Relatório para o projeto de Laboratório de Programação 2
Central de Projetos

Explanação geral:
Busquei desenvolver o sistema pedido de forma a utilizar todos os conhecimentos aprendidos nas disciplinas de p2 e lp2 este período, focando o reuso de código e a legibilidade, para este tanto, utilizo-me principalmente de herança quando possível (nos casos pedidos é o que se adequa melhor), chamadas polimórficas quando possível e inclusive criando interfaces para possibilitar a chamada polimórfica quando não seria conveniente usá-lo por meio da herança visto que não todas as subclasses possuem aquelas características, e uma documentação extensa sobre o código privilegiando a legibilidade. Em alguns casos em que era possível o uso de polimorfismo preferi a implementação local, visto que esta tinha uma complexidade menor, privilegiando assim a legibilidade. Faço uso do junit para criação dos testes de unidade, de forma que abranja os mais diferentes tipos de casos de uso do programa, com uma grande gama de classes possuindo test cases. Aqui uma observação na organização de pacotes escolhida, prefiro deixar os tests case subdivididos em pacotes contendo o nome do pacote das classes a qual os testes são seguido de “Test” para cada pacote de classes do programa ser seguido de um pacote com seus testes. O uso de exceptions próprias para maior legibilidade de código, e uma classe de validações com métodos estáticos, evitando a repetição de código, também estam presentes no projeto.
Mais adiante, explicações detalhadas sobre cada caso de uso.

Caso de uso 1:
No caso de uso 1 pede que seja criado um CRUD(create, read, update, and delete ou analogamente criar, ler, atualizar e deletar) de pessoas, necessário para gerenciar as pessoas que venham a participar de projetos, mantendo assim um cadastro para cada pessoa com os atributos “cpf”, “nome” e “email”. Foi criado então a classe Pessoa, contendo esses atributos, nesta, métodos “getters” e “setters” e sobrescrita do “hashCode”, “equals” e “toString”.
Para gerenciar as pessoas cadastradas em nosso sistema, criamos a “PessoaController”, classe responsável por armazenar objetos do tipo Pessoa, fazendo uso de um ArrayList, oferece métodos para funções básicas no gerenciamento de pessoas como cadastrar, remover, pesquisar, editar informações sobre uma pessoa.

Caso de uso 2:
Desta vez é pedido CRUD para projetos, no qual nosso programa deva ser capaz de fazer o gerenciamento de projetos existentes em nossa unidade, desta forma todo projeto deve ter os seguintes atributos: nome, objetivo, data de início e duração. Além destes, projetos devem ter um código único atribuído a ele durante sua criação, além de uma forma de guardar as despesas que ele terá. Aqui opto por duas classes novas, uma responsável pela dinâmica na geração de códigos novos unicos, no qual chamo de GeradorCodigo , e outra para representar as despesas que um projeto pode ter, na qual chamo de Despesa, já que um projeto tem 3 tipos de despesas, cada tipo desses é um atributo(um double) na classe Despesa, possuindo cada um deles “getters” para acessar seu valor.

Existem tipos específicos de projetos, são eles: Monitoria, Extensão, P&D e PET. Como qualquer projeto criado pertence a um desses 4 tipos, e todos esses tipos compartilham os atributos da classe Projeto, torno-a abstrata e estendo-a ela a mais 4 classes, que são respectivamente esses 4 tipos específicos de projetos, cada um com características adicionais às padrão de todo projeto. Embora projetos do tipo “P&D” ainda se subdivida em 4, essas 4 subdivisões não possuem características que torne necessário um novo uso de herança.
Assim pessoas possui um controle, que armazena as pessoas cadastradas no sistema, projetos também possuem um controle, classe de nome “ProjetoController”, que também faz uso de um ArrayList e oferece funções básicas sobre o controle de projetos, como adicionar, remover e etc…
Aqui optei por fazer algumas escolhas interessantes, como o uso de um enum para servir de chave para um mapa que guarda as produtividades de P&D e PET, já que as produtividades possíveis são 3 apenas, um enum me certifica disso. O uso de duas interfaces, Rendimento e Impacto, para permitir chamadas polimórficas às classes que têm essas características, me provendo assim um reuso de código interessante. Além de da implementação do tipo Date, implementação própria, para suprir as necessidades de forma que eu controle neste projeto, e o uso de outra classe Periodo que apesar de ser usada somente por Monitoria me permitiu um design mais bacana de código que somente usar período como se fosse uma String.

Caso de uso 3:
Após ter os subsistemas de pessoas e projetos completos, é pedido agora para criar um mecanismo de união entre pessoa e projeto, representando que uma dada pessoa participa de um projeto, e armazenar essas associações, aqui tenho a classe “Participação”, que usa uma pessoa cadastrada no controle de pessoas e um projeto existente no controle de projetos, seguidos de atributos como duração da associação valor da hora que essa pessoa terá nesse projeto e quantidade de horas que ela vai se dedicar a ele, e armazena essas informações nos atributos dos objetos do tipo “Participacao”, que representa essa associação, para depois serem guardados num novo controle, o “ParticipacaoController”, que assim como os outros controllers, guarda as participações de pessoas em projetos, fornecendo operações como criar participação, passando nome de pessoa e código de projeto para os encontrar em seus respectivos controllers, e criar uma associação entre eles.
Existem 4 tipos de participações de pessoas em projetos diferentes, e para estes tipos, novamente faço uso de herança, tornando a classe Participacao abstrata.

Caso de uso 4:
Pede a implementação de um sistema de pontuação para pessoas que participam de projetos, com diferentes formas de cálculo dependendo do tipo de participação que a pessoa tiver, fornecendo para formas de aproveitar esses pontos dependendo igualmente de qual tipo de participação for.
Para isto, crio um método delegador na PessoaController que repassa a função de cálculo de pontos à classe pessoa (que segundo a especificação sabe calcular sua pontuação, nada mais que justo seguir o conselho), que a partir das formas de cálculo passadas na especificação gera uma pontuação final para a pessoa em questão.

Caso de uso 5:
Agora, o uso das informações de valor por hora e horas semanais em que a pessoa tem na associação com o projeto lhe rendem uma bolsa, aqui também utilizo do PessoaController para chamar a classe Pessoa para gerar esse valor de bolsa, entretanto, faço uma adição de um método abstrato em Participacao chamado geraBolsa(), para cada subclasse de Participacao implementar a forma de cálculo dependendo das exigências da especificação, uso aqui de uma chamada polimórfica, na qual pessoa chama a geraBolsa da participacao, e esta é redirecionada para qual o tipo esta participacao pertence, mudando assim o cálculo, retornando no final para pessoa esse double com o valor, que retorna para o controller, e finalmente a facade.

Caso de uso 6:
Pede a implementação da uasc, na qual recebe colaborações de cada projeto (dependendo das regras passadas pela especificação), e representa a unidade acadêmica do curso, fazendo uso desses recursos para melhorias e urgências que unidade vinher a ter, uso aqui uma classe chamada UASC, que a representa e tem atributos como receita (a disponível no momento) e total Gasto, que registra todo o valor gasto até o dado momento de colaborações provenientes dos projetos. Todo projeto sabe calcular sua colaboração para com a uasc, uso um método abstrato na superclasse Projeto, na qual cada subclasse de Projeto o implementa usando a lógica necessária para ele, tendo aqui uma chamada polimórfica aplicada. Na especificação não ficou claro a respeito dessa parte, mas criei um sistema de flag chamado checkingUASC na qual projetos que já contribuíram se tornam true, para evitar computar eles em futuras checagens de colaborações para a uasc, evitando uma nova doação de um projeto que já contribuiu, a menos que ele seja alterado para o grupo dos “false” que são incluídos nessas checagens.

Caso de uso 7:
Pede a geração de relatórios sobre os projetos cadastrados na central de projetos e o histórico de colaborações para a uasc. Aqui uso arquivos de texto no qual esses relatórios são exportados para uma pasta específica dentro da estrutura de diretórios do projeto, como pedido, utilizando as técnicas aprendidas em sala sobre manipulação de arquivos.
Para isso escolho a classe UASC para essa implementação, que aliada ao controller de projeto para fornecer as informações, organizam numa String para ser escrita no arquivo, uso aqui também uma classe de utilidade responsável por trabalhar com arquivos, classe esta do pacote myUtils de nome EntradaSaidaDados, que é usada no caso de uso 8 também.

Caso de uso 8:
Aqui precisamos criar uma persistência de dados no nosso sistema, utilizo-me principalmente da classe ObjectInputStream e ObjectOutputStream do pacote java.io para salvar os objetos criados durante a execução do programa, optei por salvar um objeto somente, que é um do tipo UASC, já que representa o público destinado do programa, nada mais justo que a uasc possua os controllers com todas os registros de pessoas, projetos e associações feitas até o dado momento, tendo eles como atributos, salvo somente a uasc

num arquivo com o writeObject da classe ObjectOutputStream, e para a partir daí ao precisar reiniciar o sistema leia do mesmo arquivo, conservando assim o estado anterior.

Gabriel Fernandes da Silva
Ciência da Computação - UFCG
Campina Grande, 2 de abril de 2017