Efolio Global PROGRAMAÇÃO POR OBJETOS

NOME: Francisco Silveiro Cardoso

N.º DE ESTUDANTE: 2303219

1.

a) No meu projeto, o polimorfismo foi utilizado principalmente na interação com os produtos e pedidos, onde diferentes tipos de objetos podem ser tratados de forma comum, mas com comportamentos específicos para cada tipo de objeto.

No caso da classe Product e da classe DiscountedProduct (que herda de Product), temos um exemplo de polimorfismo na maneira como os objetos dessas classes respondem ao método display_info().

A classe Product tem o método display_info() que exibe as informações do produto (nome, preço, estoque).

A classe DiscountedProduct, que herda de Product, também tem o método display_info(), mas ele é sobrescrito para calcular e exibir o preço com desconto. Ou seja, embora o nome do método seja o mesmo em ambas as classes, o comportamento é diferente. Quando um objeto do tipo Product ou DiscountedProduct chama o método display_info(), o comportamento é polimórfico, pois o método da DiscountedProduct apresenta o preço com desconto, enquanto o método da Product mostra o preço original.

Outros exemplos de polimorfismo ocorrem quando diferentes classes de objetos (como ShoppingCart, Order, User) interagem com o mesmo método, mas com comportamentos diferentes.

b)

A principal vantagem do polimorfismo no meu projeto é que ele permitiu que tratássemos todos os produtos de maneira semelhante, independentemente de serem produtos normais ou produtos com desconto. Ambas as classes (Product e DiscountedProduct) implementam o método display_info(), mas de forma distinta:

Para a classe Product, o método display_info() exibe apenas o preço original do produto.

Para a classe DiscountedProduct, o método display_info() calcula e exibe o preço com desconto.

A grande vantagem do polimorfismo aqui é que basta chamar o método display_info() no objeto, e o comportamento correto será executado de acordo com a classe do objeto, sem que tenhamos de verificar manualmente se o produto tem ou não um desconto.

c)

Adicionei um novo exemplo de polimorfismo com o método calculate_total() com as classes Product e DiscountedProduct, onde o método polimórfico é usado para calcular o

preço total, levando em consideração o desconto em DiscountedProduct, mas não em Product.

d)

A utilidade desse exemplo pode ser compreendida ao considerar o impacto de mudanças e novos requisitos no sistema.

Num cenário onde múltiplos tipos de produtos podem existir no sistema (como Product, DiscountedProduct, e possivelmente mais tipos no futuro), o polimorfismo ajuda a evitar a duplicação de código. Ao ter uma função comum para calcular o total (calculate_total), não vou precisar de escrever diferentes funções para cada tipo de produto. Caso surjam outros tipos de produtos no futuro, posso apenas escrever esse método em cada nova classe derivada, sem ter de reescrever ou duplicar a lógica de cálculo em várias partes do código.

Também com o polimorfismo, a lógica que interage com os produtos pode ser simplificada. Por exemplo, o código que calcula o total de um pedido não precisa saber se o produto tem desconto ou não. Ele simplesmente chama calculate_total() no produto, e o comportamento correto será invocado dependendo do tipo do objeto, seja Product, DiscountedProduct ou qualquer outro tipo que possa ser criado no futuro.

2.

a)

A biblioteca que escolhi para o tópico 7 foi a logging do Python. É uma ferramenta para registar eventos e mensagens durante a execução de um programa. Ela é muito útil para monitorizar sistemas e também para registar atividades que ocorrem durante o funcionamento de uma aplicação, como a execução de tarefas, erros, advertências e outras informações de diagnóstico.

A biblioteca logging é utilizada através de uma classe chamada LoggingMixin e um sistema logger em logger.py.

Em logger.py estamos a configurar o logger que vai ser usado durante o programa.

A LoggingMixin é uma classe que possui um método log que escreve mensagens no log. Esse método utiliza a funcionalidade da biblioteca logging para guardar as mensagens num um arquivo ou exibi-las na consola, dependendo da configuração.

As classes como User e Order herdam a classe LoggingMixin. Assim, sempre que um evento importante ocorre (por exemplo criação de um user ou adição de um pedido), o método log é chamado para registar as ações.

Os logs gerados pelo sistema são armazenados em um arquivo chamado app.log

Cada mensagem registada inclui a data e hora, o nível de severidade, e a mensagem em si. Isso torna fácil ver o que aconteceu no sistema em determinado momento e identificar potenciais problemas.

Tendo em conta que muitas classes e métodos (quase todo o projeto) utilizam esta biblioteca apenas coloquei # Resposta 2a no logger.py e na classe LoggingMixin.

b)

A maior mudança estrutural foi sem dúvida as classes user, shopping_cart, product e order agora todas herdam logging_mixin e, por isso, agora todos os métodos destas classes estão a escrever logs para todos os métodos que são utilizados permitindo não só ter uma bem melhor monitorização sobre o programa, mas também torna muito mais fácil a deteção de bugs e erros pois estes também ficam guardados no log.