PUC Minas

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Lista 06 - Ordenação Parte 1

Professora: *Marta Noronha*

Disciplina: *Algoritmos e Estrutura de Dados II*
Data de entrega: *21/04/2024*

Requisitos

1. Todos os programas deverão ser desenvolvidos na linguagem de programação Java.
2. Essas práticas poderão ser desenvolvidas em grupos de, no máximo, dois integrantes.
3. Cópias, se existirem, serão encaminhadas ao colegiado de coordenação didática do curso.
4. Fique atento ao charset dos arquivos de entrada e saída. Recomenda-se a utilização dos métodos da classe MyIO.java para leitura de dados do teclado. É necessário definir o charset a ser utilizado antes de começar a leitura de dados do teclado, da seguinte forma: `MyIO.setCharset("UTF-8")`.
5. As saídas esperadas, cadastradas no VERDE pelo professor, foram geradas empregando-se: `System.out.println()`.
6. Em cada submissão, enviar apenas um arquivo (.java). A regra será necessária para a submissão de exercícios no VERDE e no identificador de plágios utilizado na disciplina.
7. A resolução (código) de cada exercício deverá ser submetida ao VERDE.
8. A execução do código submetido será realizada automaticamente pelo VERDE, mas o código será analisado e validado pelo professor.
9. Se for necessário ler o arquivo *"pub.in"*, o mesmo será disponibilizado para *download* no VERDE juntamente com o arquivo *"pub.out"*.

Exercícios - Criação da classe Jogos

O conjunto de dados utilizado neste exercício foi disponibilizado em Kaggle.
Crie uma classe Jogo com os seguintes atributos declarados com modo de acesso privado:

- `rank` (int): classificação (posição) do jogo em relação aos demais.
- `nome do jogo` (String): o nome do jogo.
- `plataforma` (String): Onde o jogo pode ser executado (Wii, DS, entre outros).
- `ano` (int): ano de lançamento.
- `gênero` (String): esportes, corrida, entre outros.
- `editora` (String): empresa que disponibiliza o jogo (produtora).
- `NA_Vendas` (double): Vendas do jogo na América do Norte.

- `EU_Vendas` (double): Vendas do jogo na Europa.
- `JP_Vendas` (double): Vendas do jogo no Japão.
- `Outras_Vendas` (double): Vendas do jogo em outros lugares.
- `Vendas_Global` (double): Vendas global do jogo.

A classe Jogo também deve conter, obrigatoriamente, ao menos, dois construtores (*1. padrão (default); 2. nome_do_jogo/plataforma/ano*), e os métodos *gets*, *sets*, métodos `clone()`, `ler()`, `imprimir()` e `toString()`.

O método `clone()` deve retornar um objeto da mesma classe e contendo os mesmos valores de atributos do atual objeto analisado (**vide exemplo do método clone no slide "Unidade 0 - Nivelamento - Ponteiros (C/C++) e Referências (Java)**" postado na disciplina teórica). **Não é permitido usar a interface Cloneable para esta finalidade.**

O método `ler()` deve receber cada linha do arquivo como parâmetro e armazenar os valores contidos em cada linha nos atributos de cada objeto que foi instanciado.

O método `imprimir()` exibe os valores dos atributos do objeto Jogo, conforme o modelo indicado no fim deste documento, conforme mostrado AQUI.

O método `toString()` deve ser criado para permitir a impressão dos atributos da classe, sem necessidade de invocação do método `imprimir()`.

> Os métodos que não obedecem a forma como devem ser implementados, conforme descrição acima, serão penalizados ainda que o envio no Verde contabilize 100% de acerto. Isso será feito porque o desenvolvedor deve seguir atentamente as regras de implementação de um método, quando houver, considerando os parâmetros e o tipo de retorno que um método deve possuir quando estes são solicitados. Em caso de dúvidas, consulte a Professora.

Após a criação da classe, deve ser criado um mecanismo para processamento de uma entrada de dados. A entrada de dados é dividida em 2 partes:

- Parte 1: Armazenamento de informações em vetor;
- Parte 2: Pesquisa de informações armazenadas no vetor criado na parte 1.

Parte 1 (Leitura de Arquivo): Armazenamento de informações contidos no arquivo jogos.txt em vetor

Seu programa deve ler um arquivo-texto chamado **"jogos.txt"** que, no VERDE, localiza-se na pasta **"/tmp"**.

O aluno(a) deve preencher um vetor de objetos da classe Jogo com os dados dos diversos jogos informados no arquivo "jogos.txt". Use um ArrayList<Jogos> para aumentar dinamicamente o *array* contendo os jogos que serão incluídos. Consulte um guia disponível em Java ArrayList para compreender o funcionamento do *ArrayList*.

Cada uma das linhas seguintes apresenta os dados de um jogo, separados pelo símbolo "|" (Use " \\|" para fazer o split do vetor). Os dados possuem, em ordem, as seguintes informações:

- `rank` (int);
- `nome do jogo` (String);
- `plataforma` (String);
- `ano` (int);
- `gênero` (String);
- `editora` (String);
- `NA_Vendas` (double);
- `EU_Vendas` (double);
- `JP_Vendas` (double);
- `Outras_Vendas` (double);
- `Vendas_Global` (double);

A última linha do arquivo é vazia.

Parte 2: Pesquisa de informações armazenadas no vetor (ArrayList) criado na parte 1.

Após o processamento da primeira parte da entrada de dados, o programa deve processar a entrada contida no arquivo *pub.in*. Cada linha da segunda parte contém, em ordem, as seguintes informações:

- nome do jogo;
- ano;
- editora;
- plataforma.

A última linha do arquivo *pub.in* contém a palavra FIM.

As entradas devem ser pesquisadas no vetor de jogos. Armazene as entradas em um outro array, que posteriormente será ordenado. Para esta tarefa, é recomendado o uso de um ArrayList.

O programa deve contabilizar a quantidade de jogos lidos na entrada padrão, neste caso *pub.in*, que **foram** encontrados no arquivo *jogos.txt*. Ao final do arquivo, deverá ser impresso a informação:

Quantidade de jogos encontrados: {total de jogos}
(Obs: as chaves não devem ser impressas)

Parte 3: Ordenação do vetor armazenado na Parte 2.

Nesta etapa será feita a ordenação do vetor de registros pesquisados, utilizando 3 algoritmos de ordenação:

- Bubble sort;
- Selection sort; e,
- Insertion sort.

Parte 3.1: Critérios de ordenação. A ordenação de elementos no vetor será feita com base em atributos da classe Jogo. Estes atributos devem ser utilizados para comparar se um dado jogo é maior/menor que outro de acordo com os seguintes critérios que são utilizados para ordená-los:

- Nome do jogo (crescente);
- Plataforma (crescente);
- Vendas global (decrescente).

Para que os métodos utilizem a mesma entrada, devem ser gerados 3 clones do array original. Cada clone do array será utilizado para ordenação por meio do uso de um dos métodos de ordenação descritos acima.

Após o processamento de um dado método de ordenação, devem ser impressas as informações dos jogos ordenados, segundo critério previamente informado.

Para cada jogo, deve ser escrito na saída padrão uma linha contendo as informações do jogo, com os dados do registro correspondente. Além da venda global do jogo, uma função deve ser implementada para informar em que região ou país o jogo foi mais vendido (NA_vendas, EU_Vendas, JP_Vendas ou Outras_Vendas). Caso mais de uma região ou país tenha tido o mesmo número de vendas, informar somente o primeiro encontrado com a maior venda. A saída padrão deve obedecer o seguinte formato (*não incluir chaves*):

[*Nome do jogo*] [*Plataforma*] [*Vendas global*] {*Rank*}. {*Gênero*}. {*Editora*}. Mais vendido:{*País ou região mais vendido*}.

Este formato de saída deve ser implementado utilizando o método `imprimir()` mencionado na Parte 1 deste enunciado.

Exemplo de uma possível saída ordenada:

[*Dynasty Warriors: Gundam 2*] [*PSV*] [*0.06*] *12452. Adventure. Aksys Games. Mais vendido: NA_Vendas.*
[*Dynasty Warriors: Gundam 3*] [*PS*] [*0.67*] *3229. Role-Playing. Square. Mais vendido: JP_Vendas.*
[*Dynasty Warriors: Gundam 3*] [*PS*] [*0.63*] *3228. Role-Playing. Square. Mais vendido: JP_Vendas.*

Dicas (não obrigatórias):

- Crie *métodos para comparação de objetos ("ehMaior" e "ehMenor"), contendo os critérios indicados;*
- Crie uma *classe para ordenação, contendo os 3 algoritmos (bubble, selection, insertion);*
- *Altere operações dos algoritmos de ordenação (">" ou "<") para os métodos "ehMaior" e "ehMenor".*

Parte 3.2: Resumo da ordenação.
Durante o processo de ordenação, devem ser armazenadas as seguintes métricas:

- Número de comparações entre jogos;
- Número de movimentações (trocas) de posições no vetor.

Ao fim da execução de cada um dos algoritmos de ordenação, deve ser exibido um resumo do número de comparações e movimentações executadas. O resumo deve ser impresso com o seguinte padrão:

`## {nome_metodo} [COMPARACOES] [{num_comparacoes}] [MOVIMENTACOES] [{num_movimentacoes}]`

Onde:

- "NOME-METODO" = SELECTION ou BUBBLE ou INSERTION;
- "{num_comparacoes}" = número de comparações divididas por 1000, sem casas decimais;
- "{num_movimentacoes}" = número de movimentações divididas por 1000, sem casas decimais.

Após cada valor do número de comparações e movimentações, acrescentar a letra "k", para indicar que a contagem de números está em função de 1000.
Exemplo de saída:

```
## BUBBLE [COMPARACOES] [7k] [MOVIMENTACOES] [8k]
```

Após a conclusão do trabalho, observe a quantidade de comparações e movimentações realizadas por cada um dos algoritmos. Analise os resultados e avalie se os mesmos estão de acordo com o esperado, a partir do estudo da disciplina teórica.
Dicas (não obrigatórias):

- *Crie uma classe log, para gerenciamento da contagem de operações;*
- *Conte cada um das cláusulas de comparação (IF).*
- *Quando houver IFs aninhados, não se esqueça de contar as cláusulas anteriores, que já foram avaliadas.*
- *A criação de um método único para comparação ("ehMaior" e "ehMenor") facilita o processo de comparação.*