Projeto: Sistema FlexEntregas - Uso de Polimorfismo em PHP

1. Como o polimorfismo foi aplicado na sua solução?

O polimorfismo foi aplicado por meio do uso de uma interface comum (MeioDeEntregaInterface) que define os comportamentos obrigatórios para qualquer meio de entrega: calcularTempoEntrega, calcularCustoEntrega, podeOperar e getNome. Todas as classes concretas (Moto, Bicicleta, Drone, Cavalo) implementam essa interface, por meio da herança de uma classe abstrata chamada MeioDeEntregaBase, que fornece a implementação padrão para esses comportamentos.

Na prática, isso permite que objetos de diferentes tipos sejam tratados de forma uniforme no simulador. O método simular() da classe SimuladorDeEntrega recebe qualquer objeto que implemente a interface, e chama os métodos adequados, independentemente do tipo específico do objeto. Isso é um exemplo claro de polimorfismo em tempo de execução.

2. Foi por herança? Interface? Função como parâmetro? Outro?

O polimorfismo foi implementado utilizando interface (contrato de métodos) e herança de classe base para evitar duplicar código entre os meios de entrega. A interface define os métodos que todas as classes devem implementar, enquanto a classe abstrata MeioDeEntregaBase já implementa esses comportamentos de forma genérica, permitindo que as subclasses apenas definam os parâmetros específicos (nome, velocidade, custo por km e restrições climáticas).

3. A linguagem usada facilitou ou dificultou o uso de estratégias polimórficas?

O PHP facilitou bastante o uso de polimorfismo. Ele é uma linguagem orientada a objetos que permite:
- Criação de interfaces
- Herança de classes abstratas
- Tipagem de parâmetros por interface
- Tratamento uniforme de objetos com interface comum

Esses recursos tornam a implementação de padrões como Estratégia e Template Method simples e

diretos. Além disso, a linguagem é flexível o suficiente para adaptar comportamentos dinamicamente, o que é ótimo para simulações como no projeto FlexEntregas.

4. Como sua abordagem permite adicionar novos meios de entrega no futuro?

A abordagem proposta torna a adição de novos meios de entrega extremamente fácil e segura:
- Basta criar uma nova classe que herde de MeioDeEntregaBase e fornecer os parâmetros necessários (ex: nome, velocidade, custo, restrições climáticas).
- Não é necessário modificar nenhuma parte do simulador, pois ele já opera com base na interface comum.
- Nenhuma lógica duplicada precisa ser reescrita.
- O novo meio de entrega pode ser adicionado ao array de simulação, e já será processado automaticamente junto com os demais.

Exemplo:

```
class CarroEletrico extends MeioDeEntregaBase {
    public function __construct() {
        parent::__construct("Carro Elétrico", 80, 0.3, ["neve"]);
    }
}
```

E depois:

```
$meios[] = new CarroEletrico();
```

Com isso, garantimos extensibilidade, baixo acoplamento e alta coesão, atendendo aos princípios SOLID de boa arquitetura orientada a objetos.