

O que é programação orientada a objeto?

7 minutos

A OOP (programação orientada a objeto) é um paradigma de programação. Ela se baseia na ideia de *agrupar* dados e funções relacionados em "ilhas" de informações. Essas ilhas são conhecidas como *objetos*.

Não importa qual paradigma é usado, os programas usam a mesma série de etapas para resolver problemas:

1. **Entrada de dados:** os dados são lidos de algum lugar, que pode ser o armazenamento de dados como um sistema de arquivos ou um banco de dados.
2. **Processamento:** os dados são interpretados e possivelmente alterados para serem preparados para exibição.
3. **Saída de dados:** os dados são apresentados para que um usuário físico ou um sistema possa lê-los e interagir com eles.

OOP versus programação procedural

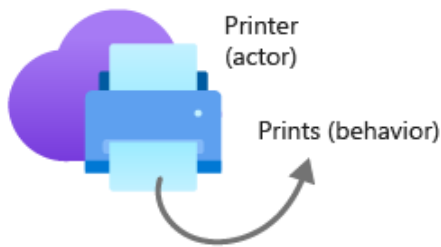
Vamos tentar definir o que é OOP comparando-a com outro paradigma: programação procedural. A programação de procedimento se dispõe a resolver um determinado problema chamando procedimentos, também conhecidos como funções ou métodos. Funções e variáveis são construídas para abordar as várias fases descritas nas etapas anteriores.

O paradigma OOP não é diferente nesse aspecto. O que o destaca é a maneira como examina o mundo. Em comparação com a programação procedural, a OOP dá um passo para trás e examina todo o panorama. Em vez de trabalhar nos dados e levá-los de uma fase para a seguinte, a OOP tenta entender o mundo em que os dados são operados. Ela faz isso *modelando* o que vê.

Modelagem de OOP: identificar conceitos

Durante a fase de modelagem, você examina uma descrição de um domínio e tenta analisar o texto sobre o que ocorre. A primeira etapa é identificar os atores. Eles são chamados de atores porque *atuam* e executam uma ação. Por exemplo, uma impressora (ator) imprime (ação).

Após identificar atores, você examina *o que* eles fazem, que é o comportamento. Em seguida, você examina as descrições dos atores e de todos os dados necessários para executar a ação. Os atores são transformados em objetos, as características são codificadas como dados nos objetos e os comportamentos são funções que também são adicionadas ao objeto.



A ideia é que os dados nos objetos podem ser alterados chamando funções nos próprios objetos. Também há a noção de que os objetos *interagem* uns com os outros para chegar a um resultado tangível.

Benefícios da OOP

Então, por que usar a OOP? Por que não usar algum outro paradigma? Para ficar claro, a OOP não é melhor ou pior do que outros paradigmas. Há prós e contras em tudo. A OOP traz alguns benefícios interessantes, por exemplo:

- **Encapsulamento de dados:** o encapsulamento de dados trata-se de ocultar dados do restante do sistema e permitir o acesso apenas a partes dele. O motivo é que os dados armazenam *estado* e esse estado pode ser composto por uma ou mais variáveis. Se essas variáveis precisarem ser alteradas ao mesmo tempo, você precisará protegê-las e permitir o acesso apenas por meio de métodos públicos para que as alterações sejam feitas de maneira previsível. A OOP tem mecanismos como níveis de acesso, em que os dados que estão em um objeto só podem ser acessados pelo próprio objeto ou podem ser tornados públicos.
- **Simplicidade:** a criação de sistemas grandes é uma tarefa complexa, com muitos problemas para serem resolvidos. A possibilidade de dividir a complexidade em problemas menores, em objetos, significa poder *simplificar* a tarefa geral.
- **Facilidade de modificar:** quando você depende de objetos e modela seu sistema com eles, é mais fácil rastrear quais partes do sistema precisam de modificação. Por exemplo, talvez seja necessário corrigir um bug ou adicionar um novo recurso.
- **Facilidade de manutenção:** manter o código, em geral, é difícil e fica mais difícil ao longo do tempo. Ela requer disciplina na forma de uma boa nomenclatura e de uma arquitetura clara e consistente, entre outras considerações. O uso de objetos facilita a localização de uma área específica do seu código que precisa de manutenção.
- **Capacidade de reutilização:** você pode usar a definição de um objeto muitas vezes em muitas partes de seu sistema ou, potencialmente, em outros sistemas também. Ao reutilizar o código, você economiza tempo e dinheiro, pois precisa escrever menos código e atinge o seu objetivo mais rapidamente.

Modelar um sistema OOP

Geralmente, o software é escrito para atender a uma necessidade de tornar algo mais rápido, mais eficiente e menos propenso a erros. As pessoas simplesmente não conseguem competir com software quando se trata de acelerar um processo em determinados casos. Usar a OOP é tanto exercer a modelagem quanto escrever o código para implementar a lógica dele. Modelagem trata-se de aprender a

identificar os atores, os dados necessários e o tipo de interação que está ocorrendo. Você pode modelar um sistema só lendo uma descrição dele.

Estudo de caso de um sistema de gerenciamento de faturas

Vamos examinar um fluxo manual muito difícil para várias empresas, a saber, o *gerenciamento de faturas*. Muitas empresas recebem faturas que precisam ser pagas no prazo. Atrasar pagamentos gera multas, que se traduzem em dinheiro perdido. Antes que uma fatura possa ser paga, ela precisa ser *processada*. É comum que uma fatura passe por algumas mãos antes que seja registrada em algum lugar e o pagamento seja feito.

O processo geralmente começa com uma fase de classificação inicial em que a fatura é enviada ao departamento apropriado. Em seguida, a fatura é conferida e aprovada por alguém com o nível de autorização adequado. Por fim, ela é paga. Para uma pequena empresa, o proprietário pode realizar todas as etapas. Em uma grande empresa, muitas pessoas e processos podem estar envolvidos, o que torna o gerenciamento de faturas uma atividade complexa.



O que essa descrição tem a ver com a OOP? Se você adotasse o fluxo de trabalho anterior, que geralmente é manual, e o transformasse em um software escrito, a primeira coisa que você faria seria tentar modelar o sistema. Com o contexto do gerenciamento de faturas, você pode começar a ver atores (objetos), comportamentos e dados lendo uma descrição do domínio do problema.

Se você pensar no domínio descrito como tendo fases, entrada, processamento e saída, poderá começar a preencher coisas como na seguinte tabela:

[Expandir a tabela](#)

Fase	O que
Entrada	Fatura
Processando	Classificando, Aprovação, Rejeição
Saída	Pagamento

A tabela anterior descreve o que acontece em cada fase. Você conseguiu encontrar os dados, as coisas que acontecem com os dados durante o processamento e o resultado final, que é um pagamento. Neste

ponto, você ainda pode resolver o fluxo de trabalho do sistema de gerenciamento de faturas com qualquer paradigma que desejar. Como você o leva daqui para a OOP?

Localizar objetos, dados e comportamento


Você encontra os diferentes artefatos do seu sistema fazendo perguntas como:

- Quem interage com quem?
- Quem faz o quê com quem?

Com essas perguntas em mente, você pode criar instruções. Vamos destacar os diferentes artefatos nessas instruções para que fique claro quais partes são importantes para nosso sistema.

1. O *serviço de email***entrega** uma *fatura* para o sistema.
2. A *fatura* é **classificada** por um *código de referência* o manualmente por um *classificador* para ir para o departamento correto.
3. A *fatura* é **aprovada** ou **rejeitada** por um *aprovador* com base em fatores como, por exemplo, exatidão e tamanho do *valor*.
4. A *fatura* é **paga** por um *processador de pagamentos* usando as *informações de pagamento* fornecidas.

Agora você pode extrair objetos, dados e o comportamento das frases e organizá-los em uma tabela, desta forma:

 Expandir a tabela

Fase	Ator	Comportamento	Dados
Entrada	Serviço de email	Entrega	Fatura
Entrada	Sistema	Recebe	Fatura
Processando	Classificadores ou sistema	Classifica ou encaminha	Fatura (código de referência)
Processando	Aprovador	Aprova ou rejeita	Fatura (valor)
Saída	Processador de pagamentos	Paga	Fatura (informações do pagamento)

Muita coisa aconteceu com a descrição inicial do sistema de gerenciamento de faturas. Atores (objetos) foram encontrados. Dados importantes foram identificados e *agrupados* com objetos identificados. O comportamento também foi encontrado, o que deixa mais claro quais atores (objetos) interagem entre si.

Como resultado, você conseguiu identificar *quem* faz qual comportamento com *quem*. Você fez uma análise inicial neste ponto, que é um ótimo começo. Mas a pergunta permanece: como você transforma essa análise em código? A resposta a essa pergunta é o que vamos resolver ao longo deste módulo.

ⓘ Observação

Provavelmente um sistema de gerenciamento de faturas real é muito mais complexo e poderia depender de muito mais dados e lógica. A possibilidade de modelar um sistema dessa forma significa que você tem uma abordagem estruturada para pensar no problema.

Unidade seguinte: Usar classes e variáveis para transferir o modelo OOP para o código

[Continuar >](#)