**RELATÓRIO (MODELO)**

**Responsável: André Cararo Lowcke**

Nesse relatório, encontram-se as propostas de melhorias no código do projeto de Supermercado.

**SUGESTÕES DA EQUIPE DE TESTE**

Abaixo, encontram-se os problemas identificados pela equipe de teste e as sugestões de melhorias propostas pelo responsável.

|  |
| --- |
| **SUGESTÃO #1: Nomeação adequada e legibilidade (Exemplo)** |
| **Arquivo:** CarrinhoDeCompras.java  **Código atual:**    **Sugestão de melhoria:**    **Justificativa**: os parâmetros “p” e “q” foram renomeados para “preço” e “quantidade” com o objetivo de deixar o código mais claro sobre o que está acontecendo. Além disso, o método foi simplificado para deixa-lo mais curto e melhorar a legibilidade. |

|  |
| --- |
| **SUGESTÃO #2: Métodos duplicados** |
| **Arquivo:** Cliente.java  **Código atual:**    **Sugestão de melhoria:**    Justificativa: Métodos getNome() e getNomeCliente() fazem a **mesma coisa, Ambos retornam o mesmo atributo** nome**. Isso é um exemplo claro de método duplicado e desnecessário. O mesmo ocorre com os setters, mesma funcionalidade, só muda o nome do método e do parâmetro. Então seguindo:**  **Clean Code: evitar código redundante e complexidade desnecessária. Refatoração: eliminar duplicações. SOLID — Single Responsibility: métodos duplicados indicam falta de clareza na responsabilidade do código.**  **Então para resolver mantive apenas um dos getters e setters. Se não seguir estes princípios teremos código mais difícil de manter.** Aumento do risco de inconsistência: se alterar setNome() e esquecer setNomeCliente(), surgem **bugs**. Não segue o princípio de **"código limpo"**, que recomenda métodos **simples, coesos e claros**. |

|  |
| --- |
| **SUGESTÃO #3: Substituição de atributo por objeto** |
| **Arquivo:** Pedido.java  **Código atual:**    **Sugestão de melhoria:**    **Justificativa**: Ter os dados do cliente já organizados dentro da classe Cliente evita repetições desnecessárias no código, já que colocar essas mesmas informações na classe Pedido seria redundante. Além disso, deixar os dados centralizados facilita muito a manutenção, porque qualquer alteração na estrutura do cliente, como adicionar um número de telefone, só precisaria ser feita na classe Cliente, sem precisar mexer em Pedido. Isso também reflete melhor a lógica do mundo real, afinal, um pedido pertence a um cliente, então faz sentido que Pedido tenha um objeto Cliente em vez de um monte de campos soltos. E claro, seguindo a filosofia da programação orientada a objetos (POO), esse modelo deixa o código mais bem organizado, coeso e fácil de entender. |

|  |
| --- |
| **SUGESTÃO #4: Espaços em branco e recuo** |
| **Arquivo:** Supermercado.java  **Código atual:**    **Sugestão de melhoria:**    **Justificativa**: O código agora está mais organizado e fácil de ler porque os elementos dentro da classe Supermercado e do método main foram corretamente indentados, deixando a estrutura mais clara. Também adicionei comentários explicativos para que cada parte do código tenha seu propósito bem definido, o que facilita o entendimento. Além disso, usei espaços em branco para separar melhor cada operação, como a adição de produtos e o fechamento do pedido, evitando um amontoado de informações difíceis de acompanhar. O recuo segue um padrão uniforme, sem desalinhamentos que poderiam atrapalhar a leitura ou até mesmo a depuração. No geral, essas práticas tornam o código mais limpo, organizado e fácil de manter, além de melhorar a colaboração entre desenvolvedores. |

|  |
| --- |
| **SUGESTÃO #5: Comentários do código** |
| **Arquivo:** Supermercado.java  **Código atual:**    **Sugestão de melhoria:**    **Justificativa**: As melhorias aplicadas foram importantes porque removemos comentários redundantes que só repetiam o que o código já mostrava, evitando poluir e deixar confuso. Também passamos a explicar melhor o "porquê" de cada ação, para ajudar quem for ler a entender a intenção do código e não só o que ele faz. Além disso, adicionamos o JavaDoc no método main, que serve para documentar de forma mais formal qual é o propósito e o funcionamento do programa. Por fim, deixamos os comentários mais informativos, explicando o fluxo do programa e destacando decisões importantes que justificam cada parte do código. |

|  |
| --- |
| **SUGESTÃO #6: Nomeação adequada** |
| **Arquivo:** Supermercado.java  **Código atual:**    **Sugestão de melhoria:**    **Justificativa**: As melhorias que eu fiz na nomeação das variáveis foram importantes porque deixam o código mais fácil de entender, tanto para mim quanto para outras pessoas que forem ler depois. Troquei os nomes curtos e genéricos como "c" e "p" por nomes mais descritivos como "carrinhoDeCompras" e "pedido", que mostram claramente o que cada variável representa. Isso segue a ideia do código limpo, onde a gente deve usar nomes que revelem a intenção do código e evitem abreviações confusas. Com isso, o código fica mais organizado, não precisa de comentários explicando o óbvio, e fica mais fácil de manter ou modificar no futuro, porque já dá pra saber o que cada parte está fazendo só de bater o olho. |

**SUGESTÕES DO DESENVOLVEDOR**

Abaixo, encontram-se os problemas identificados e as sugestões de melhorias propostas pelo responsável.

|  |
| --- |
| **💡** Esse é um espaço reservadopara sugestões extras. Se esse espaço estiver em branco, nenhuma possível melhoria foi identificada. |

|  |
| --- |
| **SUGESTÃO #7:** |
| **Arquivo:**  **Código atual:**    **Sugestão de melhoria:**    **Justificativa**: As melhorias nesse novo código foram muito importantes porque agora o cliente é um objeto separado, o que evita repetir informações e deixa o código mais organizado. Também foi criado um construtor para o pedido, que já recebe todos os dados necessários, evitando a necessidade de vários setters. Além disso, a parte de mostrar as informações do pedido foi passada para a classe PedidoView, seguindo o princípio de separar as responsabilidades, como o PDF recomenda. Os nomes das variáveis também ficaram melhores, mais claros, e mostram exatamente o que cada coisa faz, o que deixa o código mais fácil de entender e de manter no futuro. Por fim, o código ficou mais limpo e com o fluxo bem organizado, o que facilita a leitura e evita confusões. |