Relatório de Projeto da disciplina Laboratório de Programação II

Alunos: Ana Beatriz da Silva Truta, José Matheus do Nascimento Gama, Ricardo Adley da Silva Sena, Vinicius Moraes Valença Varjão.

Professora responsável: Eliane Araújo.

Monitores responsáveis: Débora Leda de Lucena Souza e Pedro Farias Wanderley

Design Geral:

Nosso projeto focou principalmente na coesão semântica, na abstração e no baixo acoplamento do código. Para isso, separamos o projeto em pacotes de forma que cada pacote gira em torno de uma entidade específica do código e, para cada entidade, há também um controlador próprio, para evitar o sobrecarregamento de todos os outros controladores, dinamizar o fluxo das informações e delegar responsabilidades para as classes menores.

Pelo alto nível de associação entre essas entidades, exigida pela especificação, alguns controladores acessam outros controladores, mas sempre prezando pela ocultação da informação.

Para a ordenação dos objetos, utilizamos a interface Comparator para cado caso exigido. Para a validação das entradas, utilizamos uma classe responsável para verificar se cada entrada é nula, vazia ou não atenda o formato exigido, e lançar as exceções correspondentes.

As próximas seções detalham tudo o que foi feito no código para cada caso de uso.

- **Caso 1:**

No caso 1, é apresentada a entidade Pesquisa. Nela estão presentes as classes Pesquisa e PesquisaController. Cada Pesquisa possui uma descrição (texto livre que faz um resumo da pesquisa a ser realizada), um campo de interesse (serve como marcador da área ou tema da pesquisa) e um código (gerado automaticamente pelas três primeiras letras do campo de interesse e um valor inteiro) que é o identificador único da pesquisa.

Para facilitar a organização do workspace dividimos em pacotes que agrupam todas as classes que possuem funcionalidades em comum. No package pesquisa a parte lógica ficou por conta da classe PesquisaController e a classe Pesquisa fornece o objeto que é utilizado no Controller.

- **Caso 2:**

No caso 2, é apresentada a entidade Pesquisador. Nela estão presentes as classes Pesquisador e PesquisadorController. O objeto pesquisador possui nome, email (identificador do pesquisador no sistema), função, biografia e foto. Já o controlador do mesmo agrega os métodos que manipulam e retornam os atributos do pesquisador.

- **Caso 3:**

No caso 3, são apresentadas duas novas entidades: Problema e Objetivo. Cada Problema possui uma descrição e um inteiro que representa o quanto ele é relevante e bem definido. O Objetivo tem dois tipos, geral ou específico, uma descrição e inteiros que representam o quanto ele está aderido a um problema e o quanto é viável de ser realizado.

Essas entidades foram armazenadas em controladores próprios, ProblemaController e ObjetivoController, dentro de coleções do tipo HashMap, já que ambas têm identificador único e precisam ser acessadas a partir do mesmo. Cada um desses controladores estão em pacotes diferentes, deixando claro por qual funcionalidade eles estão responsáveis.

- **Caso 4:**

No caso de uso 4, é apresentada uma nova entidade: Atividade , sendo que esta ainda contém em sua composição a entidade Item. Para uma criação bem sucedida da entidade Atividade é necessário que seja informado uma descrição, uma duração, o resultado que é esperado para essa atividade e , por fim, um risco associado a execução dessa atividade.

O risco de uma atividade pode, ainda, ser definido por três níveis distintos, sendo eles: baixo, médio e alto.

A entidade atividade armazena seus itens em um HashMap e são estes itens que compõem os resultados esperados, cada item possui um status, iniciado sempre como PENDENTE mas podendo ser alterado para REALIZADO de acordo com a vontade do usuário.

Todas as ações em Atividade e item são controladas por uma entidade controladora chamada ControladorAtividade que armazena todos os dados gerados em mapas, contêm todos os métodos necessários para executar as funcionalidades do sistema envolvendo Atividade e Item, e ainda verifica, utilizando a entidade Verificador, os dados informados pelo usuário antes que sejam devidamente inseridos no sistema.

- **Caso 5:**

No caso 5, é exigido que problemas e objetivos sejam associados ou desassociados a outras pesquisas. Para isso, tornamos possível o acesso de PesquisaController a ProblemaController e ObjetivoController, já que era necessário que o controlador de pesquisas pudesse acessar os problemas e os objetivos, e isso foi definido logo no construtor da Facade. Dessa forma, a entidade pesquisa só tem acesso aos métodos do que está associado a ela, dentro da própria classe.

Além das associações, a US5 também pede que as Pesquisas possam ser listadas a partir de vários critérios de ordenação: pelo ID da pesquisa em ordem decrescente, pela quantidade de objetivos associados a pesquisa em ordem decrescente e, também, pelo ID do problema associado à pesquisa. Para cado caso de ordenação, utilizamos a interface Comparator, já que essas ordenações não agem de uma maneira natural e intrínseca

unicamente a uma entidade, e sim de forma múltipla (Exemplo: ID da Pesquisa + ID do Problema, e de forma decrescente).

- **Caso 6:**

No caso de uso 6 foi alterada a entidade Pesquisador, adicionando a ela novas características para um pesquisador que esteja cadastrado como professor e aluno. Para que isso fosse possível, a opção escolhida foi a de uma Interface Especialidade e criar as classes Professor e Aluno que implementavam de Especialidade. Além de poder associar Pesquisador a uma Pesquisa. Para fazer essa associação o PesquisadorController foi chamado na classe PesquisaController que recebia os dados e os guardava em um HashMap.

Para que todos os dados pudessem ser adicionados, foi criado um verificador que recebia as entradas do usuário, conferia se estava em conformidade com o atributo e se cos não estivesse, lançava um erro. Com a chegada dos novos atributos para pesquisador, a sua exibição foi alterada de acordo com a sua função, bem como novos atributos poderam ser editados, alterando assim métodos feitos na US2.

- **Caso 7:**

No caso de uso 7, foi criada uma associação entre atividades e pesquisas, adicionando métodos que deixam a representação da pesquisa mais fiel à realidade, trazendo opções como a de realizar uma atividade e cadastrar seus resultados. A associação foi feita guardando-se as atividades em um LinkedHashMap dentro da pesquisa associada, e para o funcionamento perfeito dos métodos, foi adicionado por composição o controlador de atividades ao de pesquisa. Houveram mudança na codificação das atividades para se adequar à nova especificação, como por exemplo a duração que agora começa em 0.

Nesse caso de uso, tive problemas apenas para perceber o que a especificação queria, mas, assim que entendi, o código foi feito sem muitos problemas.

- **Caso 8:**

No caso de uso 8, foi implementando a busca, por um termo qualquer a ser informado pelo usuário, em todas as entidades do sistema. Para isso a entidade BuscaPalavra foi criada contendo nela um ArrayList para armazenar todos os dados que continham o termo pesquisado, junto a isso, em cada entidade controladora do sistema foi implementado um método para pesquisar pelo termo informado nos dados armazenados retornando um ArrayList para BuscadorPalavra com os resultados da busca.

Para o retorno do resultado da pesquisa uma ordenação específica foi solicitada durante a implementação, para isso , foram criados comparadores de objetos para cada entidade controladora do sistema, podendo assim ordenar atividades, objetivos, pesquisadores, pesquisa e problemas de acordo com o solicitado na especificação.

Dentro da entidade BuscadorPalavra que ocorre o tratamento dos dados para retorno, a contagem da quantidade de resultados encontrados, o retorno de um resultado específico e a limpeza do ArrayList para uma nova pesquisa.

- **Caso 9:**

No caso de uso 9, foi trabalhado no package atividade fazendo alterações em Atividade, dando a ela a possibilidade de possuir uma ordem. Para que isso fosse possível, foram adicionadas próximas atividades. Isso acontecia por meio do método define próximo que, usando recursividade, usava do conceito de LinkedList para dizer quem seria o próximo mas, não permitia que uma mesma atividade tivesse dois próximos ou ainda que fosse criado um loop. Já no método tiraProximaAtividade, essa sequência seria quebrada, pois ao se tirar o próximo quebraria a sequência da lista a dividindo em dois.

O uso da recursividade esteve muito presente nessa US, bem como foi necessário o uso de métodos auxiliares para melhorar o design, evitando que os métodos ficassem com um corpo muito grande. Assim como nas demais US, só era permitido a finalização da execução do método após todos os parâmetros serem analisados e estarem em conformidade com o esperado. Tudo isso foi facilitado com a criação da classe Verifica, que continha os principais métodos de verificação de todas as demais classes.

- **Caso 10:**

A US10 traz dois novo métodos à facade, o “configuraEstrategia” e o “proximaAtividade”. Seus usos são bem simples, atendendo à necessidade de que o sistema possa sugerir uma próxima atividade (de uma pesquisa) a ser feita, baseado na estratégia de sugestão declarada. O configuraEstrategia dá 4 possibilidades de sugestão de atividade, a mais antiga, a com maior risco, a com menos pendências e a com maior duração. Assim, o método proximaAtividade usa dessa estratégia para sugerir uma atividade, percorrendo as atividades associadas à pesquisa declarada.

- **Caso 11:**

Na US11, é pedido a exportação de resumos e resultados das pesquisas para arquivos de texto que são guardados na pasta raiz do projeto. Depois da Facade, é o PesquisaController que tem a responsabilidade de invocar os métodos gravarResumo e gravarResultados.

Para isso, foram criados arquivos File, que carregam como nome o “codigo_da_pesquisa + .txt”, para resumos, e “codigo_da_pesquisa + -Resultados + .txt”, para resultados. Também foi utilizado FileWriter e BufferedWriter para a escrita dentro dos arquivos de texto. As Strings que representam os resumos e os resultados foram gerados dentro da própria entidade Pesquisa, já que ela é o expert da informação das informações relacionadas às pesquisas e a única, claro, que tem acesso a tudo o que está associado a si mesma (objetivos, problema, atividades e pesquisadores).

Também foi necessário que a entidade Atividade gerasse resumos e resultados próprios, já que somente ela tem acesso aos itens associados, e, também, foi necessário que cada Item possuísse duração própria, a qual é definida no momento em que a Atividade que possui determinado Item é executada.

- **Caso 12:**

No caso de uso 12 foi solicitado a implementação da persistência de dados do sistema, feita por meio da gravação de objetos em arquivos. Para isso a entidade SalvaSistema foi criada contendo métodos para criação de arquivos de dados e para a recuperação dos dados salvo em arquivos utilizando para isso ObjectOutputStream, ObjectInputStream e o FileWrite e FileReader.

Em cada entidade controladora do sistema foi implementado um método para salvar e recuperar dados responsáveis por fazer chamar os métodos necessários da entidade SalvaSistema.

Cada entidade tem seu próprio arquivo de dados, salvo em uma pasta, chamada ‘saves’, do sistema, evitando assim que caso o arquivo seja perdido todo o sistema fique impossibilitado de funcionar.