

# Introdução ao Projeto Orientado a Objetos

---

# Programação Orientada a Objetos


- A Programação Orientada a Objetos (POO) é o paradigma dominante hoje em muitas áreas do desenvolvimento de *software*.
- A popularidade da POO se dá por sua capacidade de permitir aos desenvolvedores de *software* gerenciar a complexidade do projeto de *software*.
- Usa-se "gerenciar" ao invés de "diminuir" ou "mitigar" porque entendemos que a complexidade é algo inevitável, porém cabe ao desenvolvedor ter (ou tentar ter) o controle de como a complexidade atuará no projeto de *software*.

# Princípios do Projeto Orientado a Objetos

Morelli e Walde<sup>1</sup> colocam como princípios do Projeto Orientado a Objetos:

- dividir e conquistar;
- encapsulamento;
- *interface*;
- ocultação de informação;
- generalidade;
- extensibilidade;
- abstração.

---

<sup>1</sup>Java, Java, Java: Object-Oriented Problem Solving. 3<sup>ed</sup>. 

# Princípio de dividir e conquistar

- Os problemas a serem solucionados com uso de *software* tendem não apenas a serem complexos, mas a serem grandes.
- Segundo Morelli e Walde, o primeiro passo para projetar *software* é dividir o problema em objetos que interagirão entre si para resolver o problema.

# Princípio do encapsulamento

- Os objetos são representações de coisas materiais e ideais.
- Como representações, eles não precisam ter todas os atributos que possuem no mundo real, porém precisam dos atributos que serão necessários para realizar seu papel dentro do sistema.

# Princípio da *interface*

- Tendo os objetos e seus atributos definidos, deve-se definir como estes objetos deverão interagir com os demais objetos.
- Isto significa definir a *interface* do objeto, ou seja, que dados o objeto fornecerá aos demais e quais ele requerirá para realizar o cálculo.
- Por exemplo, um método retorna (ou não) um valor, mas para isso ele pode demandar certos dados (parâmetros).
- Do ponto de vista prático, este princípio define métodos e atributos públicos.

# Princípio da ocultação de informação

- O princípio da ocultação de informação está presente como elemento sintático em diversas linguagens de programação orientadas a objetos.
- Em outras, ele é apenas um princípio semântico.
- O conceito por trás desse princípio diz que nem todos os atributos e métodos devem estar expostos a todos os objetos.
- Morelli e Walde dão o exemplo que não ter acesso ao mecanismo de um relógio protege seu funcionamento, ao mesmo passo que não limita sua utilidade.
- Em termos de programação, este princípio define métodos e atributos privados.

# Princípio da generalidade

- O princípio da generalidade é um tanto polêmico no contexto de projeto de *software*.
- Ele trata do uso de bibliotecas e da construção de classes genéricas.
- O uso de bibliotecas de terceiros, pode tanto agregar velocidade e segurança ao desenvolvimento de *software* quanto causar o seu oposto.
- Ao usar uma biblioteca de terceiro, o desenvolvedor está confiando na mesma.
- Sobre a construção de classes genéricas, é importante ter cuidado, pois colocar o reúso em primeiro lugar pode tornar o projeto de *sotware* desnecessariamente mais complexo.
- Por outro lado, criar classes genéricas para realizar tarefas que se repetirão no *software* pode simplificar o projeto de *software*.



# Princípio da extensibilidade

- O princípio da extensibilidade está bastante ligado aos conceitos de herança.
- A herança é um conceito que permite modificar as capacidades de um objeto, porém mantendo a *interface* definida pela classe pai<sup>2</sup>.
- Tenha-se, por exemplo, um sistema de monitoramento. Um dos módulos deste sistema ficará responsável pelos cálculos e outro pela leitura dos sensores. Caso o protocolo de um ou mais sensores mude, pode-se estender a classe (ou *interface*<sup>3</sup>) responsável pela leitura para que funcione com este novo protocolo.
- Vamos lembrar que a herança não é a solução de todos os problemas e seu uso deve se dar em casos específicos.
- O uso descontrolado de herança pode gerar alto nível de acoplamento no sistema.

---

<sup>2</sup>Tradução do termo "parent class".

<sup>3</sup>Classe do tipo "interface", conceito presente no Java.

# Princípio da abstração

- O princípio da abstração é o que rege o projeto de *software* como um todo.
- Todos os objetos em um *software* nada mais são que abstrações de coisas e conceitos do mundo real.
- Por isso, os objetos não conterão todas as informações e papéis destes objetos, mas apenas os necessários para o *software* que será desenvolvido.

# Projeto Orientado a Objetos

- O projeto de *software* aqui abordado será o Projeto Orientado a Objetos.
- Se utilizado um método orientado a planos, provavelmente este projeto será bastante completo e será realizado ao fim das etapas referentes à Engenharia de Requisitos.
- Nos métodos ágeis, geralmente iterativos e incrementais, o projeto será completado a cada iteração, sendo alterado de acordo com os requisitos a serem implementados.
- Seja qual for a abordagem utilizada, é bastante interessante avaliar a possibilidade da construção de protótipos de baixa fidelidade para tornar mais claro como o *software* deve se comportar.

# Projeto Orientado a Objetos

Pode-se sistematizar a criação do Projeto Orientado a Objetos a partir das seguintes etapas:<sup>4</sup>

- decomposição do problema;
- projeto de objetos (classes);
- definição de atributos, métodos e algoritmos;

---

<sup>4</sup>Baseado no apresentado por Morelli e Walde.

# Decomposição do problema

- A decomposição do problema é a divisão do problema em problemas menores.
- A partir da análise do problema<sup>5</sup>, são observados substantivos (coisas e ideias) que fazem parte do problema e possivelmente da solução.
- Estes substantivos são candidatos a se tornarem classes.

---

<sup>5</sup>Pode-se utilizar um documento de requisitos.

# Projeto de objetos (classes)

Tendo as classes definidas, para cada uma deve-se realizar as seguintes perguntas:

- qual o papel da classe?
- quais dados ela precisará?
- quais as ações ela realizará?
- qual sua *interface*?
- quais informações serão ocultas?

# Definição de atributos, métodos e algoritmos

- O próximo passo é definir os atributos.
- Para cada atributo da classe é necessário decidir qual será o tipo de dado para representá-lo.

# Definição de atributos, métodos e algoritmos

Definidos os atributos, para cada método realiza-se as seguintes perguntas:

- qual tarefa específica o método realizará? (escopo)
- quais informações serão necessárias? (parâmetros)
- qual algoritmo utilizará?
- qual resultado produzirá? (retorno ou efeito colateral)



# Conclusão

- Os conceitos e sistematização aqui apresentados são apenas uma amostra do que é possível fazer em termos projeto de *software*.
- Apesar disto, se aplicados corretamente possibilitam o desenvolvimento ágil de aplicações.

# Referências

- Morelli, Ralph; Walde, Ralph. Java, Java, Java: Object-Oriented Problem Solving. 3ed. Hatford, EUA. 2017.