

Desenvolvimento de Sistemas - Fase II - 2022.2

# Design de Software

Cristian  
Elisa  
Gustavo  
Juno  
Leandro  
Wayne

---

UC: PROJETO DE SOFTWARE

IFSC - 07/11/2022  
FLORIANÓPOLIS



# O que veremos hoje

## Part 1: Understanding

---

- O que é ?
- Design vs Arquitetura

## Part 2: Jobs and Carriers

---

- Mercado de Trabalho
- Skills

## Part 3: Design Patterns

---

- POO
- GoF
- MVC

## Part 4: Design Principles

---

- DRY, YAGNI, KISS
- GRASP
- SOLID



DESIGN DE SOFTWARE

PROJETO DE SOFTWARE

Part 1:

# Understanding

O QUE É ?

DESIGN VS ARQUITETURA

# O que é Design de Software?



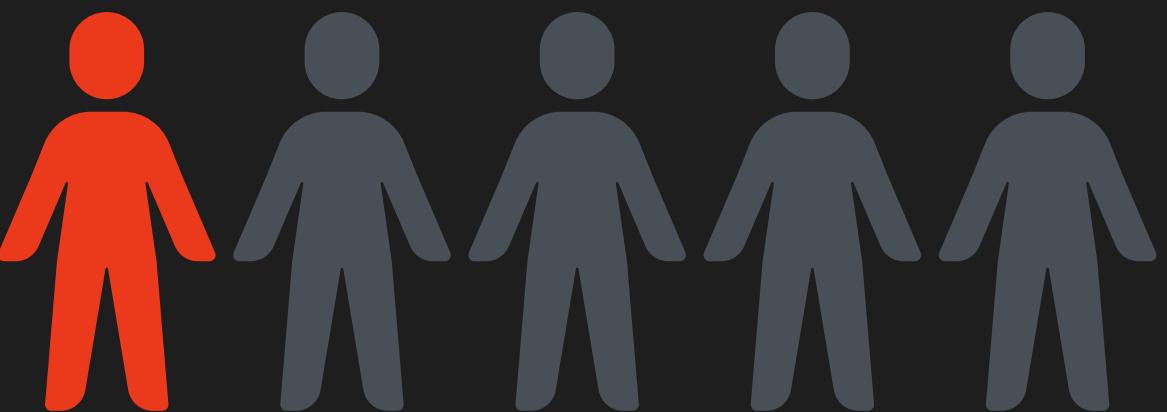
**Design de Software** (do inglês **Software design**) é a parte da **engenharia de software** que se encarrega de fazer todo o planejamento anterior ao desenvolvimento, incluindo a definição da arquitetura do software, e transformar tudo em um documento ou conjunto de documentos capazes de serem interpretados diretamente pelo programador.



# Design de software

Enquanto a arquitetura do software é responsável pelo esqueleto e pela infraestrutura de alto nível de um software, o design do software é responsável pelo design do nível de código, como o que cada módulo está fazendo, o escopo das classes e os objetivos das funções, etc.

Design  
x  
Arquitetura



Saiba a diferença !!!



DESIGN DE SOFTWARE

PROJETO DE SOFTWARE

Part 2:

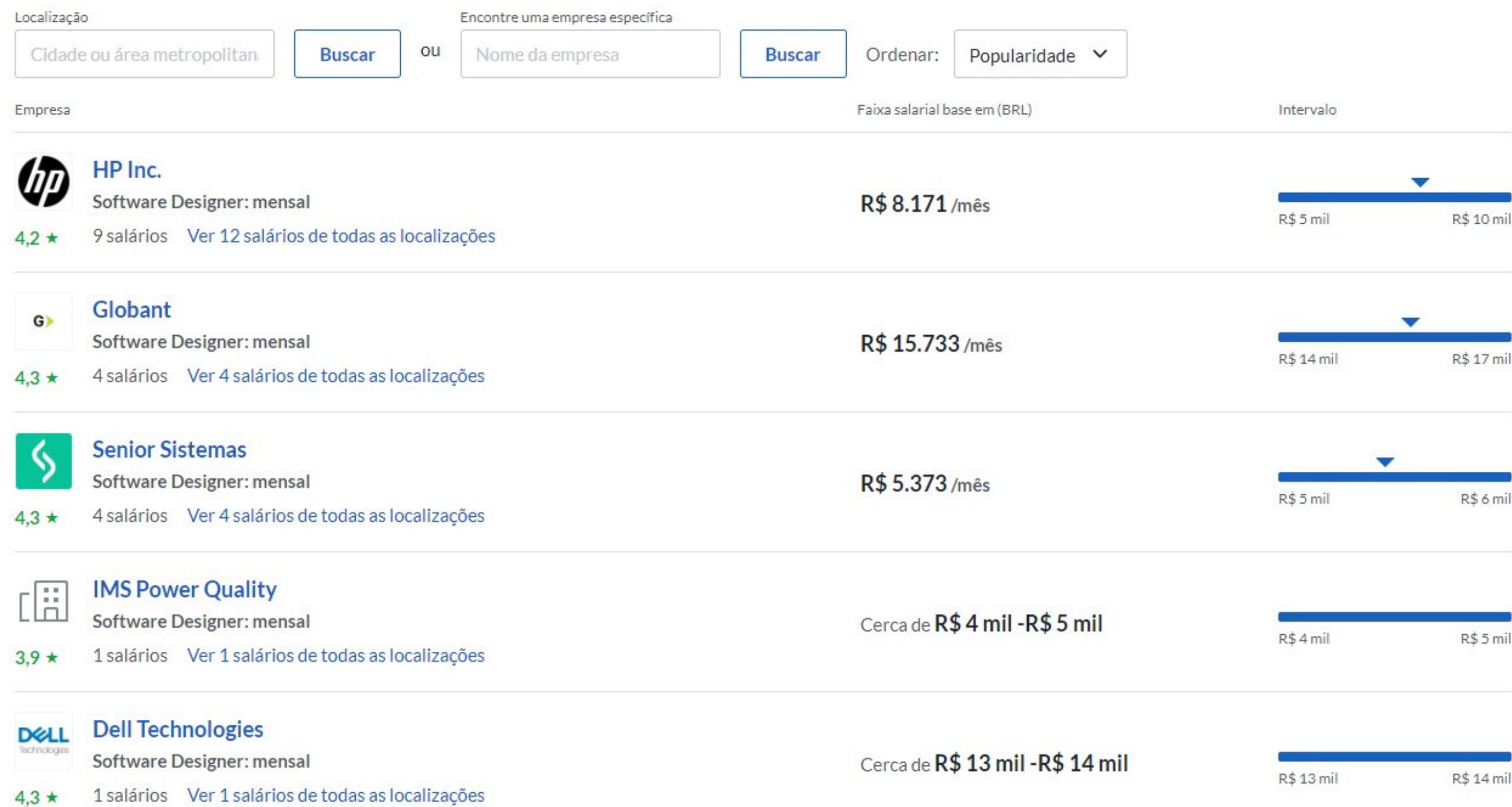
# Jobs and Carriers

MERCADO DE TRABALHO

SKILLS

# DESIGN DE SOFTWARE - MERCADO

Salários em Brasil



Média Salarial > R\$ 8.000,00

fonte: [https://www.glassdoor.com.br/Salários/software-designer-salário-SRCH\\_K00,17.htm](https://www.glassdoor.com.br/Salários/software-designer-salário-SRCH_K00,17.htm)

# SKILLS DE UM DESENVOLVEDOR DE SOFTWARE E DESIGNER DE SOFTWARE

# DESENVOLVEDOR

---

pessoa responsável por identificar, projetar, instalar e testar o sistema de software que desenvolveram para a empresa construir infraestrutura para milhões de usuários e reduzir as violações de segurança para clientes ou usuários. Eles são simplesmente mentes criativas por trás de programas de computador.

:p DESIGN

---

pessoa responsável por identificar o problema de design e preparar o plano para o aplicativo de software que, por sua vez, satisfaz os requisitos funcionais do problema. Geralmente, eles exigem um conhecimento profundo de matemática e ciências porque dá o privilégio de alterar a funcionalidade e a aparência.

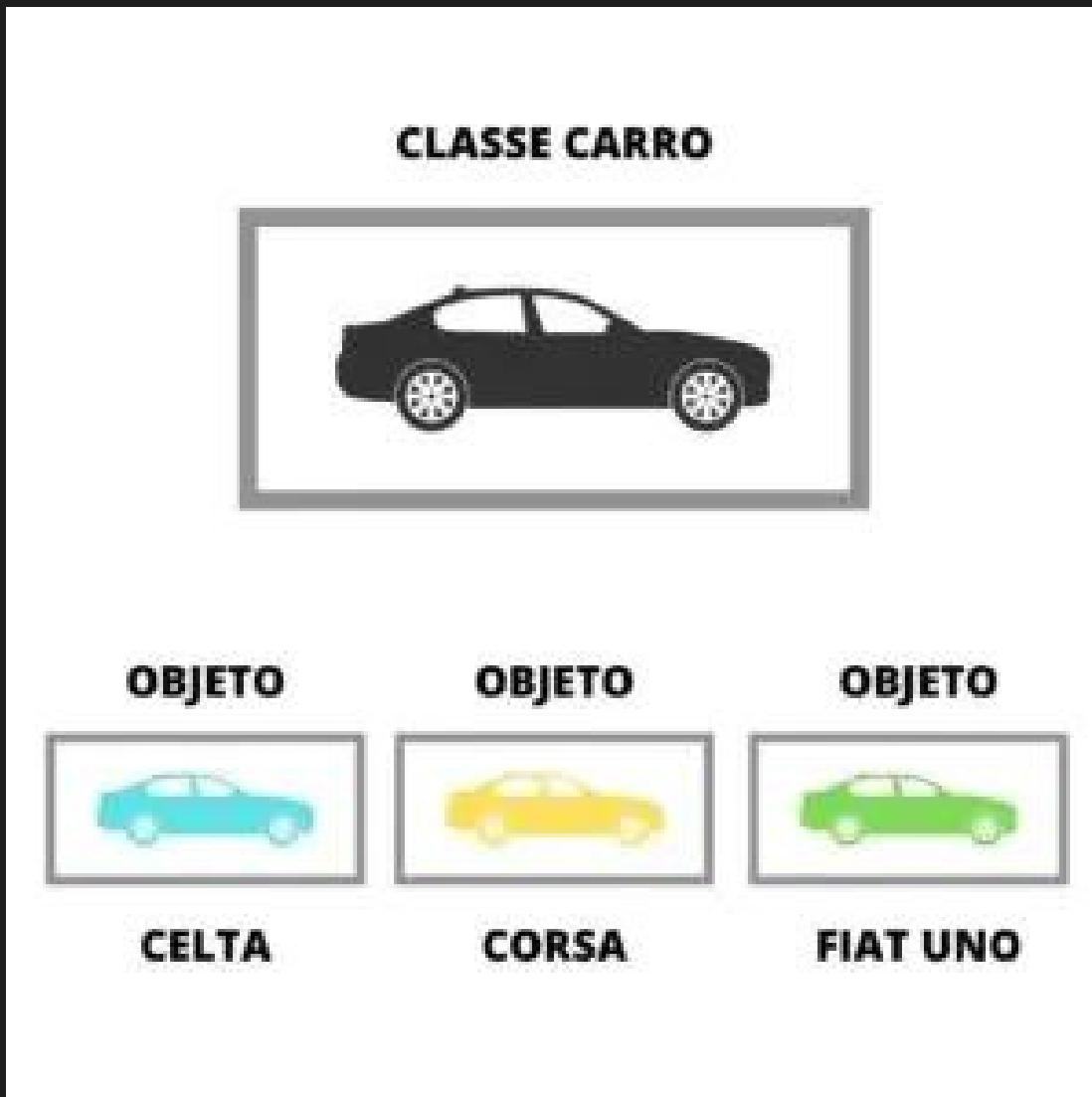


Part 3:

# Design Patterns



# Objeto e Instancia



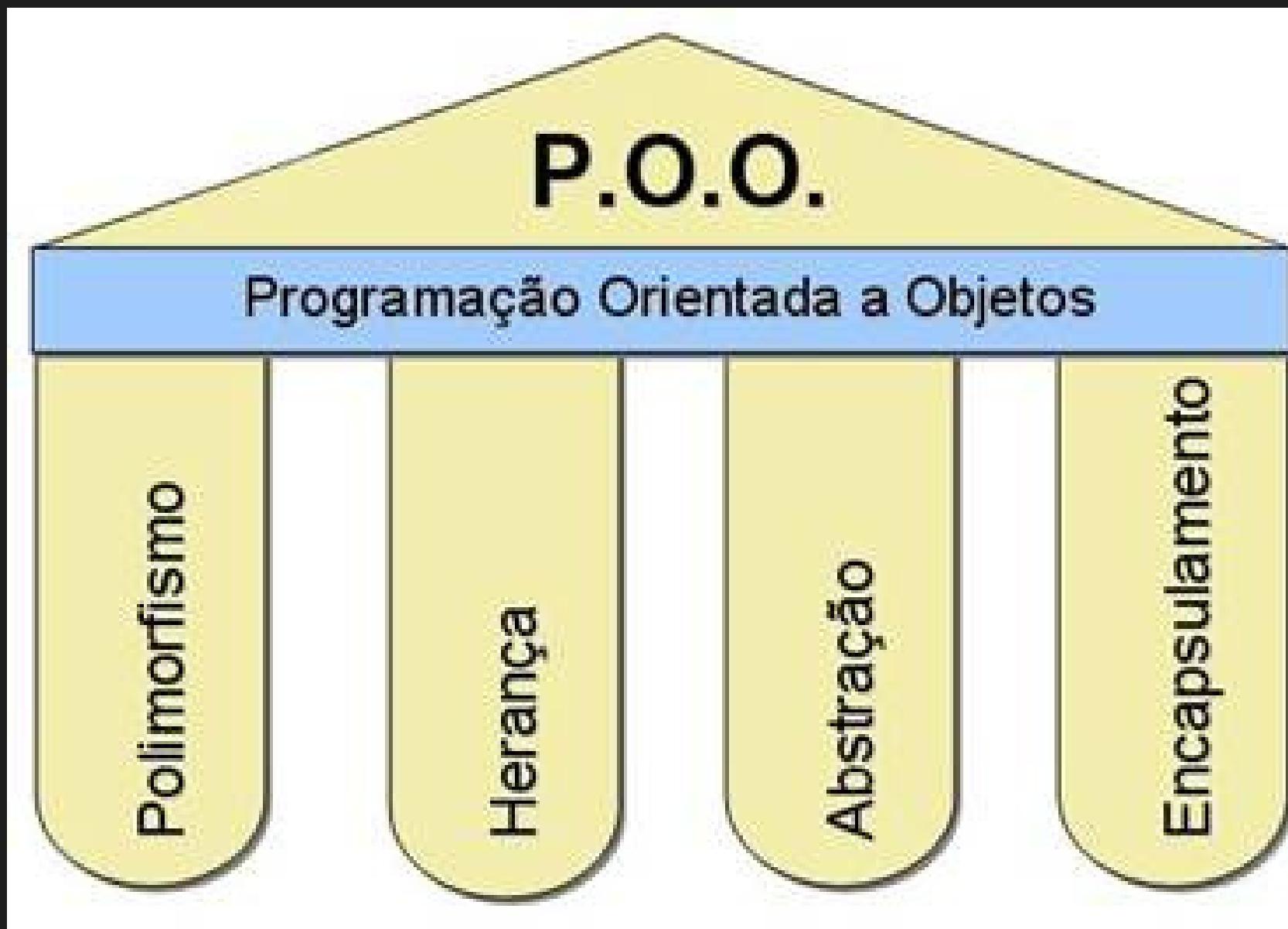
Seu carro é um objeto e onde você comprou ele já existiam outros objetos ou carros. Inclusive, muitos bem parecidos com o seu.

Por mais que existam carros igual em diversos aspectos, para o departamento de trânsito, todo carro é único, pois todo veículo tem o seu registro único no departamento de trânsito.

Dessa forma, por mais diferentes que os atributos do outro carro seja, ele ainda é considerado carro. Dessa forma, seu objeto pode ser classificado, ele pertence a uma classe, que é de carros. Então, seu carro nada mais é que uma instancia da classe chamada "carros".



# Os 4 pilares da Programação Orientada a Objetos



Para entendermos exatamente do que se trata a orientação a objetos, vamos entender quais são os requerimentos de uma linguagem para ser considerada nesse paradigma. Para isso, a linguagem precisa atender a quatro tópicos bastante importante



GoF - Design Patterns

fonte: <https://www.opus-software.com.br/design-patterns/#>

# O que são Design Patterns?

Design Patterns ou padrões de projetos são soluções generalistas para problemas recorrentes durante o desenvolvimento de um software. Não se trata de um framework ou um código pronto, mas de uma definição de alto nível de como um problema comum pode ser solucionado.



Mais de **+20**  
padrões  
Distribuídos em 3

Grupos:

- Criacional
- Estrutural
- Comportamental

### Creational Design Patterns

- Abstract Factory
- Builder
- Factory Method
- Prototype
- Singleton

### Structural Design Patterns

- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight
- Proxy

### Behavioral Patterns

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer



# Clean Code?

```
<!DOCTYPE html>
<html id="home-layout">
  <head>
    <meta name="content-type" content="text/html; charset=UTF-8"/>
    <title>Clean Code Pro</title>
    <link href="https://www.alura.com.br/assets/home.css" rel="stylesheet"/>
  </head>
  <body>
    <div id="main">
```



## Legível

Um código comprehensivo possibilita a identificação de pontos que precisam ser melhorados. Passamos mais tempo lendo código do que escrevendo então, quanto mais fácil for ler o código menos esforço fazemos para entendê-lo.

## Testável

Devemos testar nossos código, pois isso vai dar-nos segurança para podermos alterá-los. E garantir que os cenários que previmos estão de acordo com o esperado.

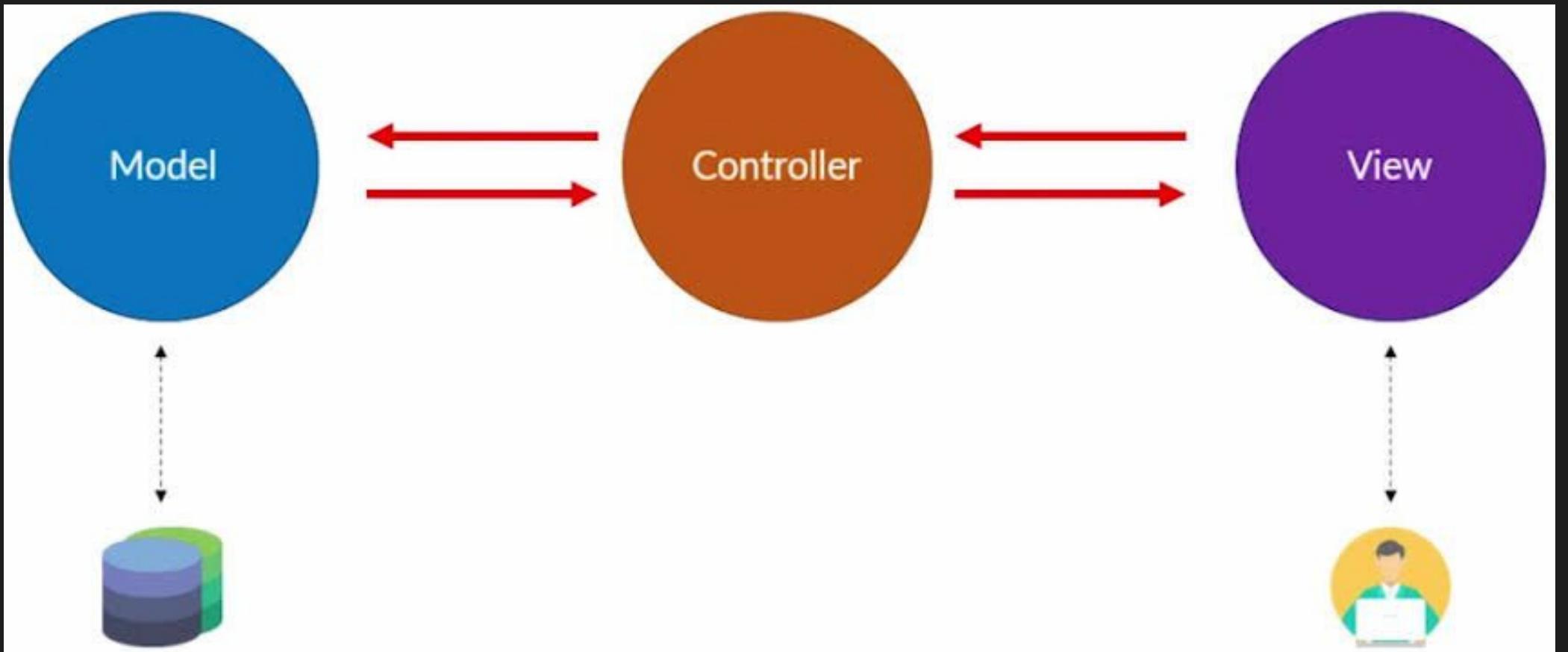
## Fácil de ser mantido

Nosso código deve passivo de alteração tanto para adição de novas funcionalidades, quanto para aumentar a legibilidade ou manutenibilidade.



# M.V.C

As principais vantagens de adotar um padrão como o MVC que se caracteriza pela facilidade na obtenção de múltiplas visões dos mesmos dados, desacoplagem da interface da lógica da aplicação, entre outras vantagens





Part 4:

# Design Principles



# Princípios de design:

Princípio *Keep It Simple, Stupid (KISS)*

Princípio *You Aren't Gonna Need It (YAGNI)*

Princípio *Don't Repeat Yourself (DRY)*





# Design patterns & Design principles

Padrões de projeto são soluções de codificação de baixo nível para projetos complexos. Essas soluções foram obtidas por tentativa e erro por vários desenvolvedores de software durante um período de tempo bastante substancial. Existem diferentes categorias de padrões de projeto.

Princípios de design são considerados um conjunto de diretrizes. Essas diretrizes ajudam a projetar sistemas complexos e ajudam os desenvolvedores a ter conhecimento abstrato sobre a solução de projetos complexos.



DESIGN DE SOFTWARE

PROJETO DE SOFTWARE



"Com grandes poderes vem  
grandes responsabilidades."

GRASP &  
SOLID



# GRASP Targets

Craig Larman, cientista da computação, afirma que "a ferramenta crucial de projeto para desenvolvimento de software é uma mente bem educada em princípios de projeto. Não é UML ou qualquer outra tecnologia". [1] Assim, GRASP é realmente um conjunto de ferramentas mentais, um auxílio de aprendizagem para ajudar no projeto de software orientado a objetos.

*fonte:*

[https://pt.wikipedia.org/wiki/GRASP\\_\(padrão\\_orientado\\_a\\_objetos\)](https://pt.wikipedia.org/wiki/GRASP_(padrão_orientado_a_objetos))

# UTILIZANDO UML E PADRÕES

3<sup>a</sup> Edição

Uma introdução à análise e ao projeto orientados  
a objetos e ao desenvolvimento iterativo

**CRAIG LARMAN**

Prefácio de Philippe Kruchten



Capa do livro onde é explicado em detalhes cada um dos padrões GRASP

# General responsibility assignment software patterns



GRASP

## Responsabilidades

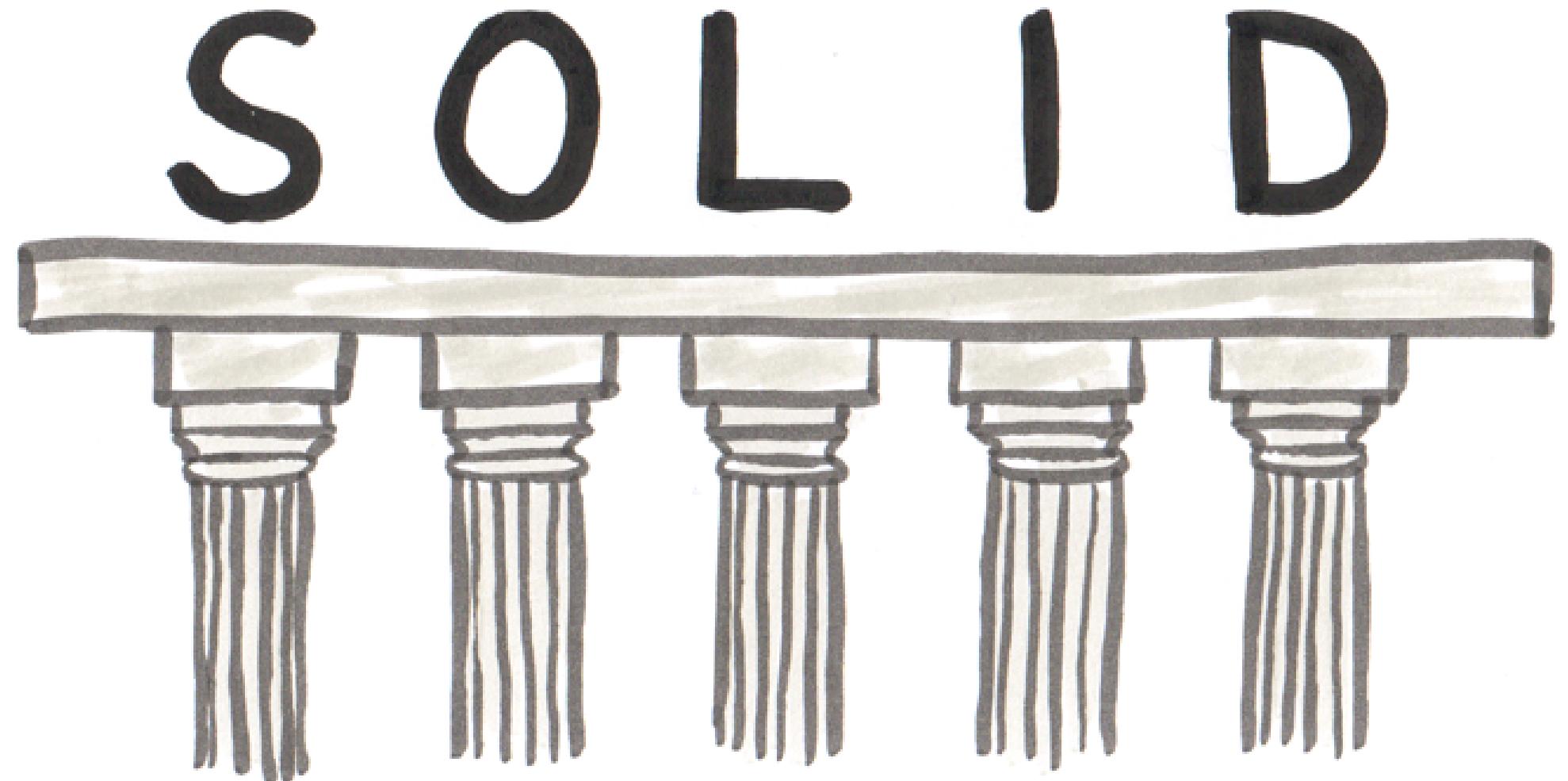
- São relacionadas com obrigações de um objeto em termos de seu comportamento.
- Responsabilidades do fazer de um objeto incluem:
  - Fazer alguma coisa por si [um cálculo, criar um outro objeto]
  - Iniciar ações em outros objetos
  - Controlar e coordenar atividades em outros objetos
- Responsabilidades do conhecer de um objeto incluem:
  - Conhecer objetos relacionados
  - Conhecer dados privados e encapsulados

# DESIGN DE SOFTWARE

```
s MovieSessionService {  
    movieSessionRepository sessionRepository;  
    UnavailabilityRepository unavailabilityRepository;  
    Converter<MovieSessionDTO, MovieSession> converter;  
  
    MovieSessionService(MovieSessionRepository sessionRepository, UnavailabilityRepository unavailabilityRepository, Converter<MovieSessionDTO, MovieSession> converter) {  
        this.sessionRepository = sessionRepository;  
        this.unavailabilityRepository = unavailabilityRepository;  
        this.converter = converter;  
  
        result<MovieSession> create(MovieSessionDTO dto) {  
            Session session = converter.toEntity(dto);  
  
            MovieSession sessions = sessionRepository.listAllByTheaterId(dto.getTheaterId());  
  
            sessions.stream().anyMatch(s -> s.getStart().equals(session.getStart()) &&  
                s.getEnd().isBefore(session.getEnd()))  
            return Result.fail(SessionConflictException.class, session);  
  
            sessions.stream().anyMatch(s -> session.getStart().isBefore(s.getStart()) ||  
                session.getEnd().isAfter(s.getEnd()))  
            return Result.fail(SessionConflictException.class, session);  
  
            Unavailability unavailabilities = unavailabilityRepository.listAllByTheaterId(dto.getTheaterId());  
  
            unavailabilities.stream().anyMatch(u -> u.getStart().equals(session.getStart()) &&  
                u.getEnd().isAfter(session.getEnd()))  
            return Result.fail(UnavailablePeriodException.class, session);  
  
            unavailabilities.stream().anyMatch(u -> session.getStart().isBefore(u.getStart()) ||  
                session.getEnd().isAfter(u.getEnd()))  
            return Result.fail(UnavailablePeriodException.class, session);  
  
            sessionRepository.save(session);  
            return Result.success(session);  
        }  
    }  
}
```

**Histórico**  
Os princípios SOLID foram apresentados pela primeira vez pelo famoso cientista da computação Robert J. Martin (também conhecido como Uncle Bob) em seu trabalho (texto em inglês) lançado no ano 2000. A abreviação SOLID, no entanto, foi apresentada mais tarde, por Michael Feathers.

PROJETO DE SOFTWARE



fonte:<https://www.freecodecamp.org/portuguese/news/os-principios-solid-da-programacao-orientada-a-objetos-explicados-em-bom-portugues/>



# S.O.L.I.D

- Single Responsibility Principle (Princípio da responsabilidade única)
- Open-Closed Principle (Princípio aberto/fechado)
- Liskov Substitution Principle (Princípio da substituição de Liskov)
- Interface Segregation Principle (Princípio da segregação da interface)
- Dependency Inversion Principle (Princípio da inversão da dependência)



# Código Robusto + Baixa Manutenção



Estamos nos referindo à um código com Baixo Acoplamento, Alta Coesão, usando SOLID, Imutabilidade (quando fizer sentido), aplicando Design Patterns, minimizando Side Effects, maximizar o uso de Funções Puras e várias outras coisas.



# Thank you!

## Team:



Cristian

MINHA MOTO TÁ BOA!



Gustavo

CORRE PRO BERÇO



Leandro

EU ODEIO ESSE CARA



Elisa

POUSADA 443



Juno

RECUPERAÇÃO IES



Wayne

VOU PRA BLUMENAU