

EP2 - Gerador de Relatórios

Autor: André Morales de Oliveira Carneiro - **14558332**

Compilação e Execução

Para compilar, execute o compilador na classe principal.

```
> javac -d output -cp "src;" src/gerador/MainCLI.java
```

ou

```
> make build
```



Para executar:

```
> java -cp "output;" gerador/MainCLI  
    <algoritmo>  
    <ordenacao>  
    <filtro>  
    <parametro>  
    [formatadores...]
```

Modificações

◆ Pacotes

Dado que muitas classes precisaram ser implementadas para várias diferentes razões, separei o projeto em 5 pacotes:

 Pacote	 Função
gerador	Classes principais do gerador de relatórios, incluindo o próprio gerador e a classe Main
criteria	Critérios de ordenação usados
filters	Todos os filtros possíveis
formatters	Os herdeiros de IFormatter incluindo o próprio IFormatter
sorts	Os dois algoritmos de ordenação em conjunto com ISortingAlgorithm

◆ List<> ao invés de []

As ocorrências de array substitui pela interface List<>. A interface List pareceu apropriada pois permite acesso posicional em qualquer ponto da lista, e esse fato era necessário para que os algoritmos de ordenação pudessem ser mantidos.

Ademais, os algoritmos de ordenação permaneceram quase intocados, assim como pedido, fazendo apenas as substituições mais necessárias para usar a interface List (substituindo acesso com [] por set/get).

◆ Estratégias para algoritmos de ordenação:

Cada algoritmo possível é uma estratégia diferente. Antes de utilizar uma estratégia de algoritmo, deve-se configurar o critério que o algoritmo deverá utilizar através do setCriterion().

Decidi fazer o objeto do algoritmo guardar qual critério ao invés de apenas receber como parâmetro pois um algoritmo de ordenação pode ser implementado em múltiplas etapas, como é o caso do QuickSort, e a passagem do critério entre as funções internas ficaria mais deslegante do que apenas guardar o critério dentro do objeto do algoritmo.

◆ Critério de ordenação:

Cada critério herda de `ICriterion`, que não passa de um alias para `Comparator<Produto>`. Esse comparador depois é utilizado diretamente pelo algoritmo de ordenação selecionado.

Os critérios aceitam como parâmetro um `enum OrderRule` que define se o critério deve ser crescente ou decrescente.

✨ **Critérios Decrescentes:** Os critérios decrescentes podem ser acessados usando 'd' ao invés de 'c' no nome do critério de ordenação.

◆ Estratégias de filtração:

Cada filtro possível agora herda de `IFilter`, que é uma interface que implementa a funcionalidade de um predicado. O filtro aprova ou não um produto se satisfazer o predicado. Junto com a funcionalidade de predicados, eu incluí em `IFilter` um método `getDescription()`, que só é utilizado na chamada de `debug()` para descrever a função do filtro.

✨ **PriceFilter:** A implementação do filtro do preço entre dois limites [a, b], deve ser invocada com o 'preco_entre' e um parâmetro string no formato <min>:<max>

✨ **DescriptionFilter:** Pode ser usado passando 'desc_contem' como filtro.

◆ Formataadores:

Os formataadores usam o padrão Decorator para imprimir no `PrintWriter`. Um formataador deve ser capaz de imprimir usando `format(stream, produto)`. Todos os formataadores com exceção do padrão aceitam uma base como parâmetro de construção, permitindo os

decoradores se aninharem recursivamente com as “decorações” necessárias.

Na CLI, a lista de formatadores pode ser o quão longa você desejar.

✨ **Formatador Estilizado:** O formatador estilizado pode ser invocado com ‘estilizado’ passado na sequência de formatadores.

Em específico verifica a hierarquia do produto buscando ProdutoEstilizado e apenas então aplica formatações ao produto na saída. Apesar do instanceof ter todos os sinais para um code smell, justifico a presença dele pois preferi manter o código de formatação justamente nos formatadores.

Uma alternativa seria fazer a formatação estilizada no próprio

`formataParaImpressao()` do produto estilizado, porém, isso resultaria em saída estilizada mesmo sem o formatador estilizado, o que derrota o seu objetivo em primeiro lugar.

A solução dada mantém compatibilidade com a interface Produto e qualquer combinação de decoradores (formatadores)

◆ Separação da CLI e do Gerador de Relatórios

A responsabilidade de processar os parâmetros do programa, e de julgar qual nome de estratégia é associada a qual objeto de estratégia passou a ser responsabilidade do MainCLI. A classe geradora de relatórios deve continuar preocupada apenas com a manipulação e uso dos objetos estratégia providenciados a ela.

Justifico isso com o fato de que a adição de um novo algoritmo, critério de ordenação ou outra opção de estratégia na CLI já é de se esperar que modifique a classe da CLI, mas não deveria modificar o gerador de relatórios.

◆ Leitor de CSV

O leitor está implementado em SheetReader e lê o csv linha a linha utilizando o Scanner() para cada linha. Essa função usa o locale US para lidar com os números double com ponto no separador.