Relatório - Computação Orientada a Objetos, EP3

Nome: Jaime Jean Soares Silva **Nº USP:** 11270761

Nome: Vinícius Santana Lima **Nº USP:** 11315029

Esse relatório visa descrever as alterações aplicadas na refatoração do programa **GeradorDeRelatórios**, trabalho para composição de nota da disciplina ACHXXXX - **Computação Orientada a Objetos**. Nele, abordaremos as práticas aplicadas para a substituição dos códigos e aplicação dos padrões de projeto **Strategy** e **Decorator**.

A classe GeradorDeRelatórios

Na classe GeradorDeRelatórios, os atributos **inteiros** que representavam *algoritmo, critério* e *filtro* foram substituídos por interfaces de mesmo nome. Assim, recebiam instâncias da classe desejada como argumentos pelo método construtor na instanciação.

Os métodos *particiona* e *ordena*

O método **ordena** continha uma verificação e blocos de código correspondentes ao algoritmo escolhido, anteriormente passado pelo inteiro. O comportamento de ordenação de cada cada algoritmo foi encapsulado no método **ordena** das classes **insertionSort** e **quickSort** . Assim, essa verificação já não era necessária, bastando apenas a chamada do método com os argumentos respectivos.

- Assim, passamos ao método **ordena** do algoritmo selecionado os inteiros **i e j**, o arranjo de produtos e o critério selecionado.
- Cada classe que implementa **Algoritmo** aplica seu algoritmo correspondente, acessando os métodos de verificação do critério correspondente em seu código.
- **Quick Sort —** Uma vez que o método **particiona** era utilizado apenas pelo algoritmo **quickSort**, ele foi movido para a classe do algoritmo.

- As verificações anteriormente feitas por IFs foram movidas para as classes de critério correspondentes por meio de métodos booleanos.

O método *geraRelatorio*

O método continha um laço *for* para iterar pelo arranjo `Produto[] produtos` e uma cadeia de ifs para verificar se o produto atual seria ou não selecionado. Essa cadeia foi substituída por um método de retorno booleano, cuja **estratégia** variava de acordo com o filtro escolhido. Assim, a cadeia de IFs foi substituída, e cabia a cada filtro validar a seleção do **produto**.

Formatações

- Para aplicar o **decorator,** foi criada uma classe para aplicar o **negrito**, o **italico** e a **cor**. Todas as 3 classes, assim como a classe **ProdutoPadrao** (empacotadora), implementam a interface **Produto** e possuem em comum um atributo do tipo **Produto** e sobrescrevem o método **formataParaImpressao()**.
- Para cada escolha diferente de formatação, seja negrito, italico ou negrito e italico, as classes utilizam seus métodos (que estão sobrescritos para cada tipo de formatação) para aplicar a formatação desejada ou combinar as formatações adicionando essas opções junto com a formatação padrão que está no metodo formataParaImpressao() da classe ProdutoPadrao. Ou seja, para cada chamada do método sobrescrito de uma classe de formatação, a classe ira aplicar o tipo de formatação mais a formatação padrão(por exemplo: negrito + texto padrão ou italico + negrito + texto padrão).