Relatório do Projeto: Sistema de Loja Online

1. Quais são as classes que criou para o projeto? Explique a função de cada classe e o motivo de ter optado por essa estrutura.

Para o desenvolvimento deste projeto, foram criadas cinco classes principais: Product, ShoppingCart, Order, User e Store.

- **Product**: Esta classe representa os produtos disponíveis na loja, contendo atributos como id, name, price, stock e description. Escolhi esta classe para encapsular as informações básicas de cada produto, permitindo também atualizações no stock e exibição de detalhes. Isso torna a gestão dos produtos centralizada e fácil de manipular.
- **ShoppingCart**: Representa o carrinho de compras onde o usuário adiciona itens antes de concluir o pedido. Inclui um dicionário items para armazenar produtos e suas quantidades e métodos para adicionar, remover e listar itens. A criação desta classe permite que o processo de seleção de itens seja separado da criação de um pedido final, facilitando a organização e a flexibilidade do código.
- **Order**: A classe Order é responsável por representar um pedido realizado. Armazena o usuário que fez a compra, os itens adquiridos e o total do pedido, além do status, que indica se o pedido foi processado. Esta classe permite que cada pedido seja uma entidade distinta, simplificando a gestão e o histórico de pedidos.
- **User**: Esta classe gere as informações dos clientes, como name, email, address e order_history. Ela armazena o histórico de pedidos de cada cliente, possibilitando um registo individualizado e de fácil acesso. A separação de um utilizador como uma entidade independente facilita a adição de funcionalidades futuras, como contas de clientes ou perfis personalizados.
- **Store**: A classe Store atua como um ponto central para as operações da loja, armazenando a lista de produtos e usuários e processando pedidos. A inclusão dessa classe permite uma gestão centralizada de produtos e usuários, além de facilitar o controle das operações principais da loja, como o processamento de pedidos.

Esta estrutura foi escolhida para manter o código organizado e modular, com cada classe desempenhando um papel específico no sistema. Isto permite a fácil manutenção e futuras expansões do sistema.

2. Proponha uma resposta alternativa à anterior, explicando porque preferiu uma em relação a outra.

Uma alternativa seria combinar as classes ShoppingCart e Order em uma única classe Transaction. Essa abordagem poderia simplificar o código e reduzir o número de classes, centralizando o controle do processo de compra e de pedidos em uma só entidade. No entanto, optei pela estrutura original porque a separação entre ShoppingCart e Order permite flexibilidade. Com elas separadas, o usuário pode modificar o carrinho sem afetar o pedido e vice-versa, o que facilita a gestão do processo de compra e atende melhor a

um fluxo de loja online. A estrutura atual também possibilita que pedidos e histórico sejam rastreados separadamente, o que é benéfico para o usuário.

3. Como decidiu a distribuição de responsabilidades entre as classes?

A distribuição das responsabilidades entre as classes foi baseada no princípio de responsabilidade única, onde cada classe é responsável por um único aspeto do sistema:

- A classe Product gere as informações básicas de um produto.
- ShoppingCart gere a seleção de produtos para compra, permitindo adicionar e remover itens antes de finalizar um pedido.
- Order cuida do processamento do pedido e do seu status.
- User mantém as informações do cliente e o seu histórico de pedidos.
- Store atua como o núcleo do sistema, gerindo produtos e utilizadores e coordenando o processo de pedidos.

Essa distribuição permite que cada classe seja testada e modificada independentemente, facilitando o desenvolvimento e a manutenção do sistema.

4. O seu projeto pode evoluir para versões futuras com novas funcionalidades ou uma interface melhorada. Quais modificações seriam necessárias no design inicial para incluir essas variantes no futuro?

Este projeto tem um design modular, o que facilita a expansão. As seguintes modificações poderiam ser adicionadas no futuro:

- **Integração de métodos de pagamento**: A inclusão de uma classe Payment permitiria lidar com diferentes métodos de pagamento (cartão de crédito, PayPal, etc.), tornando o sistema mais flexível.
- **Interface Gráfica de Usuário (GUI)**: A adição de uma interface gráfica poderia envolver a criação de uma camada de apresentação independente, utilizando bibliotecas como o Tkinter. As classes existentes seriam usadas como base para manipulação de dados, facilitando a criação de uma interface amigável sem a necessidade de modificar as funcionalidades principais.
- **Sistema de Avaliações**: Uma classe Review poderia ser criada para permitir que os usuários avaliem produtos. Isso exigiria adicionar um relacionamento entre User e Product, permitindo o registo e a exibição de avaliações.
- **Catálogo de Produtos**: A criação de categorias e filtros para os produtos exigiria modificar a classe Product para incluir uma categoria, além de ajustar a Store para possibilitar filtros avançados.

Com essas mudanças, o sistema poderia ser expandido para suportar novas funcionalidades, mantendo a compatibilidade com o design inicial.

5. Como poderia estruturar o código de forma que seja fácil adicionar novos elementos sem causar grandes modificações no código existente?

Para facilitar a adição de novos elementos sem grandes mudanças no código, o projeto segue princípios de orientação a objetos e modularidade. Algumas práticas específicas são:

- **Uso de Interfaces e Abstrações**: Se for necessário implementar métodos de pagamento, por exemplo, uma interface abstrata PaymentMethod poderia ser definida, com classes concretas para diferentes métodos (cartão, PayPal). Assim, novos métodos de pagamento poderiam ser adicionados sem modificar a estrutura existente.
- **Princípio de Aberto/Fechado (Open/Closed Principle)**: O projeto foi estruturado para que novas funcionalidades possam ser adicionadas estendendo as classes atuais ou criando novas classes. Por exemplo, a classe Product pode ser estendida para criar subclasses de produtos específicos (como eletrónicos ou roupas) sem modificar a lógica do sistema.
- **Injeção de Dependências**: As dependências entre classes são minimizadas, permitindo que objetos sejam facilmente substituídos ou estendidos. A Store pode, por exemplo, ser configurada para aceitar novos tipos de User ou Product sem modificações substanciais.
- **Separação de Camadas**: A lógica de negócios (como Store, Product, Order) está separada da apresentação, permitindo que o sistema seja facilmente integrado com uma interface gráfica ou API sem alterar as classes fundamentais.

Essas práticas garantem que o sistema seja flexível e possa ser adaptado para atender a novas necessidades com o mínimo de retrabalho, permitindo a introdução de novas funcionalidades sem comprometer o design existente.