Para que serve OO?

Modelagem e Dependências

Você aprendeu Orientação a Objetos.

Entendeu classes, objetos, atributos, métodos, herança, polimorfismo, interfaces.

Aprendeu algumas soluções comuns para problemas recorrentes estudando alguns *Design Patterns*.

Mas como e quando usar OO?

Sem dúvida, a resposta tem a ver com organizar e facilitar a mudança do código no médio/longo prazo.

Mas, para Martin (2005), há duas abordagens complementares no uso de OO:

- criar um **modelo** do mundo real
- gerenciar as **dependências** do seu código

> *These principles expose the dependency management aspects of OOD as opposed to the conceptualization and modeling aspects. This is not to say that OO is a poor tool for conceptualization of the problem space, or that it is not a good venue for creating models. Certainly many people get value out of these aspects of OO. The principles, however, focus very tightly on dependency management.*
>
> (MARTIN, 2005)

OO é uma ótima ferramenta para representar, em código, os conceitos do problema que estamos resolvendo. É importante selecionar entidades de negócio e criar um modelo de domínio que as "traduza" para uma linguagem de programação.

Um bom *domain model* é o foco de metodologias e técnicas como:

- Feature-Driven Development
- Card-Responsibility-Collaboration (CRC)
- **Domain-Driven Design**

Mas OO também é uma ótima maneira de evitar código "amarrado" demais, controlando as dependências e

minimizando o acoplamento. Um código OO bem modelado, com as dependências administradas com cuidado, leva a mais flexibilidade, robustez e possibilidade de reuso.

Dependências bem gerenciadas são o foco de técnicas como:

- General Responsibility Assignment So ware Principles (GRASP)
- Design Patterns
- Dependency Injection
- **Princípios SOLID**.

Nosso foco nesse curso é aprofundar no estudo dos princípios SOLID, usando OO como uma maneira de gerenciar as dependências do nosso código.

S.O.L.I.D.

Os dez (ou onze) mandamentos de Orientação a Objetos

Robert Cecil Martin, o famoso Uncle Bob, listou os seus 10 (na verdade, 11) mandamentos da Programação Orientada a Objetos, no grupo Usenet *comp.object* (MARTIN, 1995):

> *If I had to write commandments, these would be candidates.*
>
> 1.*Software entities (classes, modules, etc) should be open for extension, but closed for modification. (The open/closed principle -- Bertrand Meyer)*
>
> 2.*Derived classes must usable through the base class interface without the need for the user to know the difference. (The Liskov Substitution Principle)*
>
> 3.*Details should depend upon abstractions. Abstractions should not depend upon details. (Principle of Dependency Inversion)*
>
> 4.*The granule of reuse is the same as the granule of release. Only components that are released through a tracking system can be effectively reused.*
>
> 5.*Classes within a released component should share common closure. That is, if one needs to be changed, they all are likely to need to be changed. What affects one, affects all.*
>
> 6.*Classes within a released component should be reused together. That is, it is impossible to separate the components from each other in order to reuse less than the total.*
>
> 7.*The dependency structure for released components must be a DAG. There can be no cycles.*
>
> 8.*Dependencies between released components must run in the direction of stability. The dependee must be more stable than the depender.*
>
> 9.*The more stable a released component is, the more it must consist of abstract classes. A completely stable component should consist of nothing but abstract classes.*
>
> 10.*Where possible, use proven patterns to solve design problems.*
>
> 11.*When crossing between two different paradigms, build an interface layer that separates the two. Don't pollute one side with the paradigm of the other.*

Os Princípios de Orientação a Objetos

Em 1996, fez uma série de artigos na revista *C++ Report* sobre o que chamou de **princípios**:

- *Open-Closed Principle* (MARTIN, 1996a)
- *Liskov Substitution Principle* (MARTIN, 1996b)
- *Dependency Inversion Principle* (MARTIN, 1996c)
- *Interface Segregation Principle* (MARTIN, 1996d)
- *Granularity* (MARTIN, 1996e)
- *Stability* (MARTIN, 1997)

O foco desses princípios é nas dependências entre objetos e componentes/módulos.

Em (Martin, 2002), Uncle Bob descreve o *Single Responsibility Principle*, reordenando os princípios e cunhando o acrônimo **S.O.L.I.D**.

Uma versão atualizada desses princípios foi lançada em C# (MARTIN, 2006).

Uncle Bob indica (MARTIN, 2009) que os princípios não são *check lists*, nem leis ou regras. São bons conselhos vindos do senso comum de gente experiente, coletados em projetos reais ao longo do tempo. Não significa que sempre funcionam ou que sempre devem ser seguidos.

Princípios de classes

Os 5 princípios S.O.L.I.D. são focados em modelagem de classes:

- ***S**ingle Responsibility Principle*: **Uma classe deve ter um, e apenas um, motivo para ser mudada**.
- ***O**pen/Closed Principle*: **Deve ser possível extender o comportamento de uma classe sem modificá-la**.
- ***L**iskov Substitution Principle*: **Classes derivadas devem ser substituíveis pelas classes base**.
- ***I**nterface Segregation Principle*: **Uma classe deve ter um, e apenas um, motivo para ser mudada**.
- ***D**ependency Inversion Principle*: **Dependa de abstrações, não de classes concretas**.

Referências

MARTIN, Robert Cecil. *The Principles of OOD*. 2005. Em: http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod.

MARTIN, Robert Cecil. *The Ten Commandments of OO Programming*. 1995. Em: https://groups.google.com/forum/?hl=en#!topic/comp.object/WICPDcXAMG8.

MARTIN, Robert Cecil. *The Open-Closed Principle*. The C++ Report, Jan. 1996a. Em: https://drive.google.com/file/d/0BwhCYaYDn8EgN2M5MTkwM2EtNWFkZC00ZTI3LWFjZTUtNTFhZGZiYmUz

MARTIN, Robert Cecil. *The Liskov Substitution Principle*. The C++ Report, Mar. 1996b. Em: https://drive.google.com/file/d/0BwhCYaYDn8EgNzAzZjA5ZmItNjU3NS00MzQ5LTkwYjMtMDJhNDU5ZTM0M
.

MARTIN, Robert Cecil. *The Dependency Inversion Principle*. The C++ Report, Jun. 1996c. Em: https://drive.google.com/file/d/0BwhCYaYDn8EgMjdlMWIzNGUtZTQ0NC00ZjQ5LTkwYzQtZjRhMDRlNTQ3Z
.

MARTIN, Robert Cecil. *The Interface Segregation Principle*. The C++ Report, Aug. 1996d. Em: https://drive.google.com/file/d/0BwhCYaYDn8EgOTViYjJhYzMtMzYxMC00MzFjLWJjMzYtOGJiMDc5N2JkYm

MARTIN, Robert Cecil. *Granularity*. The C++ Report, Nov. 1996e. Em: https://drive.google.com/file/d/0BwhCYaYDn8EgOGM2ZGFhNmYtNmE4ZS00OGY5LWFkZTYtMjE0ZGNjODC

MARTIN, Robert Cecil. *Stability*. The C++ Report, Jan. 1997. Em: https://drive.google.com/file/d/0BwhCYaYDn8EgZjI3OTU4ZTAtYmM4Mi00MWMyLTgxN2YtMzk5YTY1NTVi

MARTIN, Robert Cecil. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, 2002. 529 p. Em: https://www.amazon.com.br/Software-Development-Principles-Patterns-Practices/dp/0135974445.

MARTIN, Robert Cecil. *Agile Principles, Patterns, and Practices in C#*. Prentice Hall, 2006. 732 p. Em: https://www.amazon.com.br/Agile-Principles-Patterns-Practices-C/dp/0131857258.

MARTIN, Robert Cecil. *Getting a SOLID start.*. 2009. Em: https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start.