Relatório do Projeto LP2

Design geral:

O design de nosso projeto foi pensado e projetado com o objetivo de integrar seções distintas que desempenham determinadas funções, mantendo o baixo acoplamento e a alta coesão. Tal planejamento se desenvolve por meio de um controller geral que delega e coordena os gerenciadores, controllers menores que administram as partições.

Em relação a criação de objetos, foi adotado o uso de factories, afim de não sobrecarregar as responsabilidades dos gerenciadores. Além disso, foi preferível o uso de Enumerators ao padrão Strategy, tendo em vista a imutabilidade dos atributos e a catalogação.

As próximas seções detalham a implementação em cada caso.

Caso 1:

Neste caso, verificamos a necessidade de o administrador do sistema adicionar, recuperar, editar ou apagar participantes, onde para satisfazê-la, foi criado o objeto Pessoa, que possui atributos e métodos inerentes. O conjunto de pessoas é armazenado e manipulado por um dos controllers menores: GerenciadorPessoa.

Caso 2:

No caso 2, nos deparamos com uma entidade: os projetos. Estes possuem, apesar de características comuns, comportamentos característicos dos tipos de cada projeto. É pedido que o administrador do sistema adicione, recupere, edite ou apague projetos. A partir disso, foi criada a classe abstrata Projeto, superclasse de Monitoria, Extensão, PET e P&D, esta última, pai de duas outras classes: ProgramaInst, que é catalogado como PIBIC, PIBITI ou PIVIC, por meio de um Enum ([TipoProgramaInst](https://github.com/paulojbleitao/ProjetoLP2/blob/master/src/projeto/TipoProgramaInst.java)); e CooperacaoEmpresas.

Além da divisão em tipos, os projetos P&D possuem Producao, um atributo que pode ser inventariado, através de um Enum (Produtividade), como Produção\_Tecnica, Produção\_Academica ou Patentes, cujo processo de criação é gerido por FactoryProducao. Devido à complexidade de criar projetos referenciados como P&D em relação as outras classificações, foi criada uma factory dedicada a este tipo (FactoryPED).

Caso 3:

Tal caso pede que associemos pessoas à projetos, e que o administrador do sistema possa adicionar, remover e pesquisar participações de pessoas em projetos. Para satisfazer a necessidade de uma classe que seja armazenada tanto por pessoas (como na forma de histórico) e de projetos (a fim de conter a lista de membros), foi criada a classe abstrata Participacao, superclasse de Graduando, Professor (que pode ser rotulado como Coordenador por meio de um boolean), PosGraduando (classificado como relacionado a Mestrado ou Doutorado através de um Enum, TipoPosGraduando) e Profissional (classificado como Desenvolvedor, Gerente ou Pesquisador por meio de um Enum, TipoProfissional).

A criação dos objetos que tem como referência Participacao é realizada por uma factory (FactoryParticipacao). Já a associação de pessoas à projetos, feita a partir da administração de objetos de Participacao, feita por um controller: GerenciadorParticipacao.

Caso 4:

Neste caso, foi imposto a necessidade de se calcular os pontosde participação que uma pessoa já acumulou durante sua permanência em projetos. Tendo em vista que a classe Pessoa é capaz de deduzir os pontos acumulados a partir de suas participações em projetos e que cada tipo de Participacao possui determinada lógica de pontuação, foi implementado um método na classe Pessoa que itera sobre as participações dessa pessoa e sobre cada uma delas, calcula a pontuação, retornando o total.

Caso 5:

Este caso tem como enfoque o cálculo das bolsas dos participantes em projetos, baseado nas horas semanais e no valor da carga horária, que fazem parte de uma lógica própria para cada tipo de Participacao. Levando isso em consideração, foi declarado um método abstrato em Participação e cada um de seus tipos sobrescreve e implementa esse método em seu escopo, calculando o valor das bolsas de acordo com sua respectiva lógica. Valor este que será armazenado pela pessoa que o recebe.

Caso 6:

Tratamos da gerência de recursos financeiros proveniente da colaboração dos projetos. Isso inclui consultar e atualizar a receita (quando houverem gastos) e consultar os projetos quanto a disponibilidade para atender as demandas extras. Considerando que o cálculo desse montante depende das características inerentes de cada projeto, decidimos por declarar dois métodos abstratos em Projeto que são sobrescritos e implementados pelos tipos de Projeto, de maneira que um calcula sua contribuição de acordo com sua respectiva lógica e o outro atualiza as despesas devidas.

Caso 7:

Tratamos de armazenar em arquivos de texto dois relatórios: um com todos os projetos dos quais uma pessoa colaborou e outro com as colaboraçoẽs feitas ao projeto que se deseja saber. Para isso mudamos a lógica do projeto de modo que a cada projeto adicionado ele possa contabilizá-lo e dar uma noção da ordem em que acontece essa adição. Isso fora pensado para ajudar a imprimi-los na ordem certa. Para a parte de “Colaborações recebidas e estado do caixa da unidade acadêmica” isso não fora necessário. Contudo ambos possuem os devidos tratamentos de exceções e afins.

Caso 8:

Para realizar o “salvar e carregar” do projeto, implementamos o serializable. Permitindo assim salvarmos a versão mais recente e por fim quando voltarmos ao programa ele irá usar da informação salva como base.

Link para o repositório no Git Hub:

https://github.com/paulojbleitao/ProjetoLP2