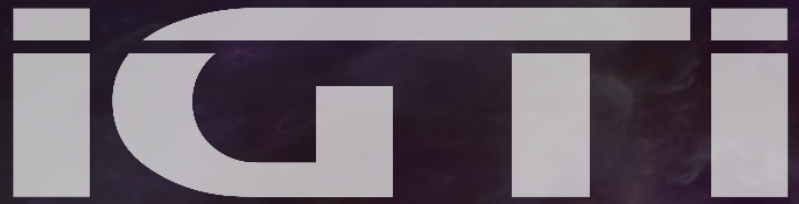


A aula interativa do **Módulo 3 - Bootcamp Arquiteto de Software** começará em breve!

Atenção:

1) Você entrará na aula com o microfone e o vídeo DESABILITADOS.
2) Apenas a nossa equipe poderá habilitar seu microfone e seu vídeo em momentos de interatividade, indicados pelo professor.
3) Utilize o recurso Q&A para dúvidas técnicas. Nossos tutores e monitores estarão prontos para te responder e as perguntas não se perderão no chat.
4) Para garantir a pontuação da aula, no momento em que o professor sinalizar, você deverá ir até o ambiente de aprendizagem e responder a enquete de presença. Não é necessário encerrar a reunião do Zoom, apenas minimize a janela.

Bootcamp Arquitetura de Software

PRIMEIRA AULA INTERATIVA

PROF. VAGNER CLMENTINO DOS SANTOS

Bootcamp Arquitetura de Software

PRIMEIRA AULA INTERATIVA

PROF. VAGNER CLEMENTINO DOS SANTOS

Nesta aula

- ❑ Revisão dos Princípios e Padrões de Projeto.
- ❑ Relevância dos Princípios SOLID.
- ❑ Padrões de Projeto ainda valem a pena?
- ❑ Espaço para dúvidas.

Apresentação

- Bacharel em Sistemas de Informação – UFMG.
- Mestre em Ciência da Computação – UFMG.
- 10 anos de experiência como desenvolvedor.
- Áreas de interesse:
  - Testes de software.
  - Padrões e Princípios de Software.
  - Arquitetura de Software.
  - Desenhos de API’s.
- https://www.linkedin.com/in/vclementino/

Acordos

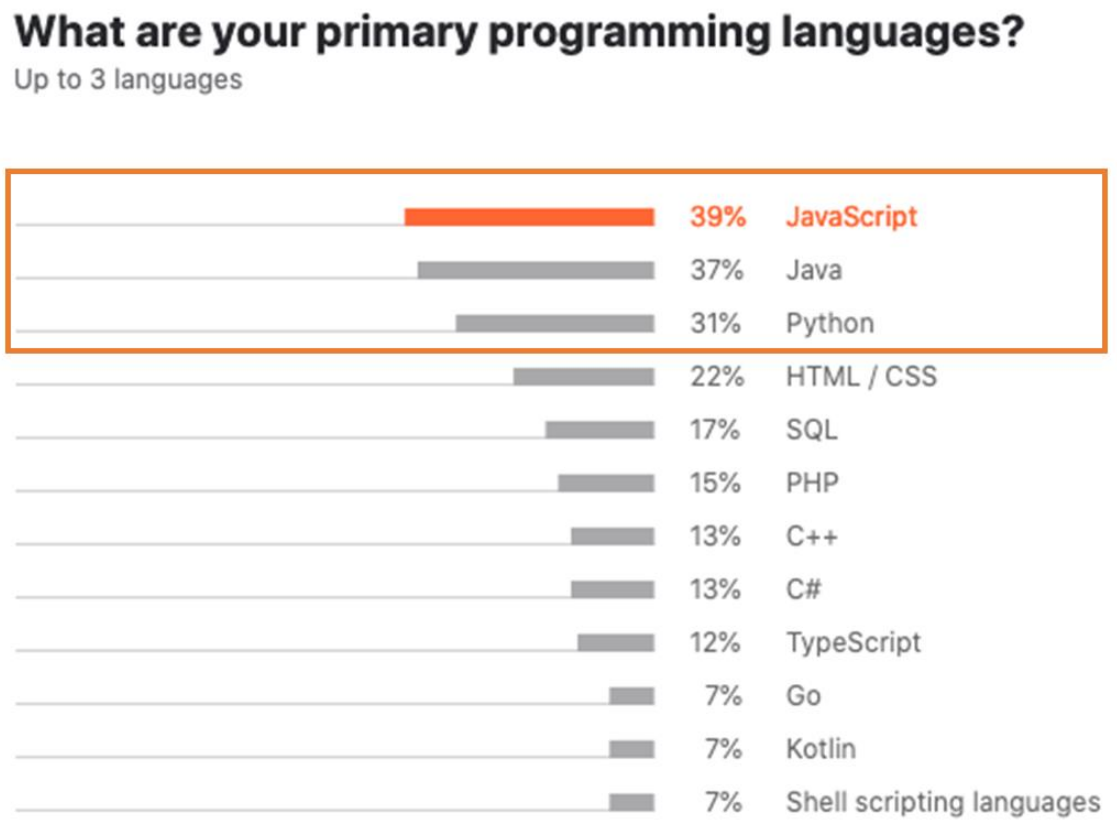
- Pratique a Paciência.
- Pratique a Empatia.
- Pratique a Curiosidade.
- Pratique o Networking.

Princípios SOLID

- Princípios SOLID é um acrônimo para:
  - **S**ingle Responsibility Principle (Responsabilidade Única).
  - **O**pen Closed/Principle (Princípio Aberto/Fechado**)**.
  - **L**iskov Substitution Principle (Princípio de Substituição de Liskov).
  - **I**nterface Segregation Principle (Princípio da Segregação de Interfaces).
  - **D**ependency Inversion Principle (Princípio de Inversão de Dependências).

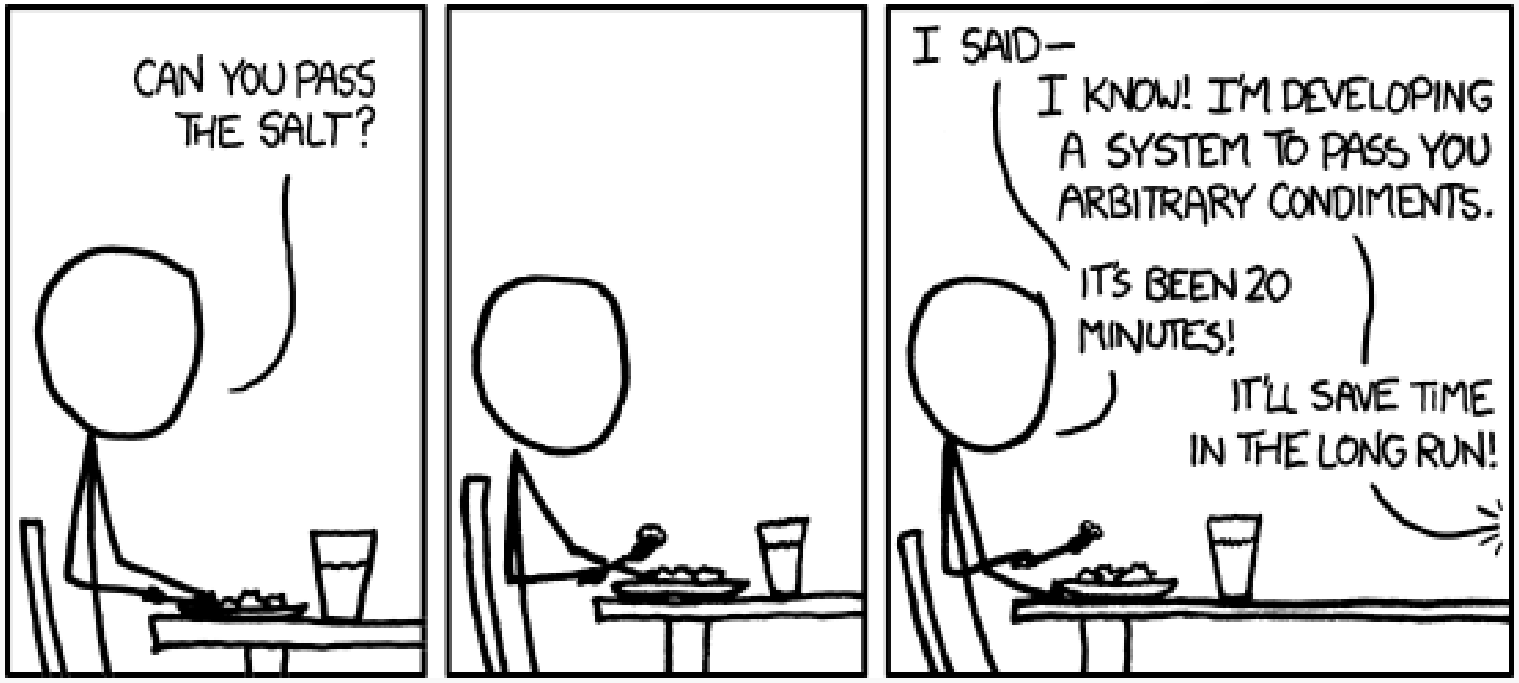
Quais as linguagens mais utilizadas em 2020?

Linguagens utilizadas em 2020



Fonte: https://www.jetbrains.com/lp/devecosystem-2020/

O uso dos padrões



Fonte: xkcd

Princípio da Segregação de Interfaces

1

Interfaces separam o comportamento que o cliente requer e como esse comportamento é implementado.

2

Interfaces têm que ser pequenas, coesas e específicas para cada tipo de cliente.

3

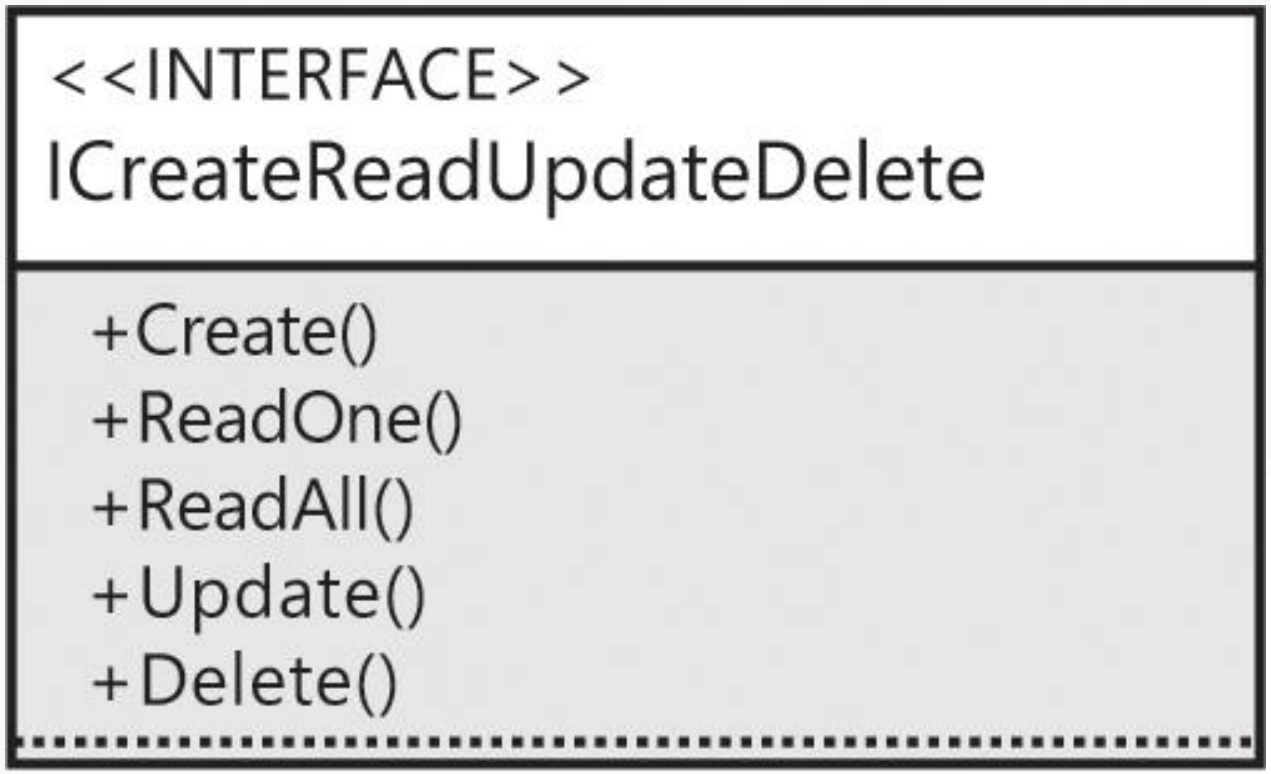
Os clientes não devem ser forçados a depender de métodos que não utilizam.

Exemplo: CRUD interface

A interface com métodos para permitir o armazenamento de uma entidade.

CRUD significa criar, ler, atualizar e excluir. As operações de leitura são divididas em dois métodos: um para a recuperação de um único registro e outro para a leitura de todos os registros.

CRUD interface

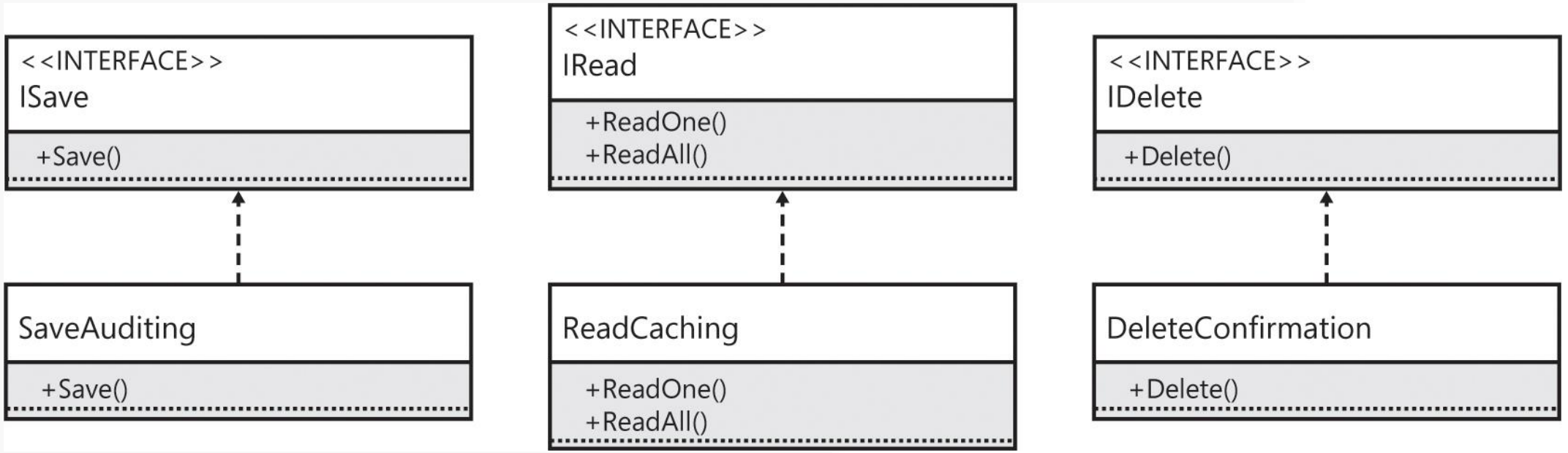


Fonte: HALL, 2017

Exemplo de Código

- https://github.com/vagnerclementino/solid/tree/main/i/java

Interface Segregada



Fonte: HALL, 2017

Exemplo Real – Golang ReadWriter

```
type ReadWriter interface {
    Reader
    Writer
}
```

Motivos para segregar interfaces

1. Ajudar “decorar” as funcionalidades.
2. Ocultar a funcionalidade dos clientes.
3. Efeito colateral de determinado padrão arquitetural (Ex.: padrão CQRS: Command/Query Responsibility Segregation).

Relevância do SOLID

- Princípio de "Aberto-Fechado”:
  - A maior parte do código que escrevemos não está contida em grandes monólitos.
  - Fazer mudanças em pequenos microsserviços é seguro e fácil.
- Princípio de Substituição Liskov:
  - Desatualizado.
  - Não usamos herança tanto quanto fazíamos há 20 anos.

FONTE: Solid Relevance, by Robert C. Martin (Uncle Bob) - https://blog.cleancoder.com/uncle-bob/2020/10/18/Solid-Relevance.html - Acessado em 20/10/2020

Responsabilidade Ùnica

- Mantemos o código em módulos separados para que as alterações em uma parte não quebrem outras partes.
- Os microsserviços não resolvem o problema se você misturar códigos que mudam por diferentes razões.

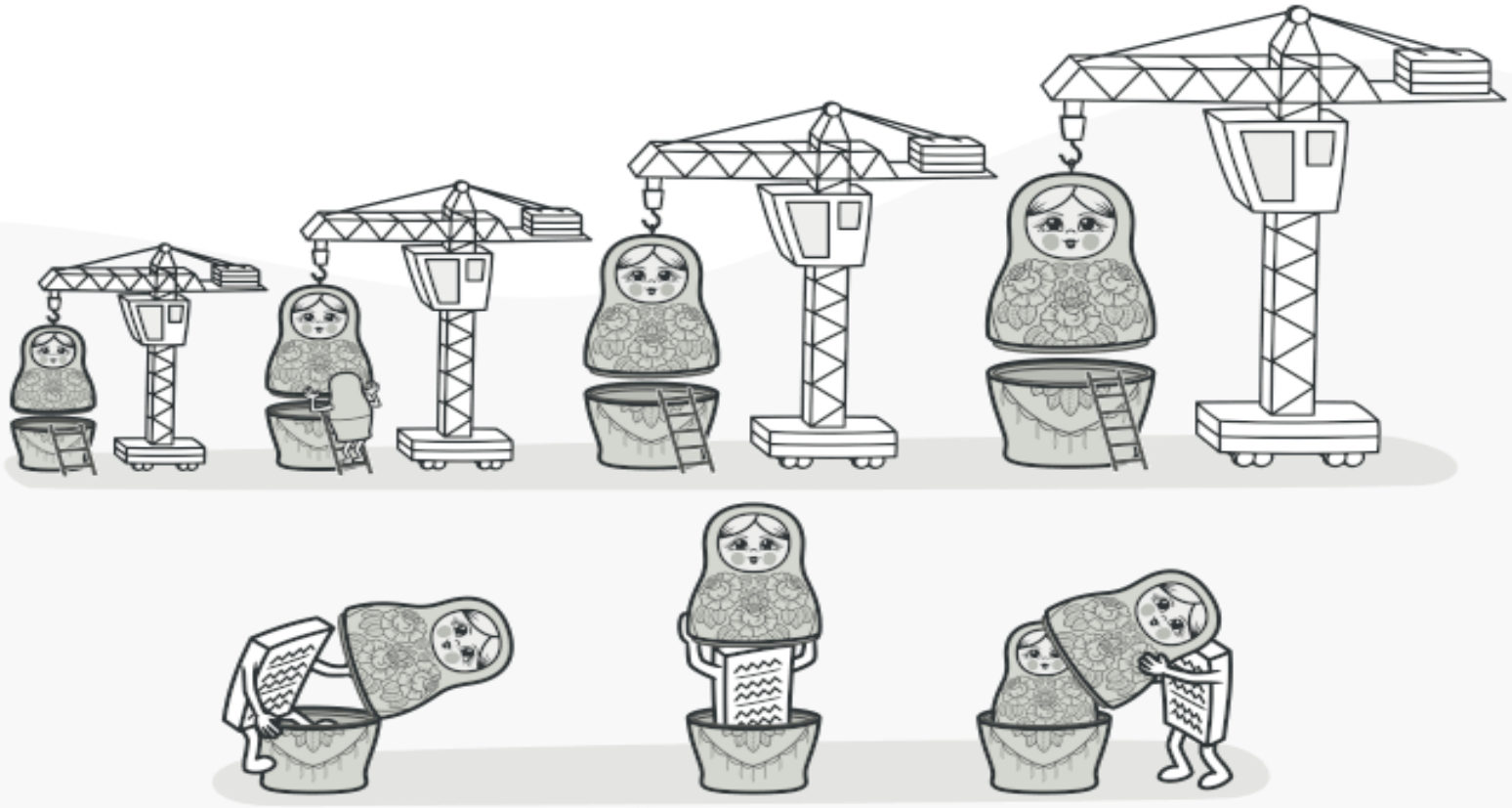
FONTE: Solid Relevance, by Robert C. Martin (Uncle Bob) - https://blog.cleancoder.com/uncle-bob/2020/10/18/Solid-Relevance.html - Acessado em 20/10/2020

Aberto/Fechado

- É claro que queremos criar módulos que possam ser ampliados sem modificá-los.
- Quando os requisitos mudam, apenas parte do código existente está errado.
- Não devemos mudar o código certo apenas para fazer o código errado funcionar novamente.

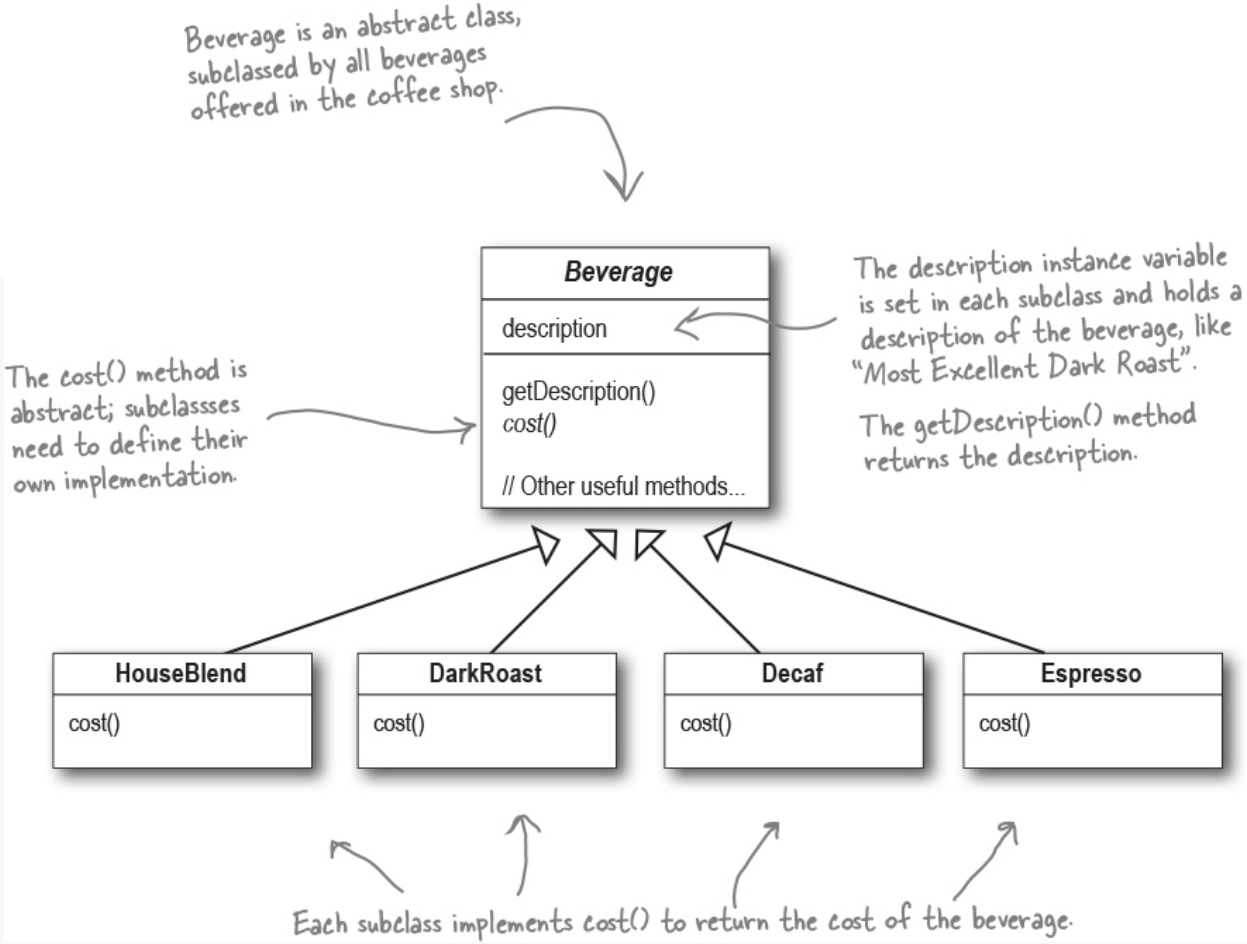
FONTE: Solid Relevance, by Robert C. Martin (Uncle Bob) - https://blog.cleancoder.com/uncle-bob/2020/10/18/Solid-Relevance.html - Acessado em 20/10/2020

Decorator



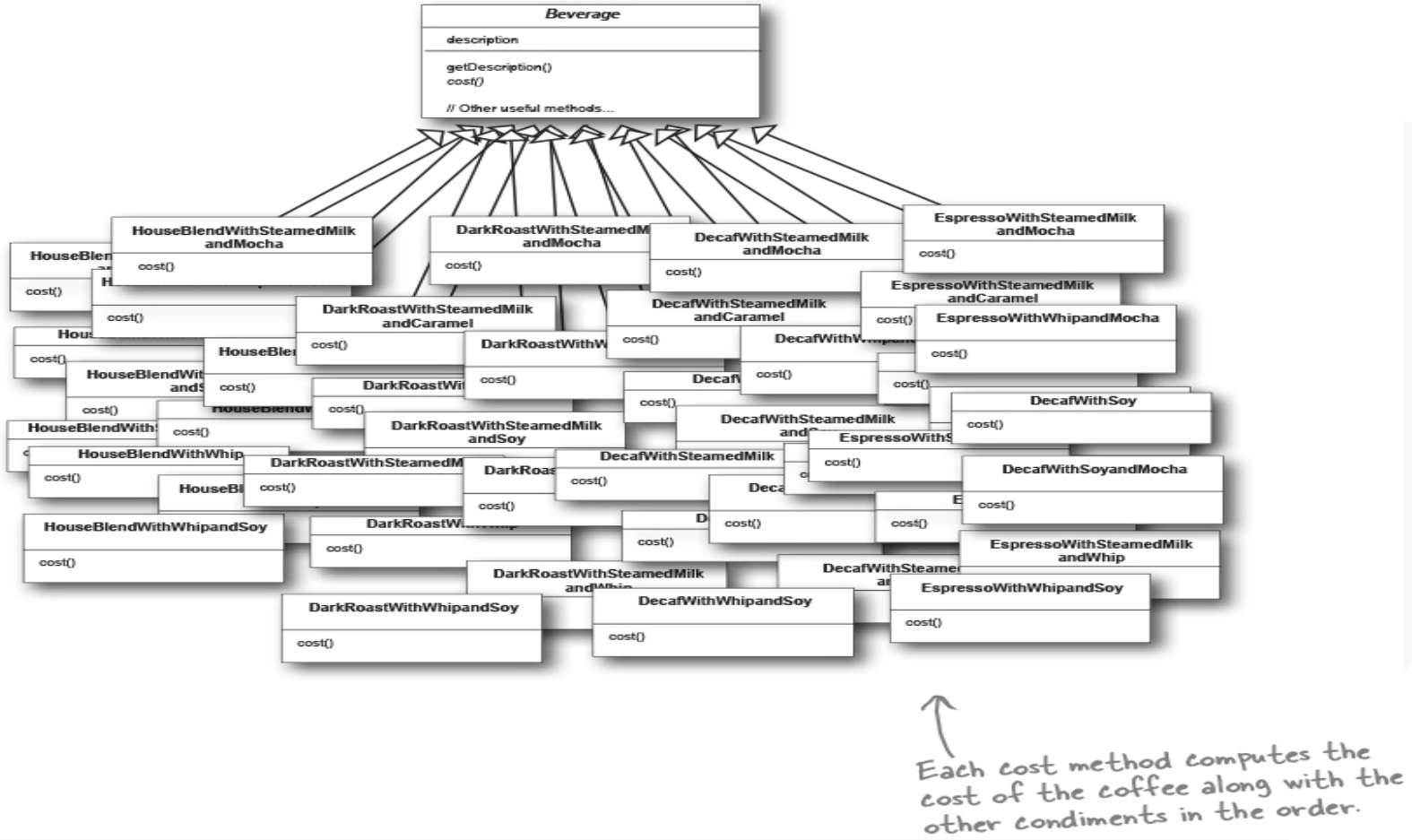
Fonte: https://refactoring.guru/

Exemplo – Cafeteria



Fonte: FREEMAN & ROBSON, 2020

Exemplo – Cafeteria



Fonte: FREEMAN & ROBSON, 2020

Herança x Composição

1

Na herança o comportamento é definido estaticamente em tempo de compilação. Na composição é possível fazer *isso dinamicamente em tempo de execução*.

2

Através da composição é possível acrescentar novas responsabilidades que não foram sequer pensadas pelo projetista da superclasse.

3

Como não mudamos o código existente, as chances de introduzir bugs ou causar efeitos colaterais são muito reduzidas.

Características do Padrão Decorator

1

Os decoradores têm o mesmo supertipo que os objetos que eles decoram.

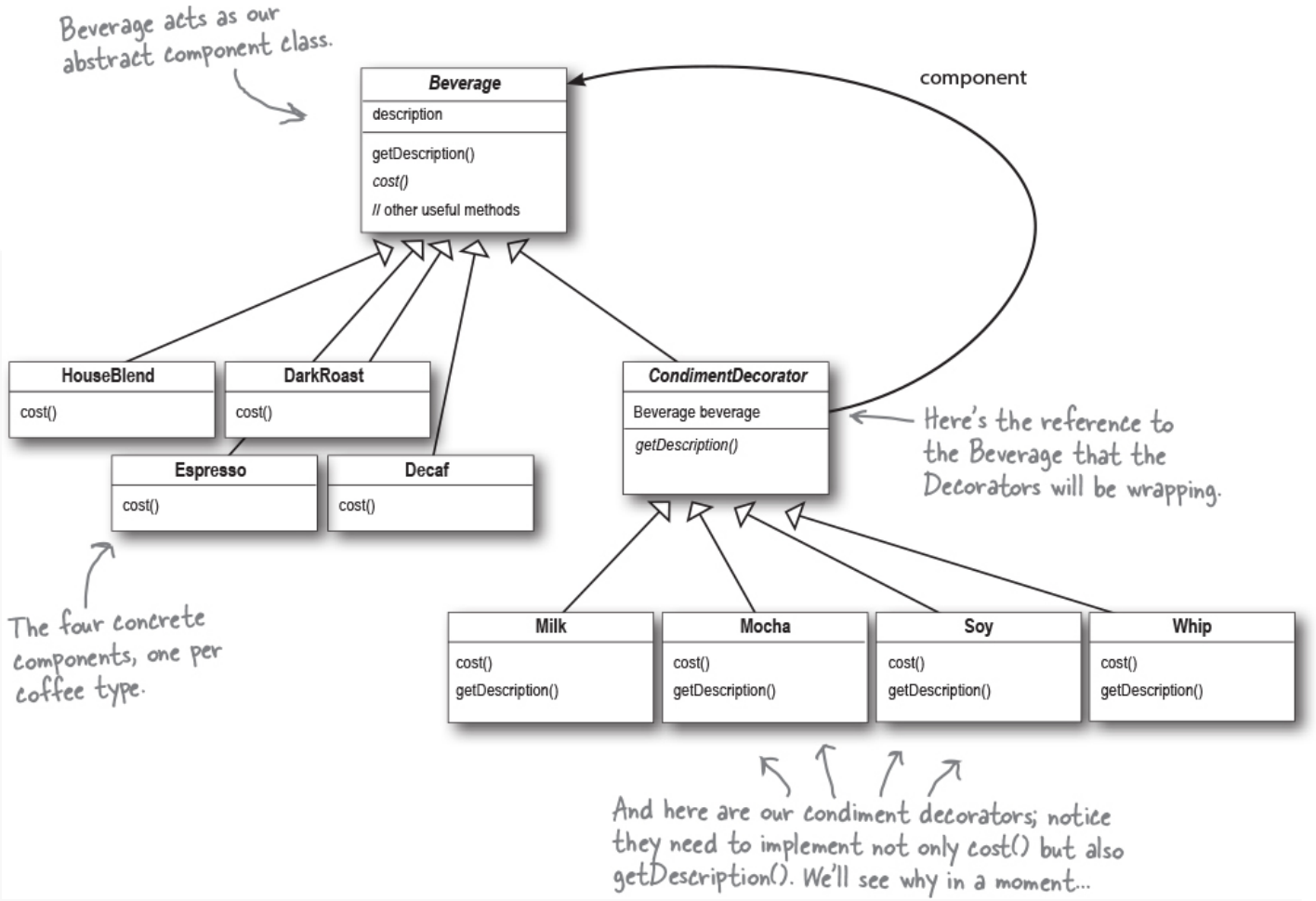
2

Você pode usar um ou mais decoradores para envolver um objeto.

3

O decorador acrescenta seu próprio comportamento ao objeto que ele decora para fazer o resto do trabalho.

Diagrama do Decorator



Fonte: FREEMAN & ROBSON, 2020

Exemplo de Código

- https://github.com/bethrobson/Head-First-Design-Patterns/tree/master/src/headfirst/designpatterns/decorator/starbuzz

Exemplo Real – java.io

- BufferedInputStream e ZipInputStream herdam de FilterInputStream.
- FilterInputStream herda de InputStream.
- InputStream funciona como um decorador abstrato.

Padrões de Projeto e Princípios Solid

1

Proposto em um cenário dominado por linguagens orientadas a objeto.

2

Mesmo o padrão mais simples tem um custo e introduz complexidade.

3

Quanto mais padrões de projeto usarmos, não há garantias que o sistema seja melhor.

Conclusão

“Olha aí meu bem

Prudência e dinheiro no bolso

Canja de galinha

Não faz mal a ninguém”

Engenho de Dentro - Jorge Ben Jor

Referências

- FREEMAN, Eric, ROBSON, Elisabeth. **Head First Design Patterns**, 2nd Edition. O'Reilly Media, Incorporated, 2020.
- HALL, Gary McLean. **Adaptive Code**: Agile coding with design patterns and SOLID principles 2nd Edition. Microsoft Press. 2017.
- Robert C. Martin (Uncle Bob). **Solid Relevance**. https://blog.cleancoder.com/uncle-bob/2020/10/18/Solid-Relevance.html - Acessado em: 11/01/2021