspondente ao semestre e um double correspondente ao seu Índice de Eficiência Acadêmica (IEA). Desse modo, cada atributo das subclasses de Pesquisador, até aqueles que não eram herdados da superclasse, poderiam ser alterados no sistema, o que só foi possível devido à sobrescrita do método *alteraPesquisador*, que é um método herdado por Professor e Aluno da classe Pesquisador.
Concomitantemente, para que a especificação fosse cumprida completamente, também foi necessário sobrescrever o método *toString*, herdado da superclasse de Professor e de Aluno para que cada representação em String de ambos os tipos tivessem as informações necessárias. A sobrescrita desse método altera a forma de como os pesquisadores especializados são exibidos no sistema e, consequentemente, como estes são listados pelo método *listaPesquisadores*.
Então, dessa maneira, através da utilização de herança, o polimorfismo para o armazenamento de pesquisadores em ControllerPesquisador (e para qualquer outra operação envolvendo Pesquisador no sistema) foi garantido, uma vez que todo Aluno e todo Professor são, também, pesquisadores.
Em adição a isso, a especificação ainda explicitava a necessidade de associar ou desassociar um Pesquisador (ou qualquer uma de suas especializações) à uma Pesquisa. Portanto, para implementar esta funcionalidade, como uma Pesquisa pode ter vários pesquisadores associados a ela, então, foi adicionado um mapa cuja chave seria o ID de um pesquisador como atributo de Pesquisa para armazenar os pesquisadores associados, sendo esta associação feita através do uso da classe Conector para juntar a lógica dos controllers de Pesquisador e de Pesquisa.

Caso 7 - Associação e execução de atividades
O caso de uso da US7 pede que seja possível associar atividade à pesquisa, para que a primeira possa ser executada. Para isso, foi necessária a criação de um HashMap que armazena as atividades associadas a uma pesquisa, sendo posteriormente alterado para LinkedHashMap, por questões de ordenação, tendo como chave de acesso o código da atividade. O métodos de associa e desassocia atividade foram implementados no Conector para que fosse possível ter acesso ao

ControllerAtividades e ControllerPesquisas simultaneamente. Além disso, a especificação pede que na execução de uma atividade, seja determinado o item a ser executado e a quantidade de horas gastas nessa execução. Sendo assim, foi implementado o método executaAtividade, também no Conector, pela necessidade de verificar se existiam atividades associadas a alguma pesquisa. No controllerAtividades contêm o método executaAtividade, que chama executaItem, implementado na classe Atividade, sendo possível, na implementação, executar o item e guardar a duração da execução. A solicitação seguinte é que seja possível cadastrar os resultados obtidos por um item da atividade. Para isso, foi criado em Atividade um HashMap contendo como valor os resultados cadastrados em determinada atividade, tendo como chave de acesso o número do resultado, incrementado a cada cadastra de resultado. É possível também remover um resultado cadastrado, método removeResultado implementado no ControllerAtividades, assim como o cadastraResultado. Pede-se que seja possível listar os resultados cadastrados e ter acesso à duração de cada execução, através do getDuracao.

Caso 8 - Busca por Palavra-chave
A US8 pede que seja possível realizar a busca de um termo nas entidades cadastradas no sistema, retornando quais tem suas ocorrência, além de retornar um resultado específico em ordem (recebendo assim, além do termo, um inteiro referente a posição desejada) ou o número de entidades que possuem a ocorrência do termo. Tendo isso em mente, o ControllerBuscas foi implementado. Este Controller recebe todas as collection's que contêm as entidades do sistema e verifica em quais delas há a ocorrência do termo desejado, de acordo com os atributos de cada entidade. Por exemplo, nas pesquisas é verificado se o termo está presente na descrição e/ou no campo de interesse. A partir disso, é feita a ordenação como descrita nas especificações desta US e retornada assim a representação textual (toString) de cada entidade. É válido ressaltar que para simplificação de implementação, as collections das classes ControllerObjetivos, ControllerPesquisador, ControllerPesquisas e ControllerProblemas foram alteradas para TreeMap, uma vez que sua utilização facilitou o retorno na ordenação anti lexicográfica.

Caso 9 - Ordem das Atividades
Para que seja possível determinar uma ordem de execução de atividades de uma determinada pesquisa foram utilizados conceitos de recursão orientada a objetos, uma vez que foram feitas alterações na classe Atividade de modo com que cada objeto desse tipo tivesse como atributo outra Atividade, que seria sua sucessora na ordem de execução dessas. Dessa forma, como a especificação deixa explícito que a ordem é utilizada apenas como "mecanismo de sugestão e planejamento,

mas não altera como as atividades são executadas” não foi necessária nenhuma alteração nos métodos de execução das atividades.
Assim, foram implementados métodos tanto na classe Atividade como em seu controller que permitiram adicionar uma outra atividade que sucedesse outra, assim como retirar uma atividade sucessora, contar quantas atividades sucediam uma outra, retornar o ID de uma atividade com um determinada índice da sequência e retornar o ID da atividade de maior risco da sequência.

Caso 10 - Próxima atividade
A US10 solicita que o sistema ofereça a possibilidade de sugerir uma próxima atividade a ser realizada dentro de um pesquisa. Para isso, é preciso definir estratégias que definam qual próxima atividade será realizada, sendo elas a atividade mais antiga com itens pendentes, que contenha menos pendências, a atividade pendente que contenha o maior risco e atividade que contenha maior duração de itens realizados, tendo como prioridade a estratégia mais_antiga. Para isso, foram criados os métodos configuraEstrategia e proximaAtividade no controllerPesquisas. Como escolha de ordenação para ter acesso às atividades de acordo com cada critério, para as estratégias de maior_duracao e menos_pendencias foram criados Comparators na classe comparators que ordenava as atividades associadas a uma pesquisa de forma decrescente para que a primeira atividade contivesse a maior duração e crescente para que a primeira atividade fosse a que contêm menos pendências, respectivamente.

Caso 11 - Resultados
Para gravar os resultados e resumos de uma pesquisa em um arquivo de texto foi delegada ao ControllerPesquisas a responsabilidade de pegar essas informações de seus objetos Pesquisa, tendo em vista de que uma Pesquisa tem todas informações necessárias para gerar seu próprio resumo e seus próprios resultados. Na implementação, utilizou-se um objeto do tipo File para criar um arquivo de texto e um objeto FileWriter para gravar as informações geradas em String pela entidade Pesquisa. Como solicitado na especificação, os arquivos são guardados na raiz do projeto e são sobrescritos caso os métodos da Facade chamem um arquivo já existente. Como também especificado, os arquivos de resumo são indicados pelo id da pesquisa e os de resultado também pelo id concatenado a “-Resultados”.

Caso 12 - Persistência
Na US12, foi requisitado que a equipe adicionasse a funcionalidade de salvar o estado do sistema para persistência de todos os dados após o encerramento. Para isso, foi implementada a classe Arquivo, responsável por gravar em arquivo

uma coleção ou um número inteiro e fazer a leitura do mesmo. Sendo assim, os atributos dos controllers que tinham essa funcionalidade de armazenar uma coleção de objetos ou número inteiro foram salvos e posteriormente carregados, sendo cada controller responsável por salvar seus arquivos e carregá-los. Além disso, foi necessário implementar a interface serializable em todas as classes do pacote Base, uma vez que estão presentes nas coleções que serão salvas. Para maior organização, foi criada a pasta "/Dados/" na raíz do projeto com a função de armazenar esses arquivos que contêm os dados do sistema. Como solicitado na especificação, os arquivos são sobrescritos caso o método de salvar seja executado novamente.

3. Considerações

A escolha de design de ter um Conector foi bastante questionada. No entanto, essa implementação se mostrou bastante válida no decorrer do desenvolvimento do sistema, uma vez que, na parte 2 do projeto, especificamente na US6, era requisitado a especialização de um Pesquisador para Aluno ou Professor. Durante a implementação da US11 observou-se a persistência de alguns erros no EasyAccept derivadas da US6 que puderam ser corrigidos com a transferência de método do ControllerPesquisas para o Conector, fazendo com que esse operasse ao mesmo tempo sobre os controllers de pesquisa e pesquisador para alterar o tipo do objeto sem ferir o encapsulamento de forma brusca e que dificultasse o desenvolvimento do projeto.

4. Diagrama de Classes

https://drive.google.com/open?id=14SKRsGaxiOH4POAm-55Rx5uWt0WurEEl