**UNIVERSIDADE DE SÃO PAULO**

**ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES**

**ACH2003 - COMPUTAÇÃO ORIENTADA A OBJETOS**

**Relatório de Refatoração de código - Gerador de Relatórios**

**Gustavo Bastos de Souza - 14576642**

**Alterações**

Alterações baseadas no padrão Strategy e Decorator de acordo com a especificação da atividade proposta. Aplicação do padrão Strategy para: Uso de algoritmos de ordenação, critérios de ordenação e critério de filtragem; e aplicação do padrão Decorator para: opções de formatação para o arquivo de saída em HTML. Além disso, o respeito aos requisitos de que: *Collections* sejam utilizadas no lugar de vetores de tipos primitivos e a não modificação da interface *Produto* e a Classe *ProdutoPadrao.*

**Algoritmos de Ordenação**

Os algoritmos de ordenação foram “desacoplados” da classe *GeradorDeRelatorios*, por meio da criação de uma interface *Sort*, com apenas um método *void ordena(),* que usa como parâmetros a Coleção de Produtos, o critério para ordenar e índices de início e fim dessas listas para ordenar a Coleção de Produtos. Com isso, temos as implementações em classes *InsertionSort* e *Quicksort,* as mesmas presentes no início do código. QuickSort também emprega a função de retorno de inteiro *particiona()*, comum de sua implementação.

**Critérios de Ordenação**

Os critérios de ordenação presentes agora são implementados a partir da interface *Criteria*, com um método booleano *compareCriteria(),* que possui como parâmetros dois Produtos distintos. Desta forma são implementadas as classes: *AscendingDescCriteria(),* critério de descrição crescente,que retorna *true* caso a descrição do primeiro produto dado não for maior ou igual em ordem alfabética ao do segundo; *AscendingPriceCriteria(),* critério de preço crescente, que retorna *true* caso o preço do primeiro produto passado como parâmetro não for igual ou superior ao segundo produto; *AscendingStockCriteria(),* critério de estoque crescente*,* que retorna true caso a quantidade do estoque do primeiro produto não for maior ou igual ao do segundo produto. Esses três critérios anteriores já estavam presentes no programa (mas não na implementação em Classes), e além disso foram criadas as implementações: *DescendingDescCriteria(),* critério de descrição decrescente, que retorna *true* caso a descrição do primeiro produto parâmetro for igual ou superior em ordem alfabética ao do segundo produto; *DescendingPriceCriteria()*, critério de preço decrescente, que retorna *true* caso o preço do primeiro produto for superior ao do segundo produto; e *DescendingStockCriteria(),* critério de estoque descrescente, que retorna *true* caso a quantidade no estoque do primeiro produto for superior ao do segundo produto. Cada *true* true retornado indica nos algoritmos de ordenação quando deve-se realizar a troca de membros para dispor na ordenação requisitada do algoritmo.

**Critérios de Filtragem**

Os critérios de filtragem são implementados pela interface Filter, que possui o método booleano *select(),* que possui como parâmetros um produto e uma Coleção em *List* de *String* de parâmetros, pois um dos filtros (intervalo de valores) precisa de mais de um valor. Os critérios de filtragem presentes anteriormente no código, e agora implementados a partir dessa interface, são: *LessThanStockFilter(),* filtro de estoque menor-que, que retorna *true* se a quantidade no estoque daquele produto não for maior que a quantidade passada pelo parâmetro; *AllProductFilter(),* filtro de todos os produtos, que sempre retorna *true;* e *CategoryEqualsThanFilter(),* filtro de categoria igual, que retorna *true* se a categoria daquele produto for equivalente a do parâmetro passado. Não só tem-se esses filtros como outros foram adicionados, como: *PriceIntervalFilter(),* filtro de intervalo de preço, que retorna *true* caso o preço do produto estiver dentro de um intervalo especificado por dois parâmetros passados para o filtro; e *SubstringFilter(),* filtro de substring, que retorna *true* caso o produto parâmetro conter uma substring passada no parâmetro. O valor booleano define quais produtos serão selecionados para a saída do arquivo, isto é, que foram retornados como *true* dado o critério de Filtragem. Vale ressaltar que para o filtro de intervalo de preço o programa indica que o usuário deve digitado de forma crescente (exemplo: um intervalo de 10 a 100 deve ter os parâmetro passados como “10 100”, mas nunca como “100 10”)

**Critérios de Formatação**

A formatação dos produtos em negrito, itálico e/ou colorido se baseou no padrão Decorator. Para isso, foi criada a classe *FormattedProduct* que implementa *Produto*, que se diferencia de *ProdutoPadrao* apenas por receber em seu construtor um *boolean* que indica negrito, um *boolean* que indica itálico e um *String* que guarda a cor do produto (os produtos sem cores guardam a cor preta). E então foi criada a interface *Formatted,* com o método *void formatting(). DefaultFormatted*, uma das classes que implementa *Formatted,* realiza a formatação padrão. *FormattedDecorator* é o “decorador” do padrão, ou seja, à partir da forma que implementa *Formatted* serve como uma base para a chamada do método *formatting(). ItalicFormatted, BoldFormatted, ItalicAndBoldFormatted* e *ColoredFormatted* são as classes que estendem o decorator e que vão negrito, itálico e cor. Aqui é importante esclarecer que todas recebem uma cor, e uma forma de implementar o uso de cores ou não no relatório seria passar a devido cor do produto ou somente a cor preta.

**Mudanças na Classe GeradorDeRelatorios**

Esse classe sofreu alterações em vários pontos devido a busca por código mais legível, eficiente e menos redundante, considerando todas as alterações já explicadas.

No construtor, tem-se o uso de uma Coleção em Lista de produtos (*List<Produto> produtos*), uma Coleção em Lista de String (*List<String> argFiltro,* pois o argumento pode precisar de mais de um parâmetro), dois Booleans (*italic* e *bold)* e um tipo *Sort (algoritmo).*

O método *GeraRelatorio()* está mais sucinto. A ordenação é chamado pelo tipo do algoritmo passado no construtor, e não um método dentro da classe. E para cada produto, vamos transformá-los em um tipo *FormattedProduct, produto formatado.* Se o produto em questão passa pelo filtro, passamos por uma série de condicionais para verificar se é necessário levar em conta o negrito, itálico e a cor para o relatório, e temos então um objeto com a formatação que precisamos a partir de um outro padrão, *DefaultFormatted,* que no método *formatting()* chama o produto no seu formato para impressão.

O método *carregaProdutos()* agora devolve uma coleção em lista de produtos, mais especificamente um *FormattedProduct.* para isso usamos um scanner que vai ler o arquivo “produtos.csv”, passado como parâmetro na função. Apesar de todo o produto lido ser considerado um produto formatado, depende da entrada do usuário a aplicação destas configurações de formatação.

O método *main()* por sua vez agora tem de tratar casos da leitura de outros critérios, descritos anteriormente no relatório, e inclusive de mais argumentos passados.