MAC0321–Exercício Programa 2 em Duplas
Gerenciador de Finanças Pessoais

Marcelo Finger

5 de junho de 2024

Instruções

Nome do projeto java: EP2-NUSP1-NUSP2. Sendo a sequência de NUSP de tamanho no máximo 2 dos Números USP dos membros da equipe.
Arquivo de envio: um ÚNICO projeto java zipado, ou seja, enviar o arquivo EP2-NUSP1-NUSP2,zip. *Não enviem arquivos RAR, que não faz parte do padrão Java.*

Seguir as seguintes instruções, sob pena de desconto de nota por código confuso:

- Esse EP pode ser feito em duplas. Neste caso: (i) inserir ambos os NUSPS no nome do projeto; e (ii) cada membro da dupla deve submeter uma cópia idêntica do projeto no eDisciplinas.
- Não enviar arquivos soltos.
- Não enviar mais de um projeto Java. Não enviar arquivo no formato RAR.
- Não enviar um projeto com muitas subpastas fazendo com que dê trabalho para encontrar o exercício.
- Não enviar rascunhos dos exercícios. Envie apenas os arquivos que vocês efetivamente querem que sejam corrigidos para que não corram o risco de corrigirmos os arquivos errados.
- Colocar cada exercício dentro de um pacote diferente (o pacote fica dentro do projeto java).
- Separar a classes funcionais das classes de testes deixando-as em arquivos diferentes.

Introdução

Neste exercício, vamos modelar e implementar um sistema que gerencia finanças pessoais, armazenando informações em planilhas e lendo informações destas planilhas. Nesta fase do projeto vamos utilizar apenas planilhas no formato CSV (*comma separated values*, ou valores separados por vírgulas).

Este sistema gerencia os lançamentos (as entradas de dinheiro e os gastos) de um grupo de pessoas próximas, tipicamente pessoas que moram numa mesma residência e partilham as receitas, chamado aqui de lar, de tal forma a gerenciarem seus gastos conjuntamente também. Tipicamente este é o caso de uma unidade familiar.

Os elementos principais do sistema resultantes da modelagem são os seguintes.

Usuário que é identificado por um apelido, em geral uma palavra curta, e possuem um nome. Por exemplo, o usuário "Pai" tem nome "Epaminondas Encerrabodes Esteves".

Tipos de Despesas que se referem às categorias e subcategorias de gastos realizados pelos membros de um mesmo lar. Um determinado tipo de despesa pode ter várias subcategorias. Por exemplo, o tipo de despesa `Educação` pode ter como subcategoria `Escola` e também `Curso de inglês`.

Tipos de Receitas que se referem às categorias e subcategorias de entradas financiaras recebidas pelos membros de um mesmo lar. Um determinado tipo de receita pode ter várias subcategorias. Por exemplo, o tipo de receita `Investimento` pode ter como subcategoria `Rendimento de renda fixa` e também `Dividendo de ação`.

Lançamento que representa uma atividade financeira de algum elemento do lar, seja ele uma receita ou um gasto. Um lançamento, possui um identificador único (um contador que só aumenta), uma data, um identificador binário R/D se é receita ou despesa, um usuário responsável pelo lançamento, um campo subcategoria (de receita ou despeza), um campo de descrição (texto livre) e um valor. Por exemplo, um lançamento identificado pelo número 1234321, no dia 01/02/2024, feito pelo usuário "Pai", foi uma despesa, na subcategoria "Escola", com descrição "Escolinha do Zezé", e valor R$500,00.

1 Leitura dos dados

Os dados devem ser lidos de arquivo CSV. Para isso você pode utilizar o método de `BufferedReader`, ou pode usar diretamente um pacote como OpenCSV, a qual se encontra disponível no sourceforge (recomendado), com documentação sobre utilização, download, classes e funcionalidades disponíveis também no `sourceforge`.

Imaginando que alguma hora o método de armazenagem pode migrar para um meio mais profissional como um banco de dados, vamos implementar a leitura dos dados através de uma interface que implementa o padrão DAO. Considere para tanto a seguinte interface `LeitorFinancasPessoaisDAO.java`:

```
1  package usp.mac321.ep2;
2  import java.util.List;
3
4  public interface LeitorFinancasPessoaisDAO {
5    public List<Usuario>   leUsuarios(String nomeArquivoUsuarios);
6    public List<TipoDespesa> leTiposDespesas(String nomeArquivoTiposDespesas);
7    public List<TipoReceita> leTiposReceitas(String nomeArquivoTiposReceitas);
8    public List<Lancamento> leLancamentos(String nomeArquivoLancamentos);
9  }
```

Essa interface é fornecida no projeto em anexo, bem como as cascas das classes mencionadas nela. A interface não deve ser alterada, mas as demais classes devem ser preenchidas. Seu código deve apresentar uma classe que implementa esta interface de leitura.

Sua leitura deve garantir que os dados estão coerentes, caso contrário a operação de leitura deverá jogar exceções que você mesmo deve implementar (exceto se a operação no arquivo jogar uma exceção).

A leitura dos dados deve jogar exceções ao menos nos seguintes casos:

- Arquivo não encontrado.
- Dois usuários com o mesmo apelido.
- Dois lançamentos com o mesmo identificador.
- Um lançamento de receita estiver relacionado a uma categoria de despesa, e vice-versa.
- Um lançamento estiver associado a um usuário que não existe.
- Um lançamento estiver associado a um valor negativo.

Notem que estas regras implicam que os arquivos devem ser lidos em uma determinada ordem. Junto com o projeto fornecemos planilhas para testar a leitura correta e a leitura com problemas dos arquivos csv. Também fornecemos um arquivo de teste com alguns testes para serem usados com os arquivos que fornecemos, `TestaLeitorFinancasPessoais.java`. Nessa classe de teste JUNIT, há testes de detecção de erro de entrada que devem ser completados por vocês.

2 Escrita dos dados

Implementar uma interface no padrão DAO que contém métodos de escrita para cada um dos arquivos lidos no item anterior. Não implemente nenhuma classe para esta interface ainda.

Pesquisar na internet sobre o princípio de *Segregação de Interfaces*. Este princípio faz parte de uma série de boas práticas de desenvolvimento de software chamado de desenvolvimento SOLID. Pesquise na internet sobre este conjunto de boas práticas e tente aplicá-las no desenvolvimento deste programa (e dos demais programas que você vier a escrever na sua vida). Note que a segregação de interfaces consiste no item I do acrônimo SOLID.

Baseado no princípio da Segregação de Interfaces, refatorar a interface única de escrita acima em diversas interfaces. Implemente as funcionalidades de armazenamento por meio de delegação de parte importante desta competência para cada uma das classes cuja casca foi fornecida. Gere uma nova classe de testes, em que arquivos CSV são lidos e escritos; seu teste deve verificar que os arquivos são idênticos ou ao menos correspondentes.

Implementar uma classe JUnit5, `TestaEscritaPlanilhas`, em que cada uma das planilhas é lida, alguns valores são adicionados, e a escrita é verificada. Use ao menos um teste por escrita de planilha.

3 Lidando com Lançamentos

Pesquise na internet sobre o padrão de desenvolvimento de software comportamental chamado State (Estado)[1]. Vamos agora usar este padrão para lidar com os lançamentos.

Um lançamento pode estar em um dos seguintes estados:

Executado quando a data do lançamento estiver no passado.

Planejado quando a data do lançamento estiver no futuro.

Inválido alguma regra de coerência mencionada na leitura dos dados.

Implemente a classe `Lancamento` de forma que consiga editar um lançamento existente, através de diversos métodos de edição. A classe também deve informar o estado de um lançamento a cada instante.

Implementar uma classe JUnit5, `TestaLancamentos`, em que um mesmo lançamento passa por todos os estados descritos acima e por todas as transições possíveis. Teste também outras funcionalidadesw da classe `Lancamento` que julgar importante.

4 Funcionalidades do Gerenciador de Finanças Pessoais

O `Gerenciador` deve possuir as seguintes funcionalidades.

- Criar e remover usuários, tipos de despesa e tipo de receitas.
- Criar e editar lançamentos.
- Salvar em arquivos csv o estado atual do gerenciador, com todos os usuários, (sub)categorias de despesas e receitas, e todos os lançamentos válidos.
- Para um dado intervalo de tempo, computar o valor total de despesas e receitas.
- Para um dado intervalo de tempo, e um tipo de despesa ou de receita, computar o valor agregado apenas dos lançamentos válidos do referido tipo.
- Em um dado intervalo de tempo, para um tipo de subcategoria de despesa ou de receita, computar o valor agregado apenas dos lançamentos válidos desta subcategoria.

[1] O exercício do EP1 é uma implementação sofisticada de um simulador que implementa este padrão, mas como aqui não há necessidade de simular nada, a sua implementação pode ser bem mais simples

- Imprimir um Relatório Finanaceiro, contendo os acumulados de todos os tipos de receita e despesa, para um dado intervalo de tempo.

Implementar a classe `Gerenciador`, que implementa essas funcionalidades.

Implementar uma classe JUnit5, `TestaGerenciado`, onde os testes verificam o seguinte comportamento: o `Gerenciador` imprime o Relatório Finanaceiro conforme mencionado nas funcionalidades do `Gerenciador`, e o teste verifica que o relatório foi gerado adequadamente.

Você deve ter ao menos três testes de geração de Relatório Financeiro. Os dados, constantes[2], de cada um dos testes do Gerenciador devem contemplar as seguintes entradas:

- Teste 01: um tipo de despesa e um de receita;
- Teste 02: dois tipos de despesas e dois de receitas;
- Teste 03: três tipos de despesas e três de receitas;

Para cada um desses testes, o conjunto de usuários, tipos de receitas e tipos de despesas pode ser o mesmo. Recomenda-se que os dados de entrada para os testes sejam lidos de um conjunto de arquivos[3].

[2]Isto é, você **não** deve gerar dados aleatórios

[3]Se fizer assim, esses dados precisam ser enviados juntos, caso contrário não há como testar.