



Ciência da **Computação**

Programação Orientada a Objetos
Prof. Luciano Rodrigo Ferretto

2024/02 - Turma B - Paradigma de Linguagens de Programação

Grupo do WhatsApp



Nossa Aula de hoje

- Estrutura do AVA
- Um pouco sobre nós
- Algumas dicas para um bom aprendizado
- Como será nosso semestre
- Paradigmas de Programação
- Paradigmas Imperativos
- Entregáveis
- Questões de Estudo

Introdução - AVA

Turma B:

<https://imed.mrooms.net/course/view.php?id=14185#section-0>

Em nosso AVA temos a seção Introdução, no qual consta os contatos que vocês podem estar utilizando para falar com o professor, assim como o link para no nosso grupo de WhatsApp

Feedback - AVA

- Seu feedback é essencial para nós continuarmos aprimorando nossas aulas e proporcionando a melhor experiência de aprendizado possível. Sua opinião sobre o conteúdo, metodologia e dinâmica das aulas é inestimável para nós.
- Cada comentário que vocês compartilham é uma oportunidade para melhorarmos juntos. Sejam honestos e específicos em suas observações, pois é através delas que podemos identificar pontos fortes e áreas de melhoria.
- Lembrem-se de que sua voz tem o poder de moldar o rumo do nosso curso. Ao expressarem suas opiniões, estão contribuindo para o desenvolvimento de uma comunidade de aprendizado mais eficaz e inclusiva.
- Juntos, podemos criar um ambiente onde o diálogo aberto e construtivo nos guie para alcançar excelência acadêmica.

Feedback das aulas

→ 👁 ⋮

Seu feedback é de fundamental importância para aprimorarmos continuamente a nossas aulas.

Obrigado por sua contribuição!

Feedback Constante das aulas de CC 2024.2 - Paradigma de Linguagem de Programação B

Não é necessário estar logado com uma conta google para responder esse formulário.
Assim, sua identidade é mantida em sigilo.

Em uma escala de 1 a 5, avalie os itens abaixo referente as aulas:

[Faça login no Google](#) para salvar o que você já preencheu. [Saiba mais](#)

* Indica uma pergunta obrigatória

E-mail *

Selecione o Professor: *

Escolher

▼

[Próxima](#)[Limpar formulário](#)

Plano de Ensino - AVA

Nosso Plano de Ensino está disponível no nosso AVA

Turma B:

<https://imed.mrooms.net/course/view.php?id=14185#section-1>

E também no repositório git da turma

Turma B: https://github.com/luciano-ferretto/ParadLingProg_202402_TurmaB

Material de Apoio - AVA

Turma B: <https://imed.mrooms.net/course/view.php?id=14185#section-2>

A seção **Material de Apoio** tem por finalidade oferecer aos alunos conteúdos essenciais para um bom aprendizado deste módulo. Estes materiais não são de consumo obrigatório, porém são premissas indicadas para um bom aprendizado. **Todas as trilhas indicadas são gratuitas e estão no YouTube.**

- **Obs.: recomendado abrir no YouTube para ver as demais aulas da trilha.**
- **Nesta seção também encontramos o link do nosso repositório git**

Preparação do Ambiente para as Aulas - AVA

Turma B:

<https://imed.mrooms.net/course/view.php?id=14185#section-4>

A seção **Preparação do Ambiente para as Aulas** tem por finalidade oferecer aos alunos a lista dos softwares que serão utilizados, assim como algumas dicas e instruções sobre a instalação e configuração dos mesmos.

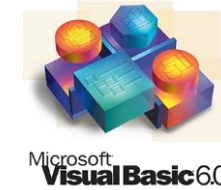
Entregáveis

Mas o que é isso???



Luciano Rodrigo Ferretto

- Academia
 - Bacharel em Sistemas de Informação
 - Mestre em Computação Aplicada
 - Sistemas de Recomendação
- Atuação Profissional (na área de desenvolvimento)
 - Início em meados de 2001 com Microsoft Visual Basic 5.0, passando depois para o VB 6.0
 - Algumas “aventuras” no C# com a chegada do .NET
 - Atualmente programando JavaScript e principalmente com TypeScript para aplicações node js (REST Web Services)
 - Também atuando em ReactNative
 - E por fim, “sempre” (kkkk) desenvolvendo em Java
 - JSP
 - REST





E você ????

- Fale um pouco sobre você para nos conhecermos:
 - Sua experiência com programação?
 - Quais as linguagens?
 - Estás trabalhando além de estudar?
 - Onde?
 - Trabalhas na área da computação?
 - Trabalhas com programação?
- Sua expectativa com a disciplina?





Vamos conversar um pouco!!!

Antes de falarmos da nossa disciplina e como será nosso semestre.. Vamos conversar um pouco.



Alguns conselhos...

- Aprender a aprender!
- Faça perguntas!
- Participar da comunidade e montar um portfólio.
- Tenha referências na área, mas não siga a opinião dos outros como verdade absoluta.
- Conceito é mais importante que a tecnologia em si.

**Se Conselho fosse
bom se Vendia,
não se Dava!**

Um Milagre
Cada Dia





Conceito é mais importante que a tecnologia em si

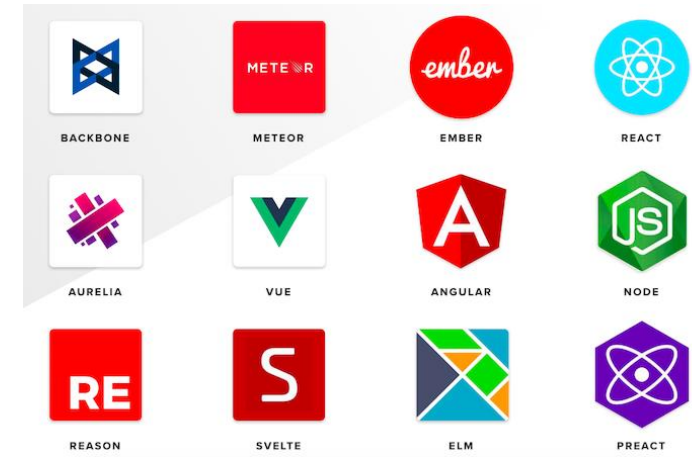
Conceito é mais importante que a tecnologia em si



- Desenvolvimento de Software = Resolução de problemas
- Linha de raciocínio/mecanismos -> respostas para problemas
 - É mais importante do que dominar a tecnologia x ou y.



Conceito é mais importante que a tecnologia em si



- Vamos pensar no JavaScript...
 - Vários Frameworks
- O impulso é sempre dominar tudo de React, de Angular, do que quer que seja.
- Esse pensamento, a curto prazo, pode resolver seu problemas, você pode conseguir um emprego ou atingir qualquer que seja seu objetivo, mas, a longo prazo, em termos de carreira, todas as tecnologias vão mudar!

Você sabe dirigir???

Com qual carro você aprendeu a dirigir?



No mundo real também vale essa premissa.

- Eu aprendi a dirigir em um Opala velho do pai de um amigo meu...
- Carburado
- Manual 4 marchas
- Direção queixo-duro
- Fazer pegar no frio
- Bebia mais do que a turma inteira...



Você sabe dirigir???

- Depois, de muito tempo, consegui uma Dodge Journey V6...
- Injeção Eletrônica
- Automático 6 marchas
- Direção hidráulica
- Mas bebia mais do que o opala velho...



Então, o conceito de dirigir é o mesmo independente do veículo!

- Se eu aprendo em um Opala velho eu consigo dirigir uma Journey ???

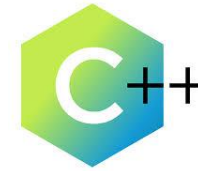
É CLARO, e isso vale para a área de TI.

Se você aprende bem o conceito, vai conseguir trabalhar com qualquer tecnologia



Exemplo: Conceitos de Orientação a Objetos

- Abstração
- Encapsulamento
- Herança
- Polimorfismo



- **Conceitos iguais para todas as linguagens OO**

Mas não vamos exagerar também...

O poliglota iletrado

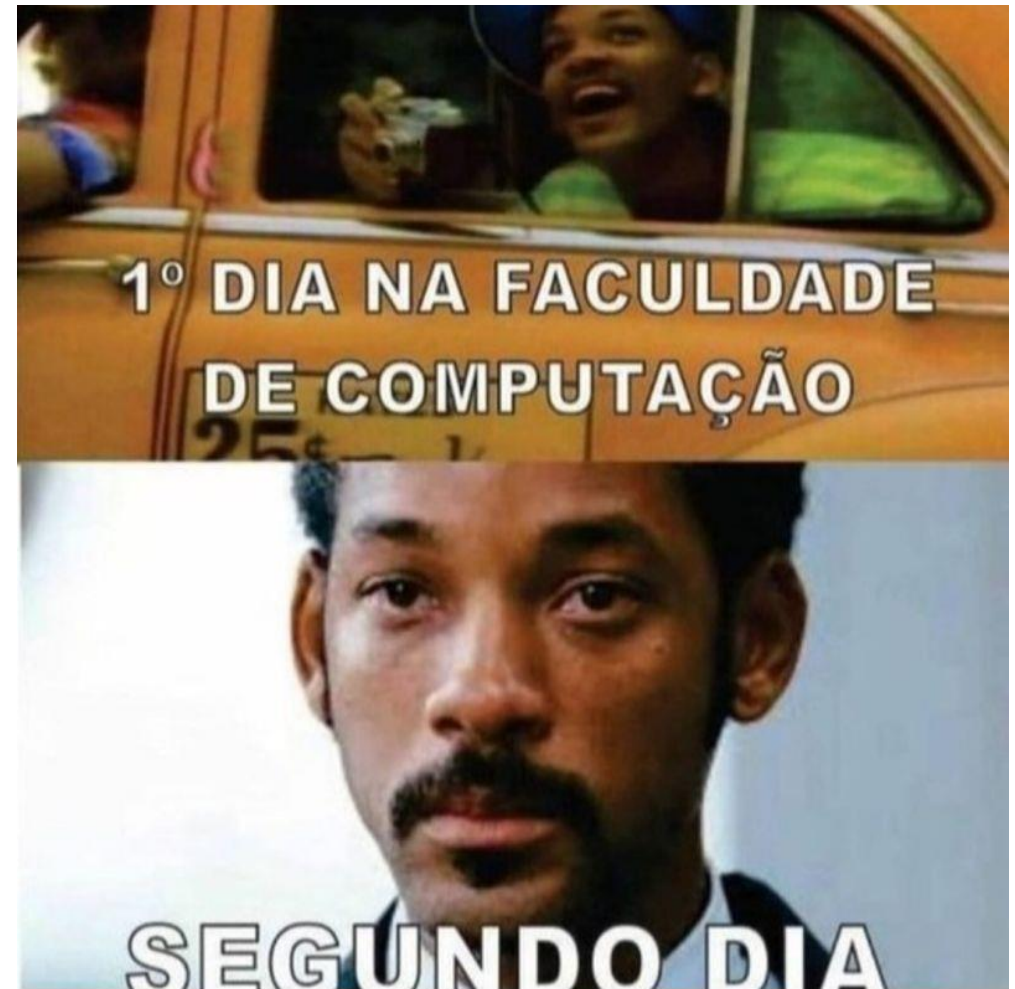
Fluente em diversos idiomas. **Desconhece assuntos de linguística**. Incapaz de generalizar e abstrair a partir de estruturas linguísticas presentes nos idiomas que ele domina e, portanto, para ele, é sempre igualmente **penoso aprender um novo idioma**. Tudo é sempre novidade.

O linguista teórico

Entende todas as estruturas linguísticas presentes em qualquer idioma, **teoriza** a respeito delas. É **incapaz**, no entanto, **de utilizar confortavelmente** os idiomas estudados em uma conversação fluente.

Nosso objetivo é alcançar um **equilíbrio** entre teoria e prática, para que não sejamos nem um nem outro, mas aproveitemos as vantagens dos dois.

Aqui entra a
Faculdade!





E a faculdade ???

- Claro que podemos aprender sozinhos mas, a faculdade nos fornece uma maneira de entender os conceitos mais estruturados, nem sempre relacionados com o mercado de trabalho.
- Por exemplo: Complexidade de algoritmos
 - Apesar de estrutura de dados e complexidade de algoritmos serem subestimados, são necessários!
 - Mas claro, também não são imprescindíveis para você saber antes de entrar no mercado. Mas se você souber, vai ser o diferencial entre subir na carreira ou ficar estagnado.

E a faculdade ???

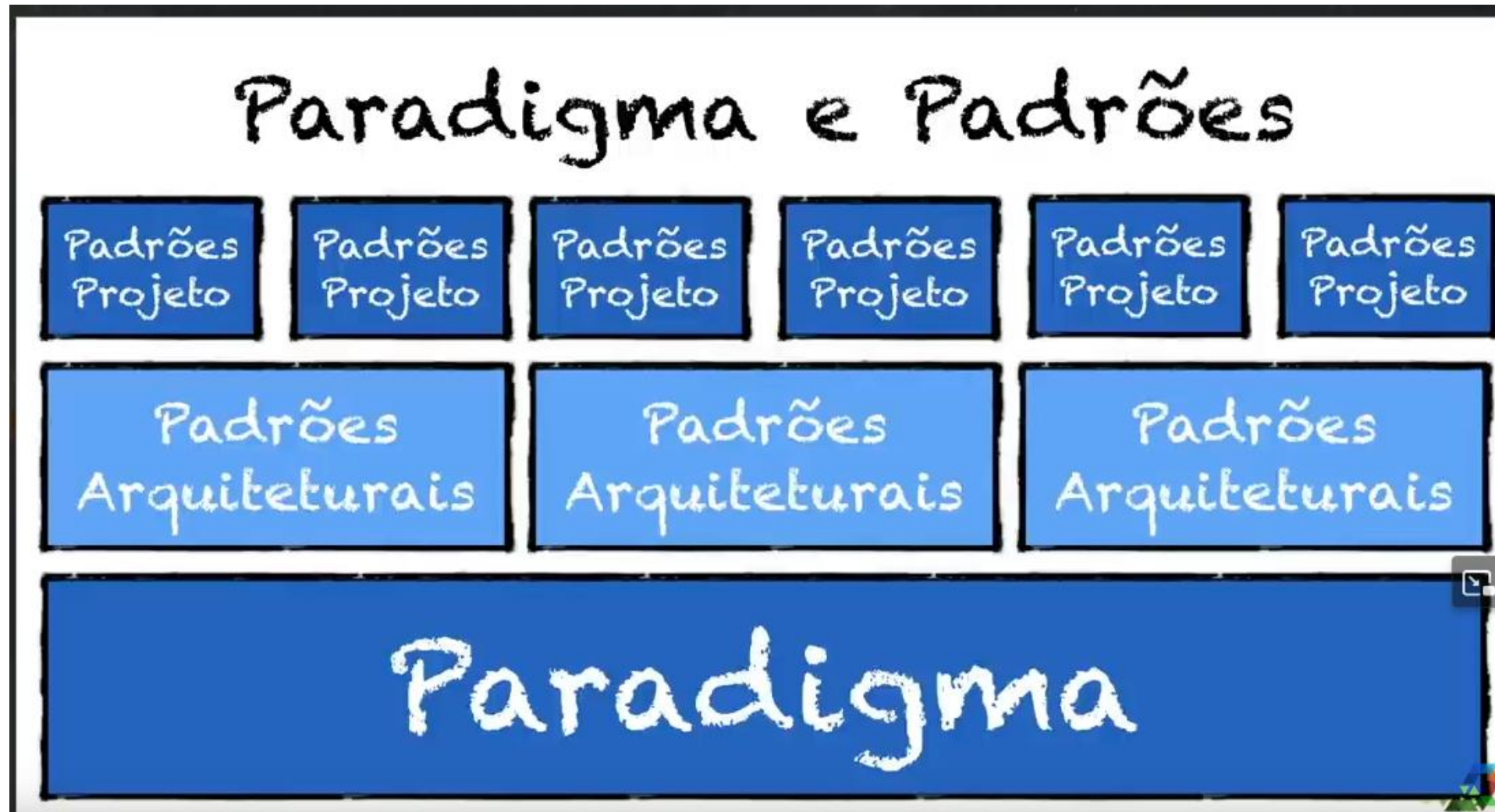
<https://www.youtube.com/watch?v=nlvtKEL8JzA>

Tem o teu

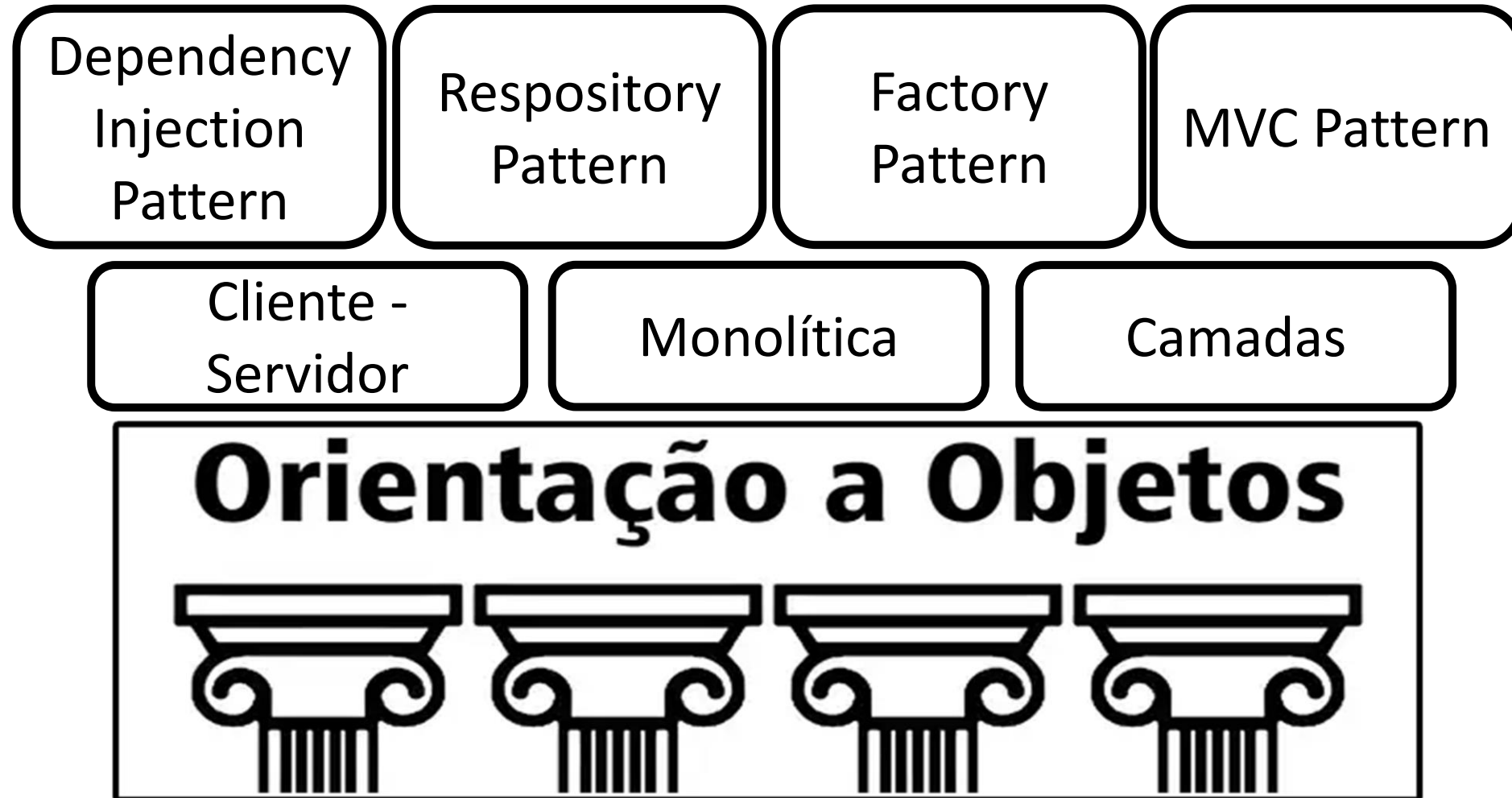
Agora sim! Vamos falar do nosso semestre!

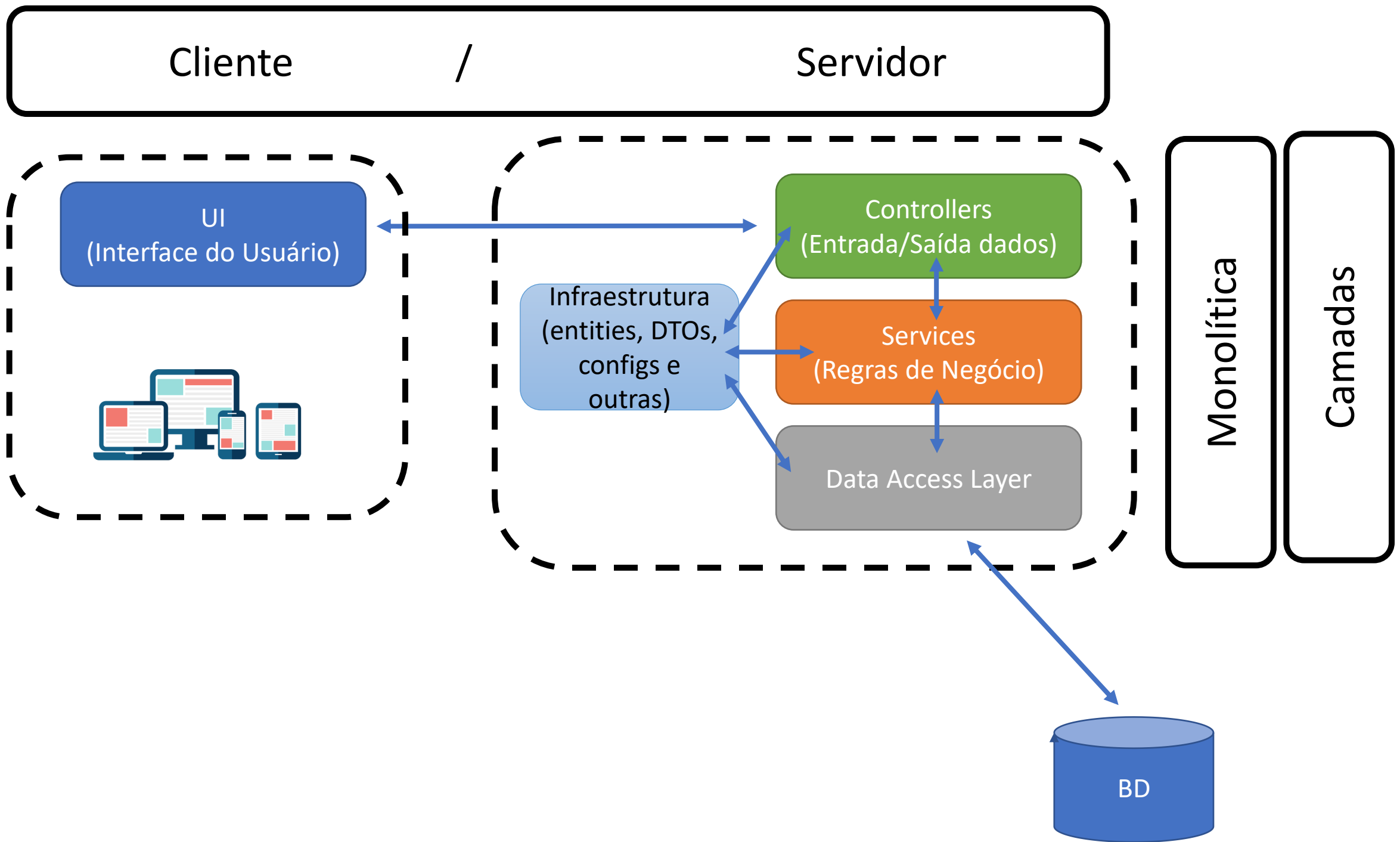


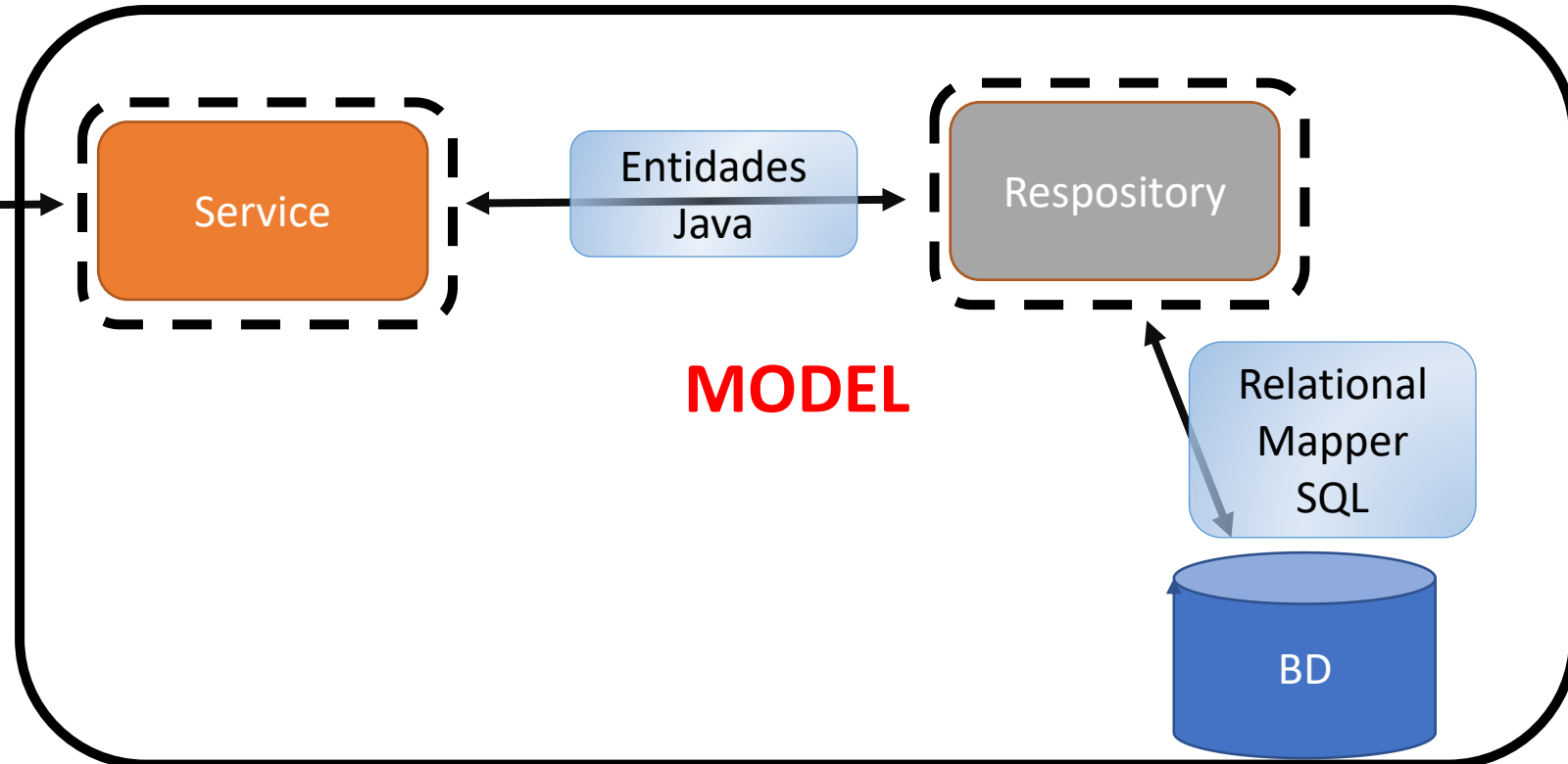
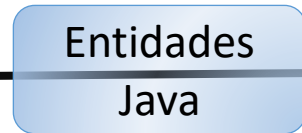
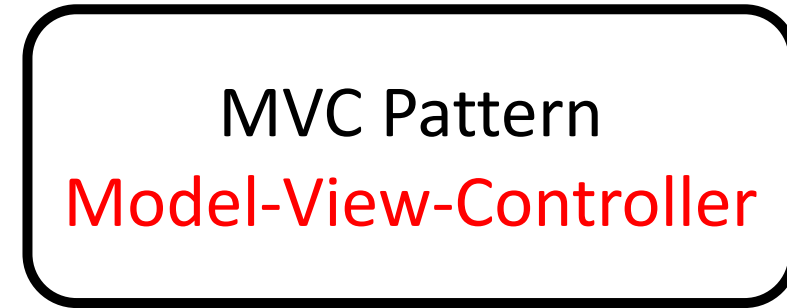
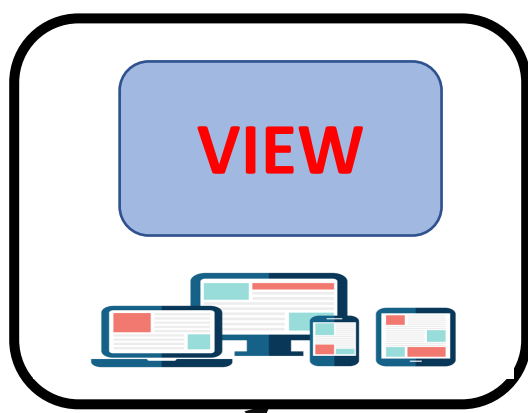
O que iremos estudar?



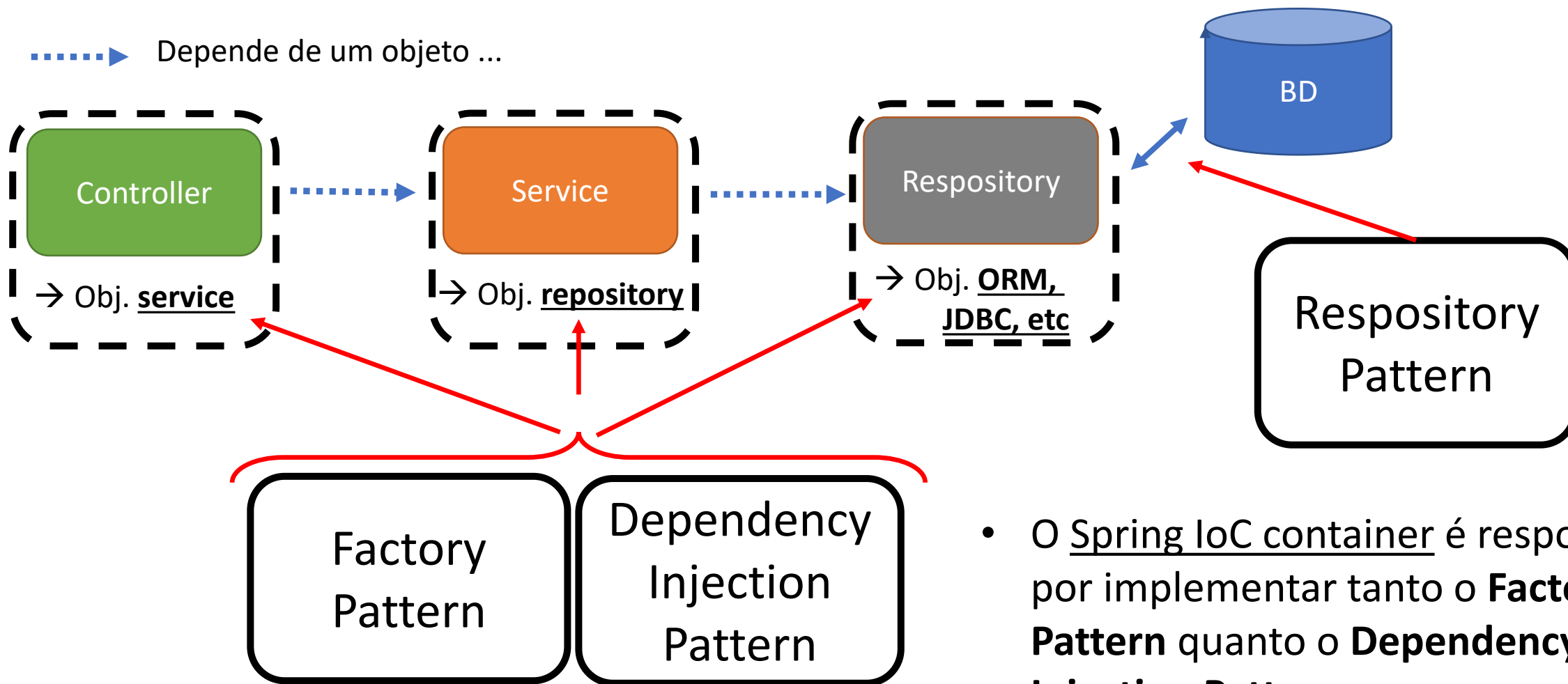
Sabia que vocês já utilizaram esses conceitos no semestre passado ????








.....> Depende de um objeto ...



- O Spring IoC container é responsável por implementar tanto o **Factory Pattern** quanto o **Dependency Injection Pattern**.
- Já o Spring Data JPA é o responsável pela implementação do **Repository Pattern**.

Factory Pattern

```
package br.edu.atitus.oop.denguealerta.services;  
  
import java.time.LocalDateTime;  
  
@Service  
public class FocoService extends GenericService<FocoEntity>{
```



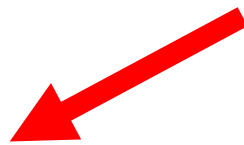
- Com a anotação @Service, indicamos que o Spring IoC container deve ser responsável por criar e gerenciar o ciclo de vida dos objetos dessa classe, implementando o **Factory Pattern** e tirando a responsabilidade do desenvolvedor.
- Dessa forma, o container instanciará os beans conforme necessário e controlará seu ciclo de vida, promovendo um gerenciamento eficiente e centralizado dos componentes da aplicação.

Dependency Injection Pattern

```
@RestController
@RequestMapping("/ws/foco")
public class FocoController extends GenericController<FocoEntity, FocoDTO>{

    private final FocoService focoService;


    public FocoController(FocoService focoService) {
        super();
        this.focoService = focoService;
    }
}
```



- No método construtor, o Spring IoC container injeta a dependência necessária através do **Dependency Injection Pattern**, aliviando o desenvolvedor da tarefa de gerenciar manualmente a criação e a ligação das dependências.

Repository Pattern

```
package br.edu.atitus.oop.denguealerta.repositories;  
  
import java.util.List;  
  
public interface FocoRepository extends JpaRepository<FocoEntity, UUID>{
```



- Ao estender a interface JpaRepository, implementamos o **Repository Pattern**, permitindo que o Spring Data JPA gerencie as operações de persistência e consulta automaticamente, sem a necessidade de escrever código repetitivo para essas operações.
- Isso facilita a interação com o banco de dados e mantém o código mais limpo e organizado.

Então é Verdade, Já usamos tudo isso!!!

Dependency
Injection
Pattern

Repository
Pattern

Factory
Pattern

MVC Pattern

Cliente -
Servidor

Monolítica

Camadas

Orientação a Objetos



Muito bem! Mas e agora?

- Então já usamos um Paradigma de Programação, três Arquiteturas e no mínimo quatro Design Patterns.
- O que vamos ver agora ???
- Agora, além de entender os diferentes conceitos que já aplicamos, vamos aprender mais ...



1 - Paradigmas de Programação

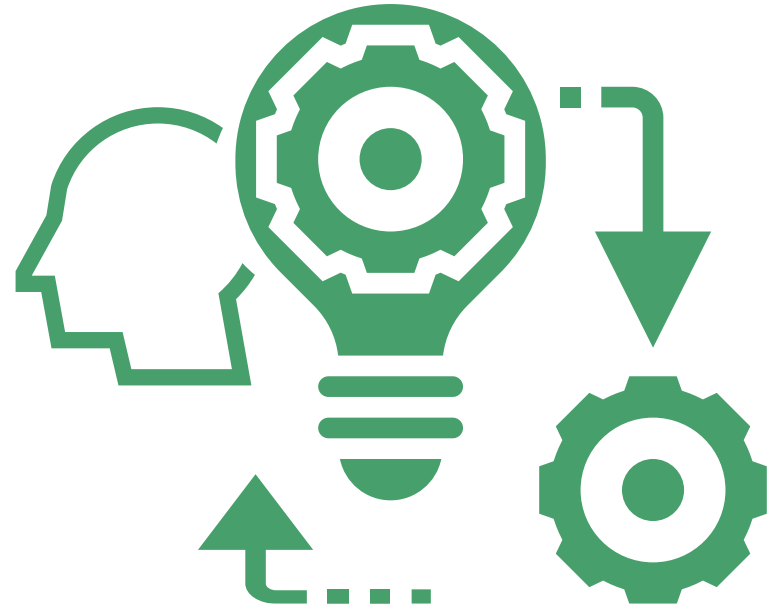
- Paradigma Imperativo
- Paradigma Declarativo
- Abordagens Complementares

2 - Padrões de Design de Software (Design Patterns)

- Padrões Criacionais
- Padrões Estruturais
- Padrões Comportamentais

3 - Microservices

- Configurações de Microservices
- Spring Boot Actuator
- Flyway
- Comunicação entre Microservices
- Serviço de Nomeação - Descoberta
- Gateway de API
- Observabilidade e Monitoramento
- Testes em Microservices



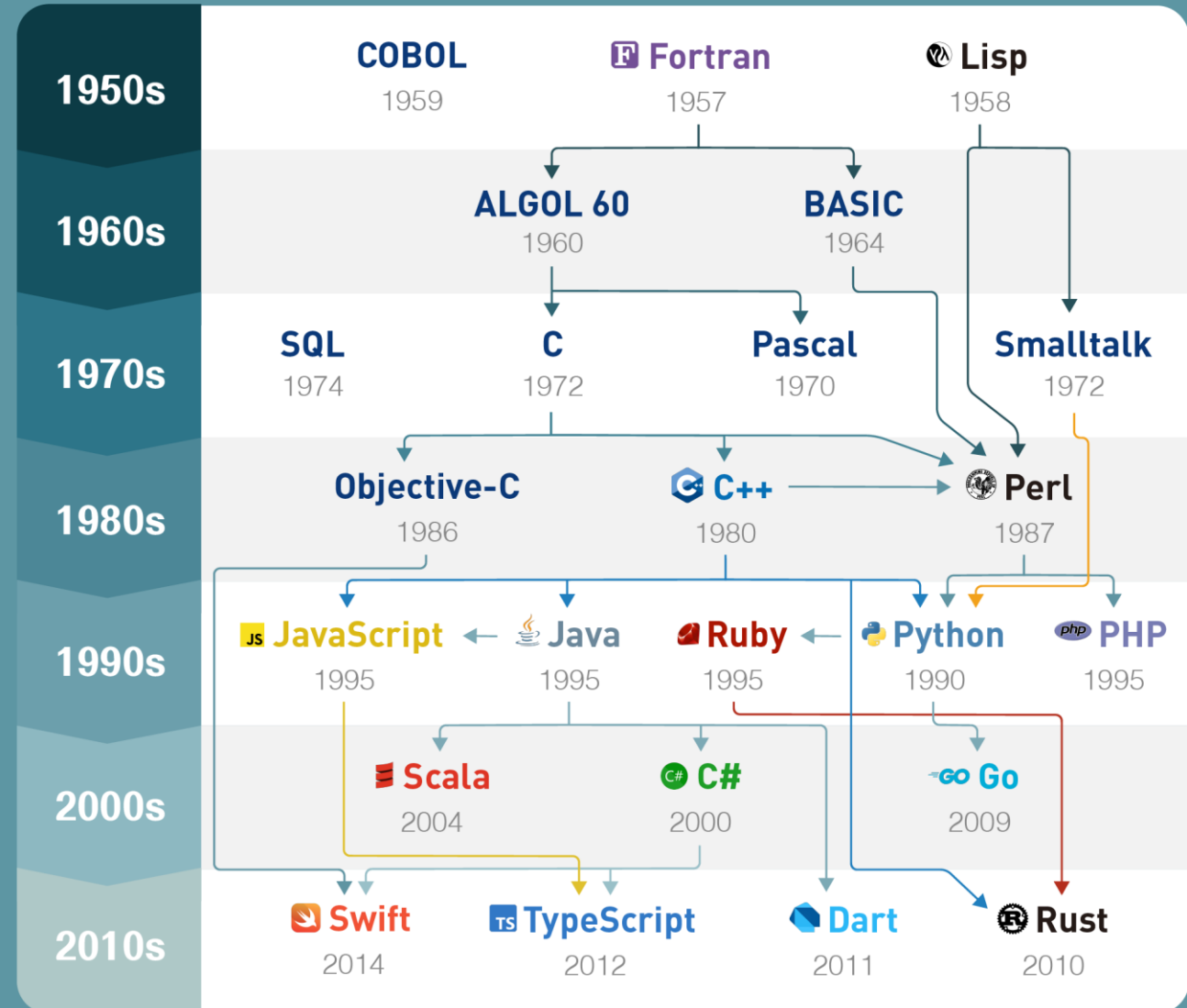
Paradigmas de Programação

Antes... Precisamos relembrarmos um pouco...

Quais as Linguagens de Programação existem?

Quais problemas/áreas elas se propõem a resolver/atender?

Era uma vez...



Aplicações Empresariais

As corporações perceberam, na década de 50, que poderiam **otimizar a manutenção de seus registros** e aumentar a **precisão** e a **confiabilidade** de suas operações através do uso de computadores e do desenvolvimento de alguns programas específicos.

- Folha de pagamento, contabilidade, controle estoque e produção, vendas on-line...

Linguagens para aplicações empresariais são caracterizadas pela capacidade de gerenciamento de **grande quantidade de dados**, produção de **relatórios** elaborados, e maneiras precisas para descrever e modificar os dados que gerenciam.

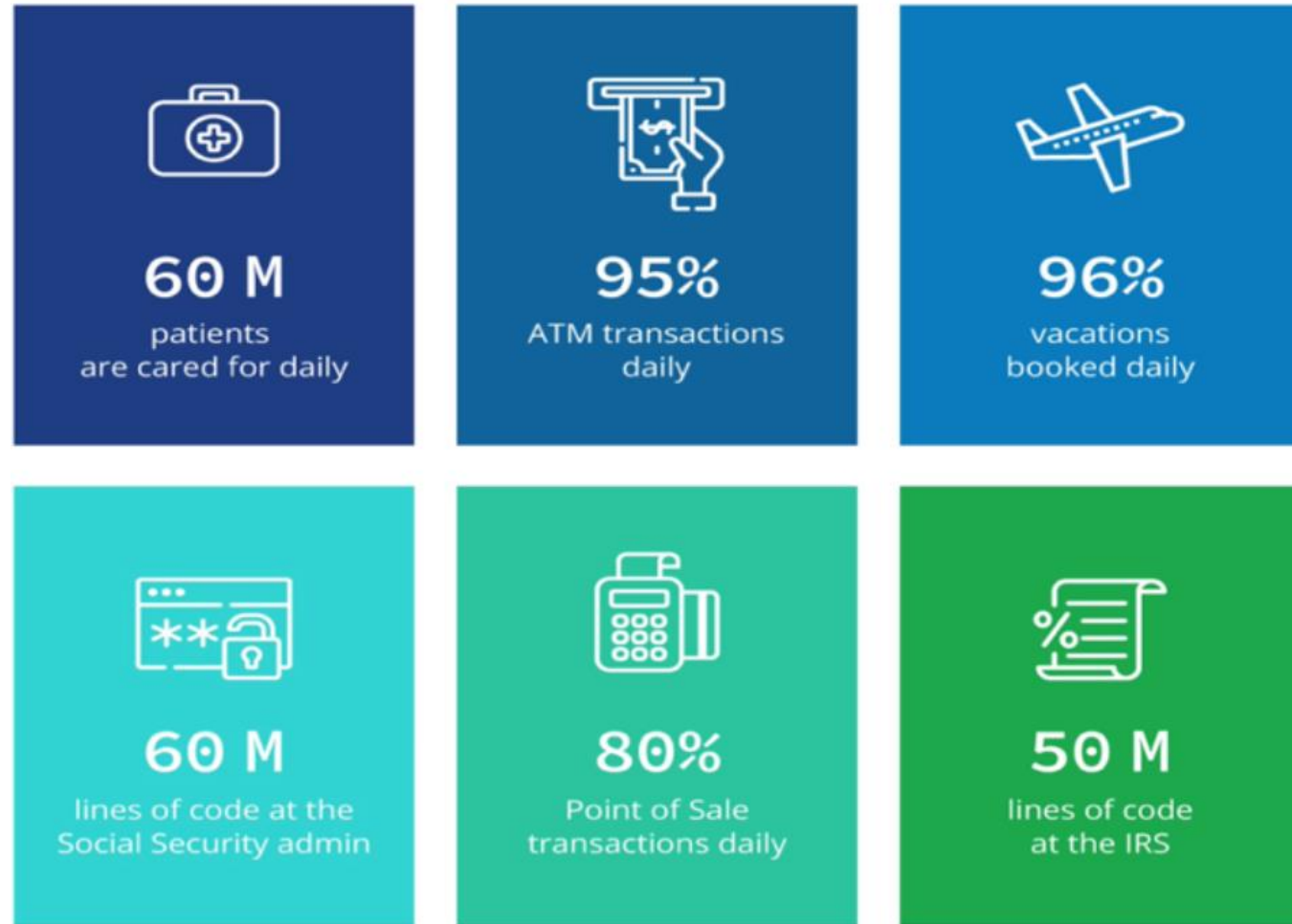
A linguagem mais utilizada, tradicionalmente, é o **COBOL** (*Common Business Oriented Language*), que usa a língua inglesa como base de sua sintaxe e suporta o estilo de programação **imperativo**.

- Normalmente, em conjunto com **SQL**, utilizada para especificação e manuseio de informações de bancos de dados relacionais.



Curiosidade sobre o Cobol

- A Figura ilustra os maiores sistemas rodando em COBOL, com base em dados de **2013**.
- De acordo com a Micro Focus, o número de linhas de código COBOL cresceu para uma faixa de **775-850 bilhões até 2023**.
- Este número fornece uma descrição clara da prevalência contínua de aplicações COBOL, sugerindo que sua substituição não ocorrerá em breve



Sistemas e Redes

Programadores de sistemas projetam e fazem a manutenção do **software básico**:

- Componentes do sistema operacional, software de redes, compiladores e depuradores de linguagens de programação, máquinas virtuais e interpretadores, sistemas embarcados e de tempo real (celulares, caixas eletrônicos, aeronaves, etc.).

Devem ser **eficientes**, são **utilizados quase continuamente**, e precisam de acesso a instruções de mais **baixo nível**, que permitam a comunicação mais direta com máquinas e dispositivos.

Por isso, a GRANDE maioria dessa programação é feita nas linguagens **C e C++**.

- **Curiosidade:** 95% do código do sistema Unix é escrito em C.

Algumas linguagens de *script* também têm sido muito utilizadas, em especial por administradores de sistemas. Por exemplo, um programa **awk** pode ser projetado para verificar a consistência em um arquivo de senhas de uma máquina Unix.

- Além de **awk**, se destacam nesse sentido: **Perl**, **Tcl/Tk** e **Python**.

Educação

Nas décadas de 60 e 70 algumas linguagens foram projetadas com a finalidade principal de serem simples e servirem como porta de entrada para alunos de programação.

Entre elas destacam-se o **BASIC** e o **Pascal** (proveniente do ALGOL).

O Pascal foi, durante muitos anos, a principal linguagem utilizada no início de cursos de computação de nível superior, mas foi gradualmente substituída por linguagens de maior “apelo comercial”, como C++, Python e Java, que são linguagens mais complexas, mas de uso imediato no mercado de trabalho.



World Wide Web

A área mais dinâmica para novas aplicações de programação é a **Internet**, que é o veículo que permite o comércio eletrônico e uma ampla gama de aplicações acadêmicas, governamentais e industriais.

A WWW é mantida por uma **diversificada coleção de linguagens**, que vão desde linguagens de marcação, como XHTML (que não é linguagem de programação!), até linguagens de programação de propósito geral, como o **Java**.

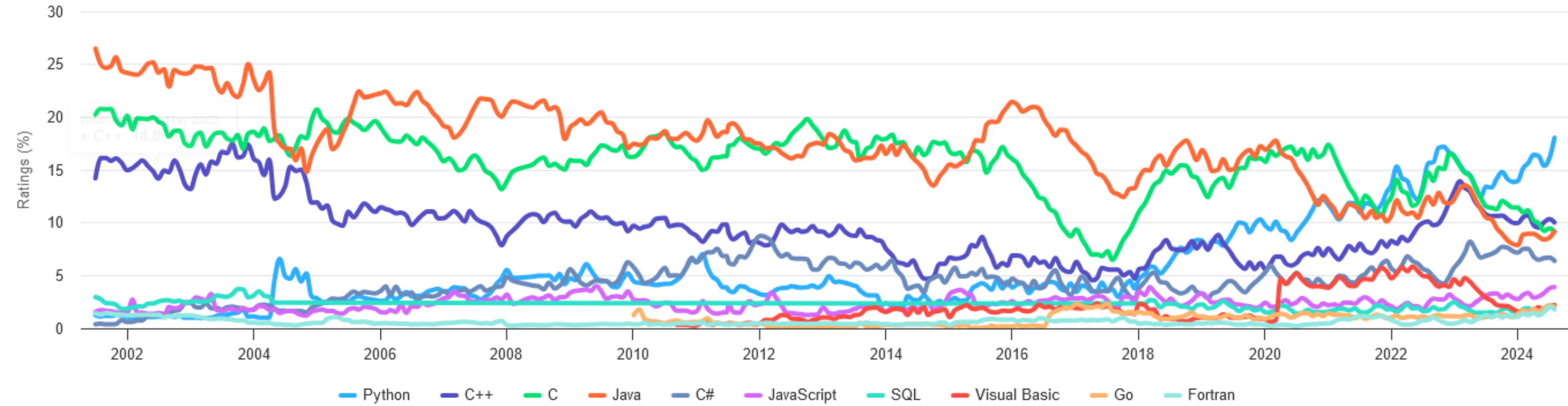
Este tipo de programação tem seu foco na interatividade, e as linguagens utilizadas possuem suporte para a **manipulação de eventos** (sistema-usuário). Geralmente, as linguagens utilizadas dão suporte, também, à **programação orientada a objeto**.











Exemplos: **PHP, Java, Python, Ruby, Javascript**.
Muitos frameworks são utilizados, como **React e Angular (Javascript), Rails (Ruby), Laravel (PHP), Django (Python)**.

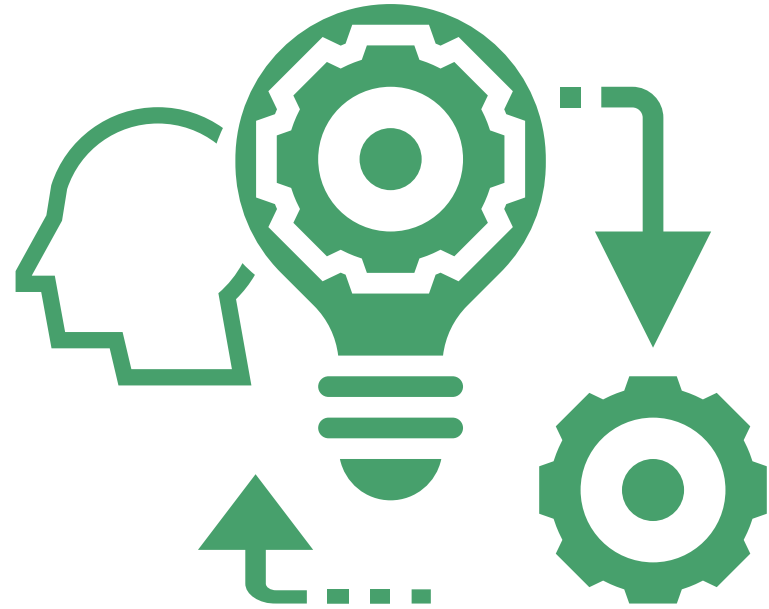


TIOBE Programming Community Index

Source: www.tiobe.com



Aug 2024	Aug 2023	Change	Programming Language		Ratings	Change
1	1			Python	18.04%	+4.71%
2	3	⬆		C++	10.04%	-0.59%
3	2	⬇		C	9.17%	-2.24%
4	4			Java	9.16%	-1.16%
5	5			C#	6.39%	-0.65%
6	6			JavaScript	3.91%	+0.62%
7	8	⬆		SQL	2.21%	+0.68%
8	7	⬇		Visual Basic	2.18%	-0.45%
9	12	⬆		Go	2.03%	+0.87%
10	14	⬆		Fortran	1.79%	+0.75%



Paradigmas de Programação

Qual a Definição?

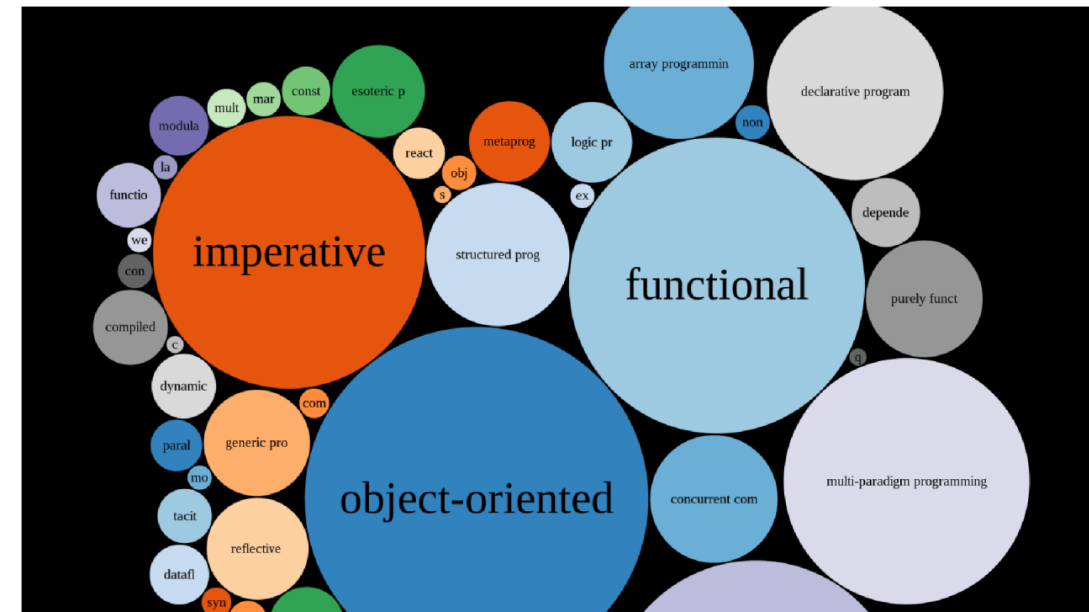
Porque estudar se eu já sei programar?

Toda a linguagem de programação está construída sobre um paradigma. Mas, afinal, o que é um paradigma? Um paradigma representa um padrão de pensamento que guia um conjunto de atividades relacionadas, trata-se de um padrão que define um modelo para a resolução de problemas e regra, basicamente, toda e qualquer linguagem de programação existente. Os paradigmas

- Podemos comparar com as regras de uma linguagem humana, como o Português por exemplo.
- A Língua Portuguesa possui várias regras e padrões que define como devemos utilizar o som ou a escrita para nos comunicarmos.
- **O Paradigma de Programação define regras que devemos usar em uma linguagem de programação para instruir o computador.**
- **O Paradigma também dita as regras que a Linguagem deve seguir.**

Mas porquê estudar os Paradigmas de Programação e não apenas as Linguagens?

- Aumento da capacidade de expressar ideias em código
- Embasamento para escolher linguagens mais adequadas
- Aumento da habilidade de aprender novas linguagens
- Melhor utilização das linguagens já conhecidas



Aumento da capacidade de expressar ideias em códigos

- Pessoas com pouco entendimento da linguagem natural (português, inglês, etc.) **são limitadas também na complexidade de seus pensamentos, especialmente na sua capacidade de abstração e expressão de ideias.**
- Na programação, **conhecer uma maior variedade de recursos e construções da linguagem reduz as limitações no desenvolvimento, e leva a um código bem escrito.**

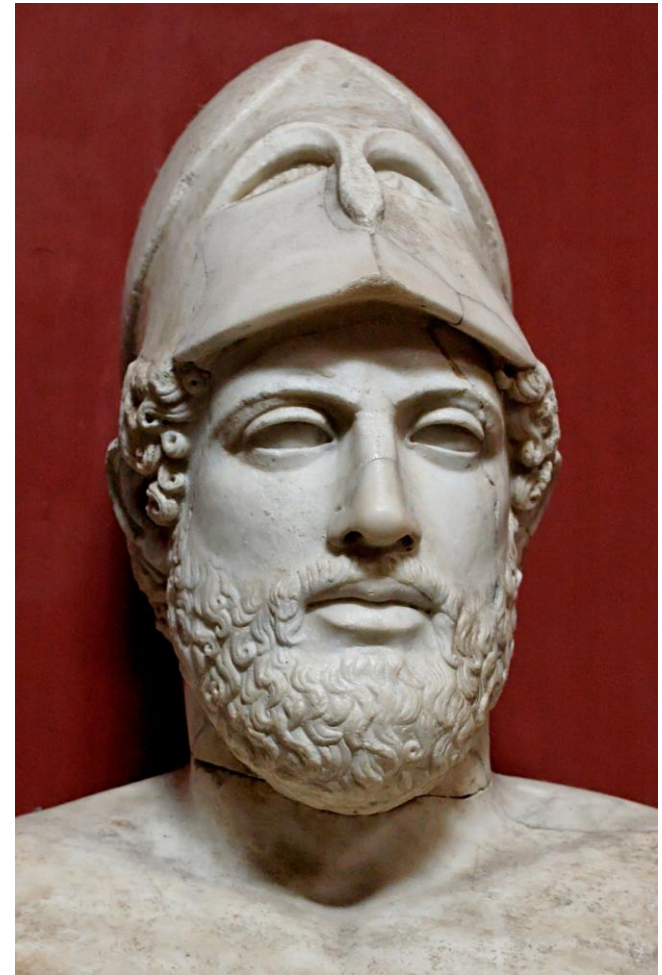
“Vocabulário” mais rico = melhor expressão de “ideias”.

Ha
lac
exp
be
any



t
ing

PERICLES



**“Ter conhecimento, mas não ter o poder de expressá-lo claramente,
não é melhor do que nunca ter ideia nenhuma.”**



Vamos pensar um pouco

- Tenho uma lista de números inteiros. Quero imprimir a soma destes números. Como faço ???

```
public class SomaComFor {  
    public static void main(String[] args) {  
        List<Integer> numeros = Arrays.asList(1, 2, 3, 4, 5);
```



Vamos pensar um pouco

- Uma das maneiras mais fáceis seria com
 - O laço FOR (for-i ou for-each)

```
public class SomaComFor {  
    public static void main(String[] args) {  
        List<Integer> numeros = Arrays.asList(1, 2, 3, 4, 5);  
  
        int soma = 0;  
        for (Integer numero : numeros) {  
            soma += numero;  
        }  
  
        System.out.println("Soma dos números é " + soma);  
    }  
}
```

Vamos pensar um pouco

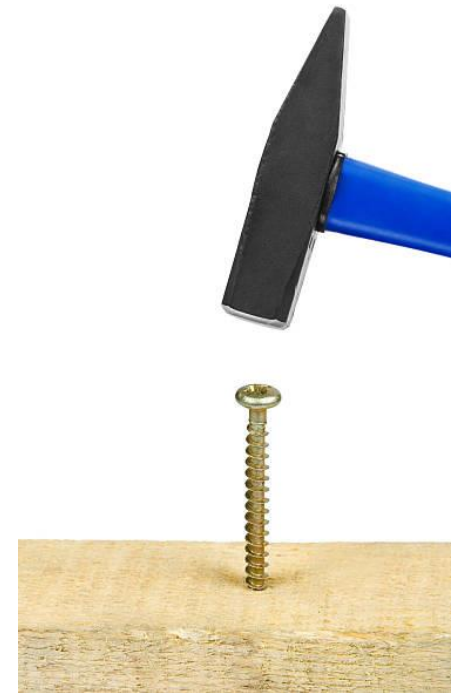


- Mas se você conhece um pouco do Paradigma Funcional, saberá que pode melhorar esse código com funções Lambdas

```
public class SomaComReduce {  
    public static void main(String[] args) {  
        List<Integer> numeros = Arrays.asList(1, 2, 3, 4, 5);  
        System.out.println("Soma dos números é " +  
            numeros.stream().reduce(0, (a, b) -> a + b));  
    }  
}
```

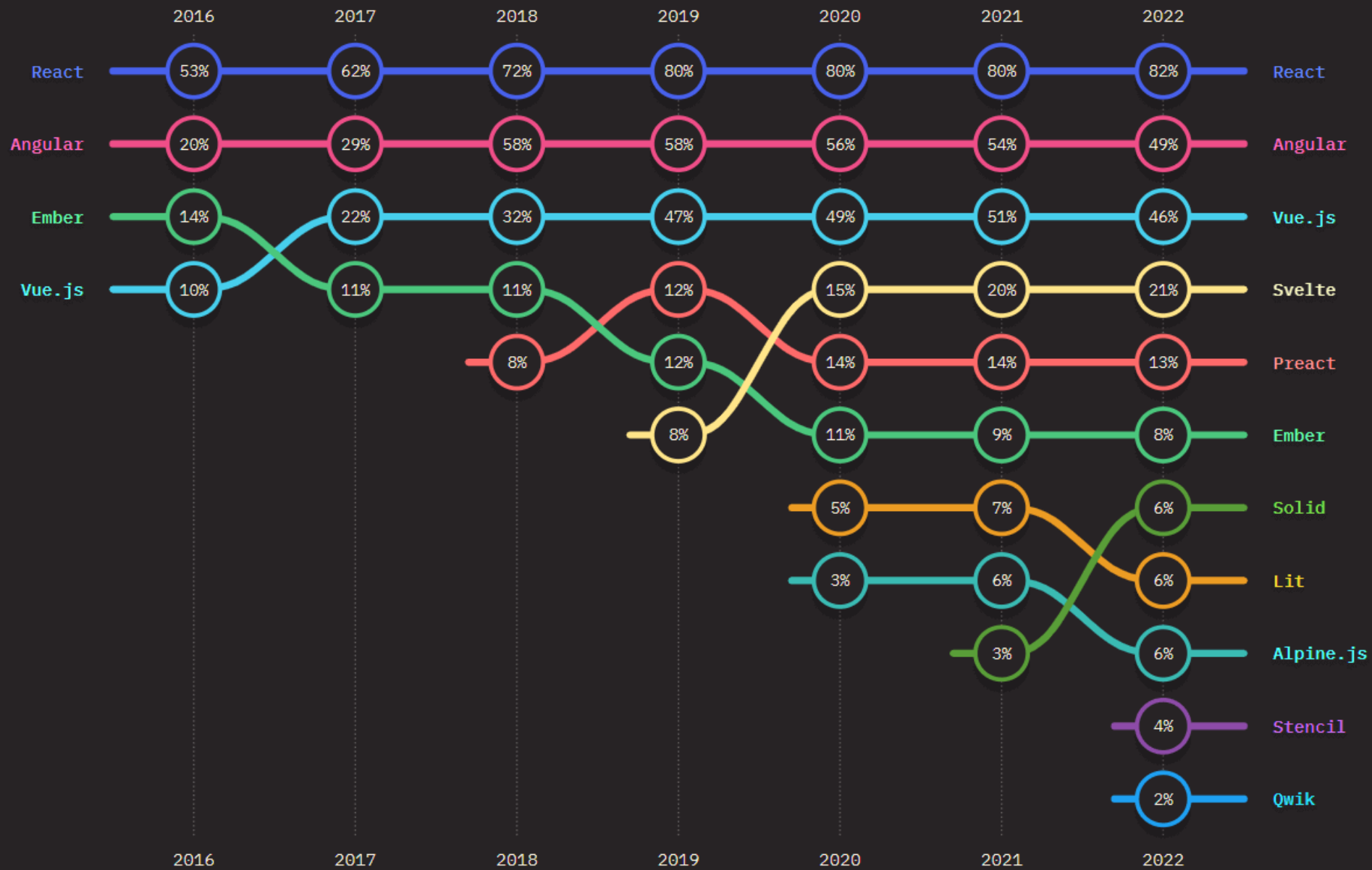
Embasamento para escolher linguagens adequadas

- Geralmente, um programador sem formação mais profunda, tende a escolher a sua **linguagem de preferência** para todo e qualquer projeto em que participa, mesmo que ela **não seja a escolha mais adequada**.
- Ao conhecer uma **faixa mais ampla de paradigmas e linguagens**, o desenvolvedor torna-se capacitado a escolher a linguagem que mais se adequa ao domínio do negócio, mesmo que não seja a que lhe é mais familiar.

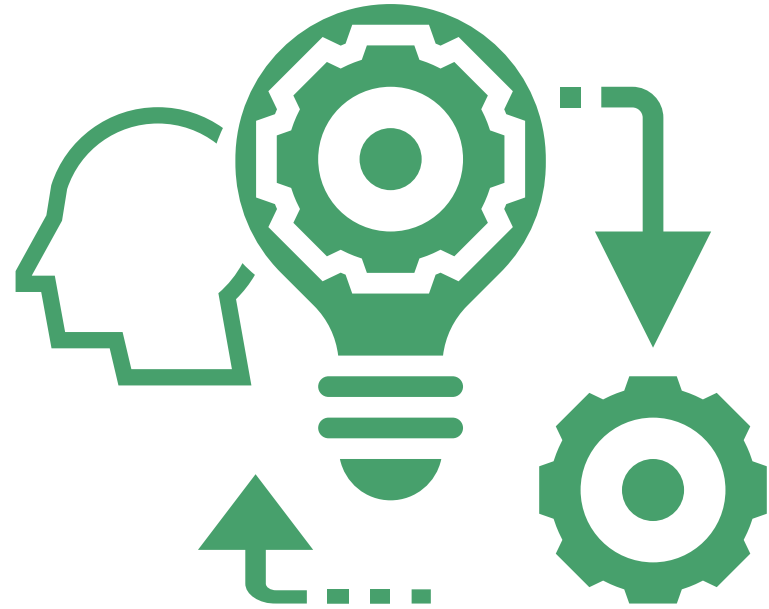


Aumento da habilidade para aprender novas linguagens e/ou novos frameworks

- Nossa área é uma das que mais exige um **aperfeiçoamento contínuo**.
- Constantemente surgem novas bibliotecas, frameworks ou até mesmo linguagens.
- Conhecer bem os conceitos fundamentais das Linguagens de Programação, facilita o processo de aprendizado.
 - Exemplo: Você conhece bem os conceitos de POO, então é mais fácil aprender Java, C# ou C++
- O mesmo ocorre com linguagens naturais: Uma pessoa que entende bem a gramática do Alemão, por exemplo, terá mais facilidade para aprender o Inglês



Retention Interest Usage Awareness



Paradigmas de Programação

Como se classificam?

Paradigmas de programação

```
graph TD; A[Paradigmas de programação] --> B[imperativo]; A --> C[declarativo]; B --> D[programação orientada a objetos]; B --> E[programação procedural]; B --> F[...]; C --> G[programação funcional]; C --> H[programação lógica]; C --> I[...];
```

imperativo

→ programação
orientada a objetos

→ programação
procedural


→ ...

declarativo

→ programação
funcional

→ programação
lógica

→ ...

Aspecto	Paradigma Imperativo	Paradigma Declarativo
Definição	Foca em <u>como fazer algo</u> , detalhando o passo a passo.	Foca em <u>o que deve ser feito</u> , descrevendo o resultado desejado.
Abordagem	Define a sequência de comandos e instruções que o computador deve executar.	Define o resultado desejado sem especificar o fluxo de controle.
Controle de Fluxo	Controle explícito do fluxo de execução (e.g., loops, condicionais).	Controle de fluxo é geralmente implícito, não explicitamente controlado.
Ex. Linguagens 	C, C++, C#, Java, Python, Fortran, Assembly.	SQL, HTML, CSS, Prolog, Haskell.
Estado Mutável	Geralmente usa variáveis e permite modificações no estado.	Evita ou minimiza a modificação de estado; usa variáveis imutáveis.
Popularidade	Mais conhecido	Menos conhecido
Legibilidade	Pode ser menos legível se o código não for bem estruturado; mais explícito.	Frequentemente mais legível e conciso; menos explícito.

Multiparadigmas



- As linguagens de programação frequentemente têm um paradigma "principal" ou predominante, que orienta seu design e uso.
- Esse paradigma é geralmente o mais enfatizado e utilizado na maioria das situações para as quais a linguagem foi projetada.
- No entanto, muitas linguagens modernas são **multi-paradigma**, o que significa que suportam e incentivam o uso de múltiplos estilos de programação, permitindo uma maior flexibilidade e adaptabilidade na solução de problemas.

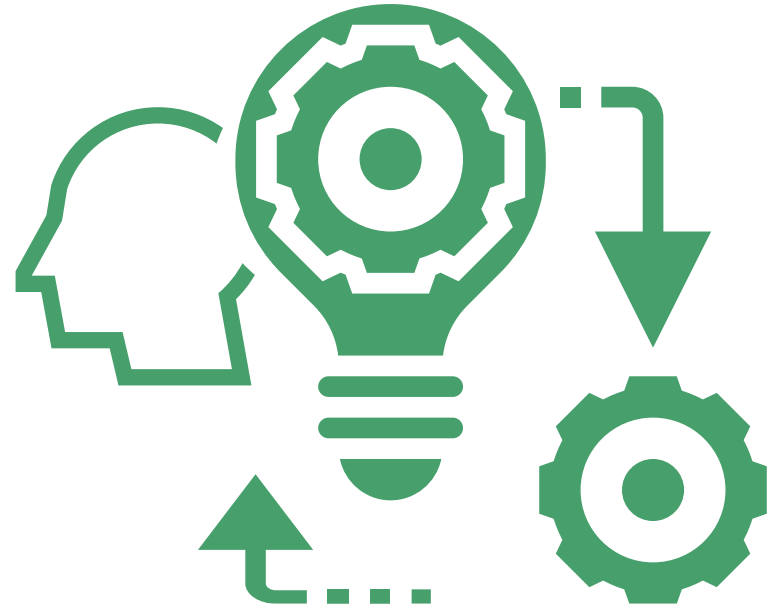
```
public class SomaComFor {  
    public static void main(String[] args) {  
        List<Integer> numeros = Arrays.asList(1, 2, 3, 4, 5);  
  
        int soma = 0;  
        for (Integer numero : numeros) {  
            soma += numero;  
        }  
  
        System.out.println("Soma dos números é " + soma);  
    }  
}
```

```
public class SomaComReduce {  
    public static void main(String[] args) {  
        List<Integer> numeros = Arrays.asList(1, 2, 3, 4, 5);  
        System.out.println("Soma dos números é " +  
            numeros.stream().reduce(0, (a, b) -> a + b));  
    }  
}
```

Imperativo ou Declarativo ???

SQL

```
SELECT id, nome, cpf  
FROM usuarios  
WHERE nome = 'Zezinho';
```



Paradigmas Imperativos

Quais os principais?

Quais as linguagens predominantemente imperativas?

Paradigmas Imperativos

- Apesar das Linguagens de Programação de Alto Nível surgirem na década de 50, o termo “Paradigma de Programação” não era usado.

Uso de goto e Spaghetti Code

- Nos primeiros dias da programação, o *goto* era amplamente utilizado para controlar o fluxo de execução. Sem regras rígidas para controle de fluxo, isso frequentemente resultava em código desordenado e difícil de manter.

Spaghetti Code – Uso demasiado do *goto*

Edgar Dijkstra: Go To Statement Considered Harmful

Go To Statement Considered Harmful

Key Words and Phrases: go to statement, jump instruction, branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing

CR Categories: 4.22, 5.23, 5.24

dynamic progress is only characterized when we also give to which call of the procedure we refer. With the inclusion of procedures we can characterize the progress of the process via a sequence of textual indices, the length of this sequence being equal to the dynamic depth of procedure calling.

Let us now consider repetition clauses (like: while *B* repeat *A*

$un \rightarrow en$), the fact remains that the progress of the process remains characterized by a single textual index.

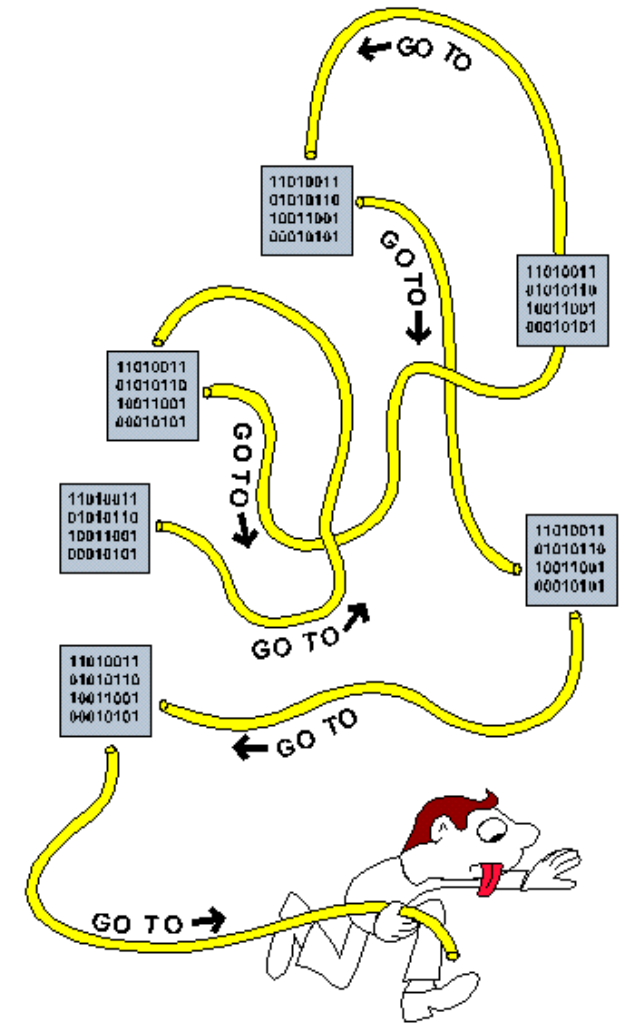
As soon as we include in our language procedures we must admit that a single textual index is no longer sufficient. In the case that a textual index points to the interior of a procedure body the

that they will satisfy all needs, but (e.g. abortion clauses) they should programmer independent coordina describe the process in a helpful a

It is hard to end this with a

Communic

Volume 11 / Number 3 / March, 1968



Java Language Keywords

Here is a list of keywords in the Java programming language. You cannot use any of the following as identifiers in your programs. The keywords cannot be used as identifiers in your programs.

abstract

assert^{***}

boolean

break

byte

case

catch

char

class

const^{*}

* not used

** added in 1.2

*** added in 1.4

**** added in 5.0

continue

default

do

double

else

enum^{****}

extends

final

finally

float

for

goto^{*}

if

implements

import

instanceof

int

interface

long

native

Paradigma Procedural e Estruturado

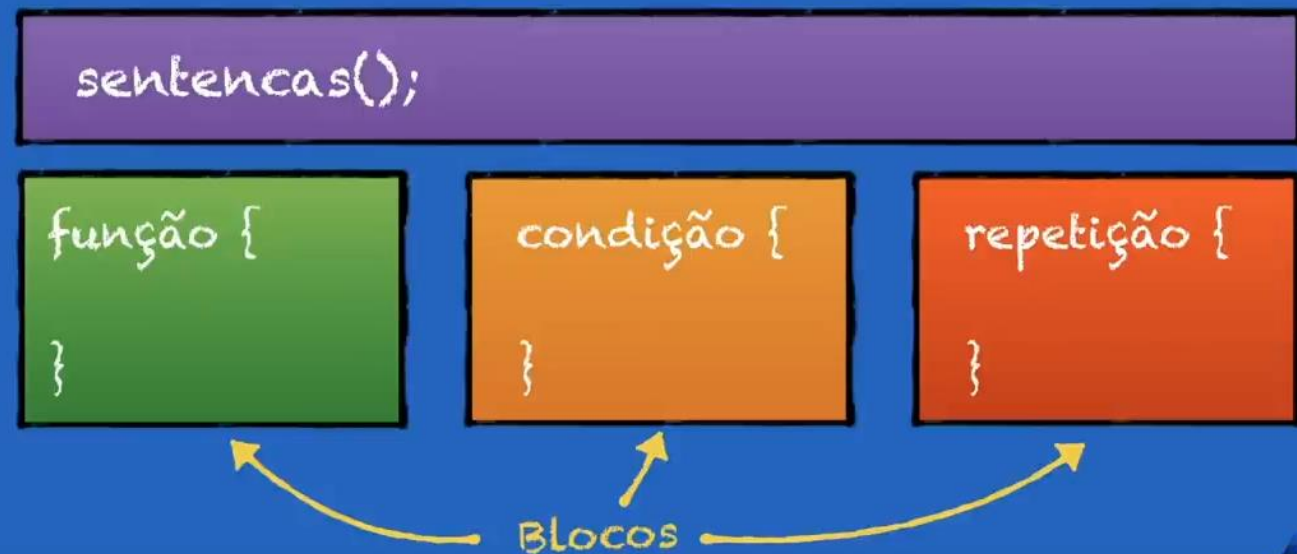
Década 1970

- **Programação Estruturada:** Estabeleceu práticas para evitar o uso excessivo de *goto*, usando estruturas de controle como loops (for, while) e condicionais (if, else) para criar um fluxo de controle mais claro e organizado.
 - **Muitos autores consideram a programação estruturada como um subconjunto do Paradigma Procedural**
- **Programação Procedural:** A programação procedural organiza o código em procedimentos ou funções, encapsulando a lógica em blocos modulares. Isso ajuda a criar um código mais legível e estruturado, minimizando a necessidade de saltos desordenados e promovendo uma abordagem modular e reutilizável.

Paradigma Procedural: Escopos



Paradigma Procedural



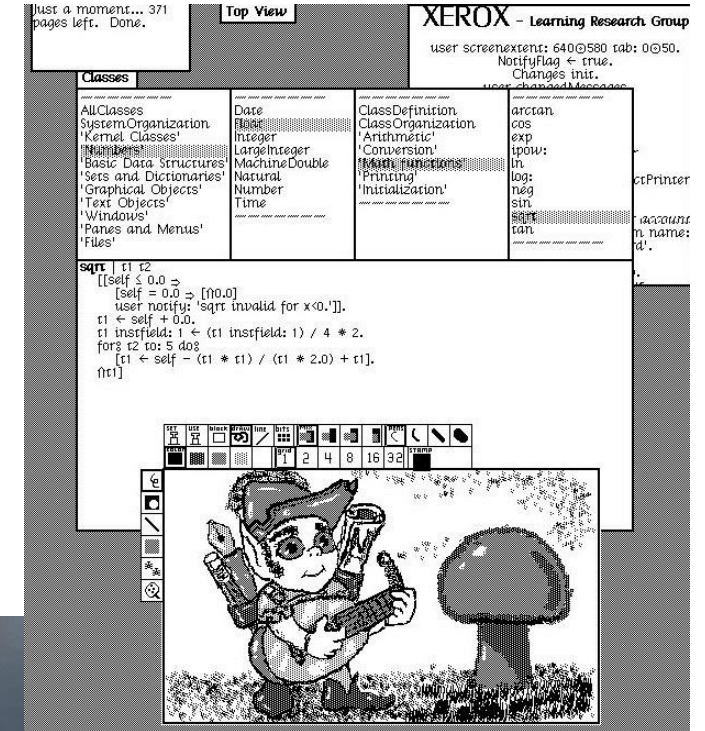
Paradigma Procedural

Linguagens de Programação

- Fortran – 1957
- COBOL – 1959
- ALGOL – 1960
- CPL (Combined Programming Language) – 1960-1963
- BCPL (Basic Combined Programming Language) – 1966
- B – 1969
- C – 1972
- Pascal – 1970

Paradigma Orientado a Objetos

- Linguagem: Smalltalk-80
 - Introduziu o conceito de IDE
- Criador: Alan Kay
- Linguagem Simula implementou os primeiros conceitos de Classes



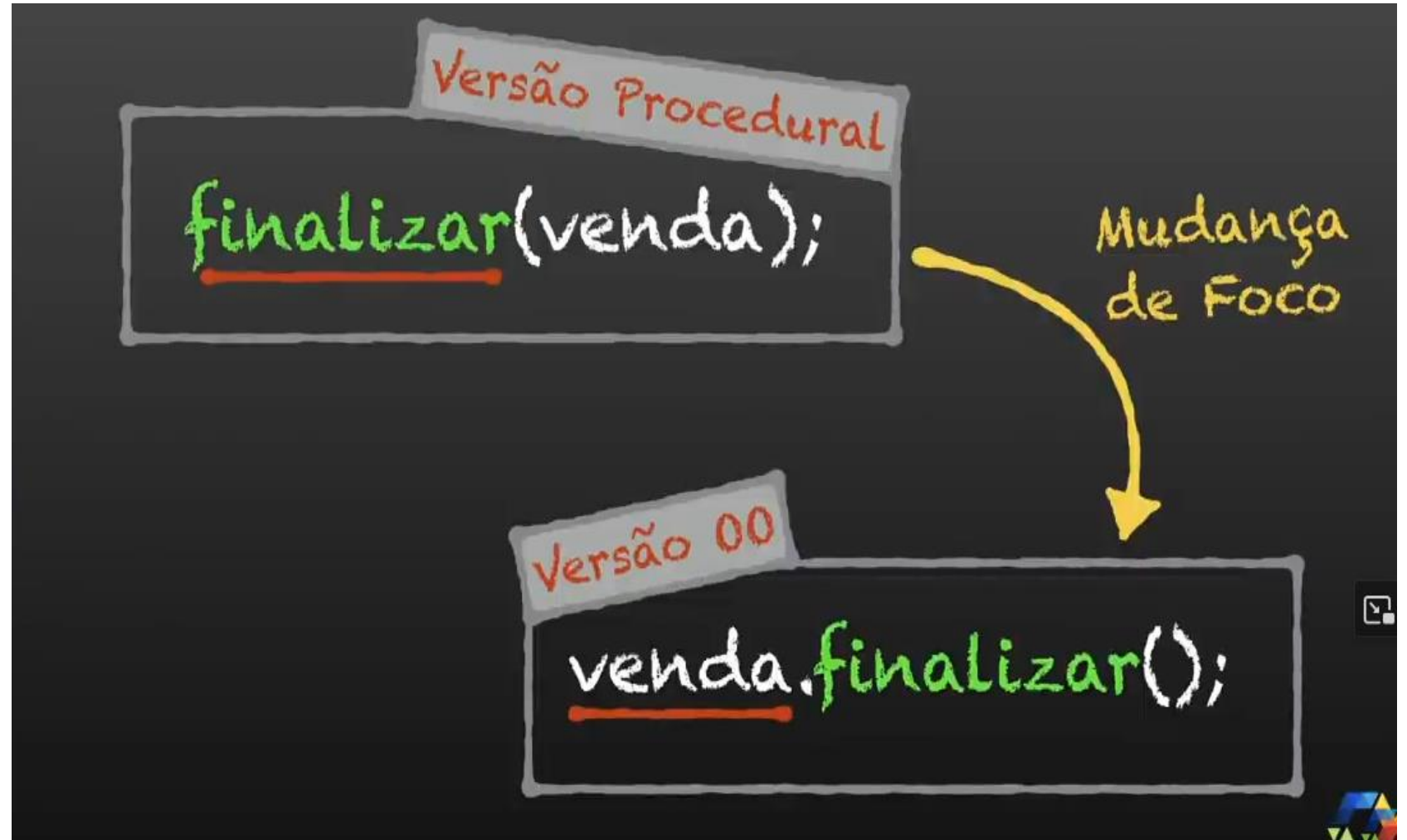
1950 – 1960 Era do Caos	1970 – 1980 Era da Estruturação	1990 até agora Era dos Objetos
Saltos, gotos, variáveis não estruturadas, variáveis espalhadas ao longo do programa	If-then-else Blocos Registros Laços-While	Objetos Mensagens Métodos Herança



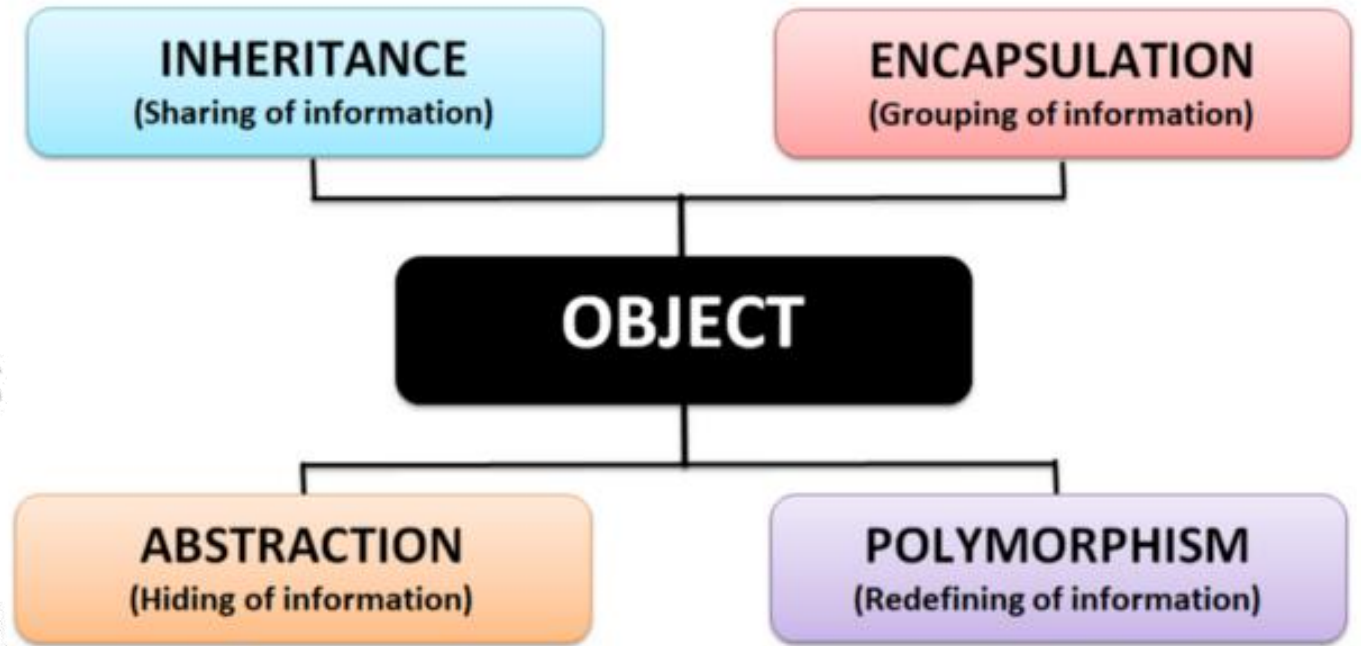
Ciência da
Computação

Procedural → OO

Foco na Ação → Foco nos Dados (objetos)



The Four Pillars

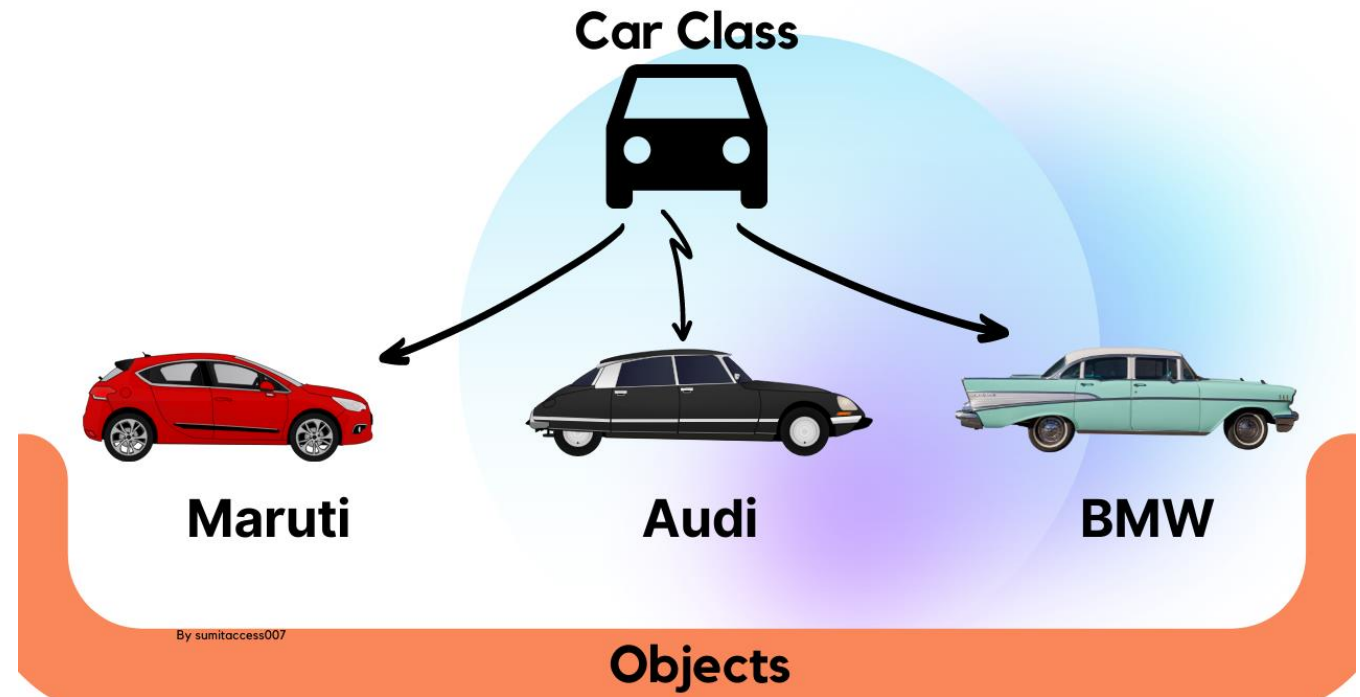
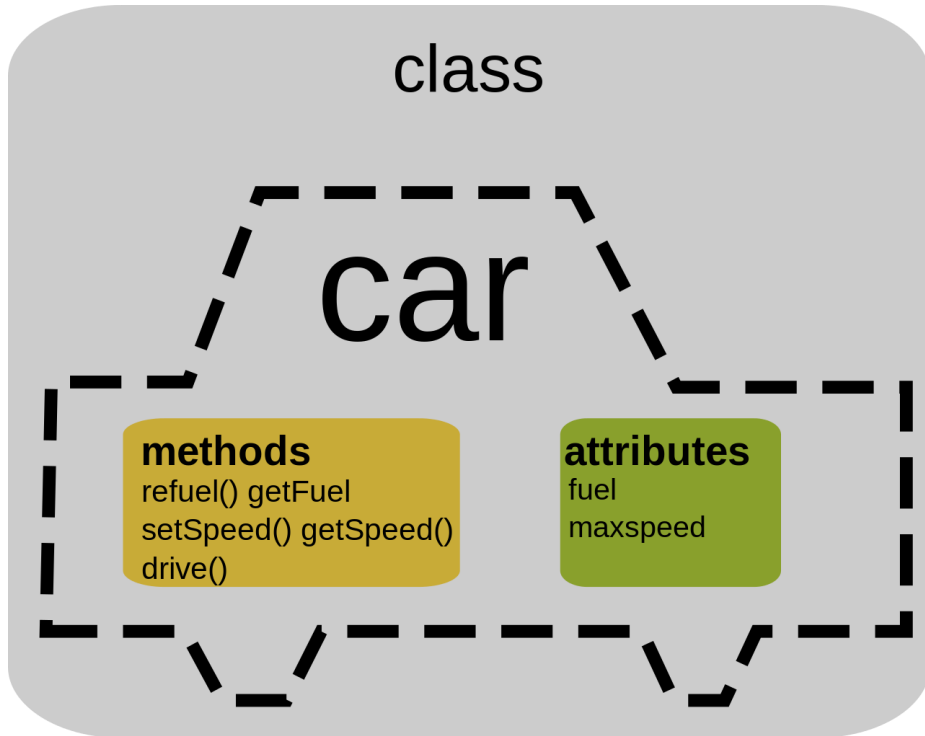


Abstração

- **Abstração** é o processo de identificar e definir características essenciais de objetos do mundo real, representando essas características em forma de **classes** e **objetos**.
- Uma **classe** é uma estrutura que define atributos e métodos representando um tipo de objeto, fornecendo um modelo para criar instâncias desse tipo.
- Um **objeto** é uma instância de uma classe, caracterizado por seus atributos e comportamentos definidos na classe.
 - Ele pode interagir com outros objetos por meio de métodos e troca de dados.

Objeto é uma **instância** concreta de uma classe na POO.

Abstração



Encapsulamento

- **Encapsulamento** é a prática de ocultar os detalhes internos de um objeto, expondo apenas as operações relevantes para manipular esses detalhes. Isso promove a modularidade, a segurança e a manutenibilidade do código.
- Na maioria das linguagens de programação orientada a objetos, o encapsulamento é alcançado através da definição de **modificadores de acesso**.
 - **public, private, protected e default** (sem modificador)

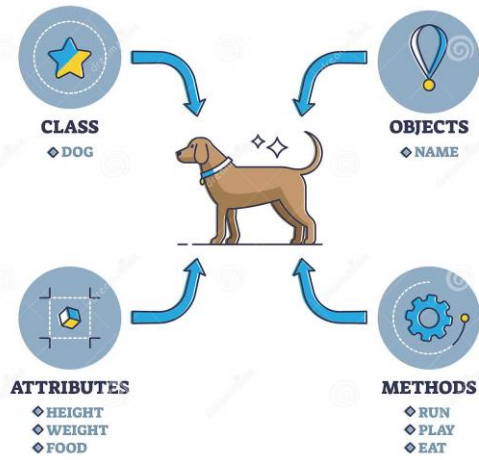
Herança

- **Herança** é o princípio que permite que uma classe (subclasse) herde os atributos e métodos de outra classe (superclasse), facilitando a reutilização de código, a organização hierárquica e a promoção de relações entre objetos.
- Quando há duas ou mais classes com atributos e métodos em comum, a herança facilita a criação de uma estrutura hierárquica. Nessa estrutura, uma classe "pai" define os elementos comuns que serão herdados pelas classes "filhas".

Polimorfismo

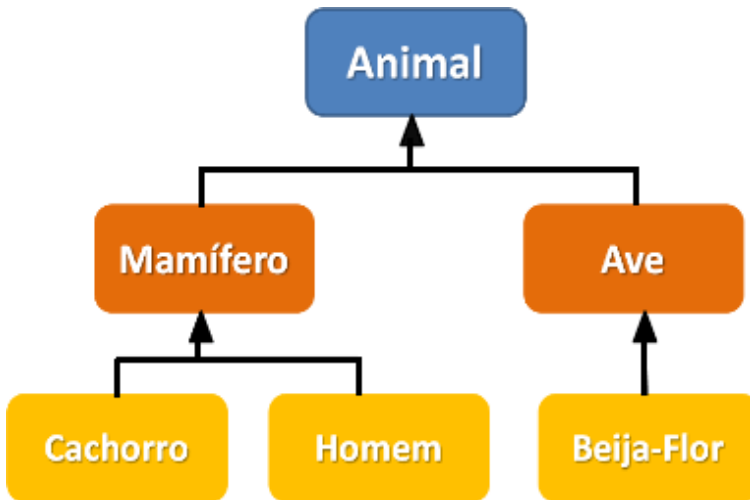
- **Polimorfismo** é o princípio que permite que objetos instanciados a partir de sub-classes de uma mesma classe base possam invocar métodos que, apesar de terem a mesma assinatura, comportam-se de maneira diferente dependendo do tipo específicos do objeto.
- Com o Polimorfismo, os mesmos atributos e objetos podem ser utilizados em objetos distintos, porém, com implementações lógicas diferentes.

OBJECT ORIENTED PROGRAMMING



dreamstime.com

ID 239724045 © VectorMine



arbabwaseer@gmail.com

Entregáveis

Vamos lá. Está na hora!!!



