Roteiro para o Trabalho Prático

Abaixo encontram-se alguns exercícios que, se resolvidos corretamente, auxiliam na construção do trabalho prático. Nota: esta não é a única forma de resolver o trabalho prático. Programar é também uma tarefa criativa e pessoas diferentes utilizam maneiras diferentes para resolver os mesmos problemas. Certifique-se apenas que a sua maneira está dentro dos princípios e práticas da Orientação a Objetos (OO).

---

1) Escreva classes que representam os elementos do domínio do problema do inventário de mídias. Tais classes foram modeladas na questão 1 da prova parcial. Coloque-as em um mesmo pacote, representando o módulo de "domínio".

Não é necessário, por enquanto, preocupar-se com métodos. Represente primeiro os atributos e as associações das classes de domínio. Defina construtores baseados no que é e o que não é informação obrigatória para cada classe. Os métodos serão criados em exercícios posteriores.

Para associações com multiplicidade maior que 1, escolha uma das coleções apropriadas: `java.util.List` (se os elementos são indexados); `java.util.Set` (se não são indexados e não há repetição); ou `java.util.Map` (se são pares chave-valor).

2) Em um pacote diferente (representando o módulo principal do programa), escreva uma classe com método `main()` (classe principal) para processar os parâmetros de entrada do programa. As possibilidades são:

a) `-g generos.csv -p pessoas.csv -m midias.csv -e emprestimos.csv` (em qualquer ordem, os nomes dos arquivos podem mudar);
b) `--read-only -g generos.csv -p pessoas.csv -m midias.csv -e emprestimos.csv` (idem);
c) `--write-only`.

Ao final do processamento, seu programa deve verificar se: (i) a opção `--write-only` foi especificada; OU (ii) as opções `-g`, `-p`, `-m`, e `-e` foram, todas elas, especificadas com um nome de arquivo para cada uma. Se nenhuma destas condições (i ou ii) forem satisfeitas, seu programa deve imprimir:

```
Arquivos de entrada não especificados. Use: -g <arquivo> -p <arquivo> -m <arquivo> -e <arquivo>
```

Se uma das condições for satisfeita, seu programa deve imprimir na tela o que falta fazer para concluir o trabalho. Por exemplo, usando os cenários (a), (b) e (c), acima:

a) `Falta ler os arquivos generos.csv, pessoas.csv, midias.csv e emprestimos.csv e gerar os relatórios;`
b) `Falta ler os arquivos generos.csv, pessoas.csv, midias.csv e emprestimos.csv e serializar os dados em inventario.dat;`
c) `Falta ler o arquivo inventario.dat e gerar os relatórios.`

3) Crie uma classe para efetuar leitura dos arquivos de entrada. Seguindo o princípio de modularização, coloque-a em um módulo de "entrada e saída de dados" (em inglês, *input/output* ou I/O). Neste ponto você já tem 3 módulos: domínio, principal e entrada/saída.

Por enquanto não se preocupe com as inconsistências e trabalhe somente com testes que você sabe que não possuem erros nos dados (teste 01, por exemplo). As inconsistências serão tratadas mais à frente. A classe deve, no entanto, propagar exceções referentes a erros de entrada e saída e erros de formatação.

Para começar, crie um método que receba como parâmetro o arquivo de gêneros (cujo nome foi lido no exercício 2) e monte uma coleção de gêneros. Essa coleção pode ser armazenada em uma propriedade da classe e, posteriormente, disponibilizada para o programa principal por meio de um método específico (se for necessário).

Este exercício e os seguintes servirão para montar coleções de objetos. As seguintes observações são importantes:

1. Alguns dados dos objetos estão disponíveis em cada linha do arquivo (ex.: para gêneros, estão disponíveis código e nome. Utilize o construtor criado para a classe para passar os dados obrigatórios. Caso existam dados opcionais, crie métodos nas classes de domínio que recebam esses dados após a construção do objeto;
2. Ao montar uma coleção, você deve escolher entre lista, conjunto ou mapa. Caso o objeto sendo lido seja citado em outro arquivo (ex.: o código do gênero é citado no arquivo de mídias, o código da mídia é citado no arquivo de empréstimo, etc.), o ideal é utilizar mapas, pois eles proveem uma forma fácil de recuperar um valor (o objeto) a partir de uma chave (código, nome, etc.);
3. Alguns objetos se referem a outros por código (como citado acima). Neste caso, é preciso converter essa referência textual (em arquivos CSV não há outro jeito de criar essas associações) para uma ligação entre objetos, pois estamos no paradigma OO. Crie métodos nas classes de domínio para o gerenciamento dessas ligações[1];
4. Em alguns casos pode ser necessário acessar alguma propriedade específica de um objeto que já exista. Neste caso, deve ser respeitado o princípio de encapsulamento que não permite que acessemos atributos dos objetos diretamente (eles seriam privativos). Crie métodos para efetuar este acesso;
5. Por fim, mas não menos importante, utilize o *try-com-recursos* para efetuar leitura dos arquivos com Scanner.

Para testar seu método, incremente o programa escrito no exercício 2, fazendo com que ele crie um objeto dessa nova classe e execute o método que monta a coleção de

[1] Por exemplo, um filme possui vários atores. Portanto, a classe Filme terá uma coleção (lista, conjunto ou mapa) de objetos da classe Pessoa. Ao se criar um Filme, essa coleção é criada vazia. Posteriormente, ao processar a lista de códigos de pessoas em uma linha CSV referente a um filme, busque as pessoas em um mapa e use um método específico da classe Filme (ex.: `adicionarAtor()`) para associar os objetos, adicionando o objeto Pessoa na coleção de atores daquele filme. Um objeto **não deve** referenciar outro objeto por meio de códigos, pois isso não obedece os princípios da Orientação a Objetos.

gêneros. Trate as exceções descritas no início do exercício (entrada e saída e formatação) em seu programa principal, da forma como foi estabelecido na seção 2.4 da especificação.

4) Ainda na classe de leitura dos arquivos de entrada, crie um método que receba como parâmetro o arquivo de pessoas e monte a coleção de pessoas. Siga as mesmas diretrizes do exercício 3, testando seu método ao final.

Para melhorar: verifique se há código que é repetido no método dos exercícios 3 e 4 e neste (todos eles leem arquivos CSV). Evite repetição de código criando um método genérico que efetua os passos que ambos têm em comum.

5) Ainda na classe de leitura dos arquivos de entrada, crie um método que receba como parâmetro o arquivo de mídias e monte a coleção de mídias. Siga as mesmas diretrizes do exercício 3, testando seu método ao final.

6) Ainda na classe de leitura dos arquivos de entrada, crie um método que receba como parâmetro o arquivo de empréstimos e monte a coleção de empréstimos. Siga as mesmas diretrizes do exercício 3, testando seu método ao final.

7) Identifique as possíveis inconsistências nos dados e faça com que a classe de leitura lance uma exceção (você pode escolher uma na API do Java ou criar uma exceção nova) nestes casos, tratando-a no programa principal conforme descrito na seção 2.4 da especificação. Alguns exemplos de inconsistências possíveis:

- No arquivo de mídias, usar algum valor diferente de 'L', 'F' ou 'S' na coluna Tipo;
- No arquivo de mídias, usar um código de gênero que não existe;
- No arquivo de mídias, usar um código de pessoa que não existe;
- No arquivo de empréstimos, usar um código de mídia que não existe.

Para testar seu exercício, utilize testes que possuam inconsistências ou crie alguma inconsistência nos arquivos de entrada do teste 01.

8) Neste ponto do roteiro, você deve ter acesso a coleções de objetos que estão interligados entre si e representam os elementos do domínio do problema descritos nos arquivos de entrada de dados. Em sua classe principal, implemente os cálculos necessários para gerar os relatórios de saída:

a) Horas consumidas: soma da quantidade de minutos das mídias (exceto livros) que já foram consumidas;
b) Horas a consumir: soma da quantidade de minutos das mídias (exceto livros) que se deseja consumir;
c) Mídias por gênero: para cada gênero do cadastro de gêneros, somar quantas mídias daquele gênero há no cadastro de mídias;
d) Temporadas por série: para cada título de série, somar quantas temporadas já foram consumidas e quantas se deseja consumir;

e) Mídias por pessoa: para cada pessoa que tenha participação em alguma mídia, montar uma lista das mídias das quais participa. Note que se há uma pessoa no cadastro que não participa de nenhuma mídia, ela não deve aparecer neste cálculo;
f) Atrasos nos empréstimos: para cada empréstimo, verificar se ele está atrasado (considerando a data 06/11/2015) e calcular quantos dias ele está atrasado;
g) *Wishlist*: não há nada a calcular, porém é preciso separar os itens que se deseja consumir e que não se possui.

Para este exercício, desconsidere os requisitos de ordenação impostos pela especificação do trabalho. Eles serão preocupação do próximo exercício. Além disso, utilize de polimorfismo para evitar ter que verificar o tipo de um determinado objeto para poder efetuar um cálculo. Por fim, lembre-se de avaliar qual classe deve ser responsável por efetuar o cálculo e retornar o valor já calculado.

Para testar, incremente o programa escrito no exercício 2, fazendo com que ele imprima na tela os dados dos relatórios que foram calculados. Os exercícios 10 a 13 servirão para escrever os dados nos arquivos.

9) Considere agora os requisitos de ordenação da especificação do trabalho, descritas na seção 2.3. Utilizando a interface `Comparable`, métodos de ordenação (e.g., `Collections.sort()`) e/ou coleções ordenadas (e.g., `TreeSet`, `TreeMap`), para ordenar o resultado dos cálculos feitos no exercício anterior.

Neste momento pode ser necessário reavaliar as estruturas usadas nos cálculos do exercício anterior. Avalie se é possível obter o resultado desejado utilizando `Comparable` na própria classe de domínio ou se é necessário criar uma outra estrutura para ordenar os dados do relatório. Evite usar estruturas que fujam da Orientação a Objetos.

Teste o exercício da mesma forma que o exercício anterior.

10) Crie uma classe para efetuar escrita dos arquivos de saída. Ela pode ficar no mesmo módulo de "entrada e saída de dados" no qual você colocou a classe que efetua a leitura dos arquivos de entrada.

Crie um método para gerar o relatório de estatísticas. Verifique quais objetos e coleções (dentre os que foram criados até este exercício) são necessários para gerar esse relatório e indique-os como parâmetros desse método.

Observações importantes: (i) utilize o *try-com-recursos* para abrir o arquivo de saída para impressão; e (ii) na escrita dos relatórios também é possível obter um "Erro de I/O" (seção 2.4 da especificação), portanto propague as exceções necessárias e trate-as no programa principal, como feito na leitura de dados.

Para testar seu método, incremente o programa escrito no exercício 2, fazendo com que ele execute o método passando os parâmetros necessários.

11) Ainda na classe de escrita dos arquivos de saída, crie um método para gerar o relatório de mídias por pessoa. Siga as mesmas diretrizes do exercício 10, testando seu método ao final.

12) Ainda na classe de escrita dos arquivos de saída, crie um método para gerar o relatório de empréstimos. Siga as mesmas diretrizes do exercício 10, testando seu método ao final. Lembre-se que datas devem ser impressas em formato dd/mm/yyyy. Utilize `printf()` ou formatadores associados a um objeto `Locale` representando a configuração regional "português do Brasil".

13) Ainda na classe de escrita dos arquivos de saída, crie um método para gerar o relatório *wishlist*. Siga as mesmas diretrizes do exercício 10, testando seu método ao final. Lembre-se que preços devem ser impressos com duas casas decimais, separado por vírgula e precedido de "R$ ". Utilize `printf()` ou formatadores associados a um objeto `Locale` representando a configuração regional "português do Brasil".

14) Por fim, vamos ao ponto extra: a serialização. Métodos para serializar e desserializar podem ser adicionados à classe que efetua a escrita dos arquivos de saída, criada no exercício 10. O código de serialização/desserialização é simples de fazer: basta observar os últimos slides da parte 10 do curso básico de Java.

O que é preciso definir aqui é o que serializar: quais objetos você criou durante a leitura dos arquivos de entrada e que serão necessários para geração dos arquivos de saída? Estes objetos devem ser serializados.

Outra coisa que deve ser definida é o ponto onde deve ocorrer a serialização e a desserialização. Volte ao exercício 1 e à sua classe principal e identifique estes pontos. Quando é especificada a opção `--read-only`, os arquivos de saída não devem ser gerados. Por outro lado, com `--write-only`, os arquivos de entrada não devem ser lidos (apenas o `inventario.dat` deve ser usado). Ajuste seu programa principal de maneira adequada.

Lembre-se: se precisar de ajuda, entre em contato com o monitor da disciplina Eduardo França (edox86@gmail.com) e marque um horário com ele para atendimento. Caso ele não esteja disponível, marque um horário comigo pelo site https://vitorsouza.youcanbook.me.

Bons códigos!