



# Ciência da **Computação**

Programação Orientada a Objetos  
Prof. Luciano Rodrigo Ferretto



2025/02 - Parada. Ling. Prog

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
- Entregáveis

# Apresentação e Plano de Aprendizagem - AVA

<https://atitus.grupoa.education/plataforma/course/3365342/content/50033752>

Em nosso AVA temos a seção Apresentação e Plano de Aprendizagem, no qual consta os contatos que vocês podem estar utilizando para falar com o professor, assim como o link para o acesso ao nosso Plano de Aprendizagem.

# Grupo do WhatsApp - AVA

<https://atitus.grupoa.education/plataforma/course/3365342/content/50036084>

Em nosso AVA também temos a seção Grupo do WhatsApp, no qual consta o link para o grupo que o professor criou com o objetivo de facilitar e agilizar os avisos da disciplina.

# 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.

The image shows a Google Form titled "Feedback das Aulas" (Classroom Feedback). The form is for the course "CC 2025.2 - Paradigma de Linguagem de Programação". It includes a header with the course name and a sub-header "Feedback Constante das aulas de". Below this, there is a note: "Não é necessário estar logado com uma conta google para responder esse formulário. Assim, sua identidade é mantida em sigilo." (It is not necessary to be logged in with a Google account to answer this form. Thus, your identity is kept confidential). The form asks for a rating from 1 to 5 for the classes. There are two buttons: "Avançar" (Next) and "Limpar formulário" (Clear form). At the bottom, it says "Google Formulários" and "Este formulário foi criado em Atitus Educação." (This form was created in Atitus Education).

Feedback das Aulas

Feedback Constante das aulas de  
CC 2025.2 - Paradigma de  
Linguagem de Programação

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 \*

Seu e-mail

Selecione o Professor: \*

Escolher

Avançar

Limpar formulário

Google Formulários Este formulário foi criado em Atitus Educação.

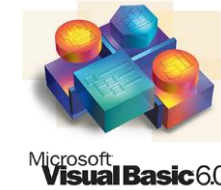
# Entregáveis

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 2000 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







# 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



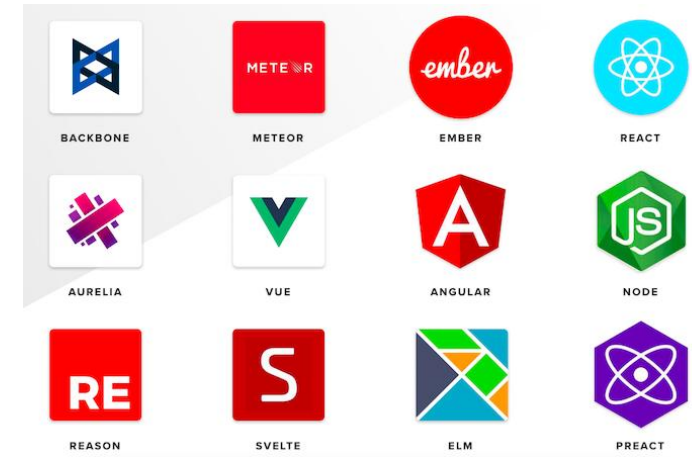
- Objetivo do Software = Resolução de problemas



- 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!

# Exemplo: Quero um contador simples em React

```
1  import { useState, useEffect } from "react";
2
3  export default function Counter() {
4    const [count, setCount] = useState(0);
5
6    useEffect(() => {
7      console.log(`O contador agora vale: ${count}`);
8    }, [count]);
9
10   return (
11     <div className="flex flex-col items-center gap-4 p-4 border rounded-xl shadow-md">
12       <h1 className="text-xl font-bold">Contador: {count}</h1>
13       <button
14         onClick={() => setCount(count + 1)}
15         className="px-4 py-2 bg-blue-500 text-white rounded-lg hover:bg-blue-600"
16       >
17         Incrementar
18       </button>
19     </div>
20   );
21 }
```

**Contador: 3**

Incrementar

Console

Executar executando o código ...

'O contador agora vale: 0'

module:16

'O contador agora vale: 1'

module:16

'O contador agora vale: 2'

module:16

'O contador agora vale: 3'

module:16

# Alguns conceitos importantes

- **Gerenciamento de estado (useState):** Qualquer aplicação precisa lidar com mudanças de valores ao longo do tempo.
- **Efeito colateral (useEffect):** Quando um valor muda, pode ser necessário executar alguma ação (como registrar logs ou buscar dados externos).
- **Reatividade:** O React reage a mudanças no estado e re-renderiza o componente automaticamente.



Então ...



O React pode mudar no futuro, mas a necessidade de gerenciar estado e efeitos continuará existindo em qualquer tecnologia de frontend.



Ainda não te convenci???

Então vamos a mais um exemplo

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...



# No mundo real também vale essa premissa.

- Depois, de muito tempo sonhando, consegui uma Dodge Journey
  - Motor Pentastar V6 – 3,6cc – 280cv
- 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 ???

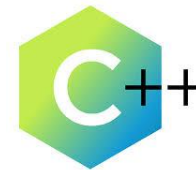
**É CLARO, e isso vale para a área de TI.**

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



# Quer mais exemplo: Conceitos de Orientação a Objetos

- Abstração
  - Encapsulamento
  - Herança
  - Polimorfismo
- 
- **Conceitos iguais para todas as linguagens OO**





# Porém....

- Sempre tem um porém....



# Porém, não vamos exagerar também...

## O poliglota iletrado

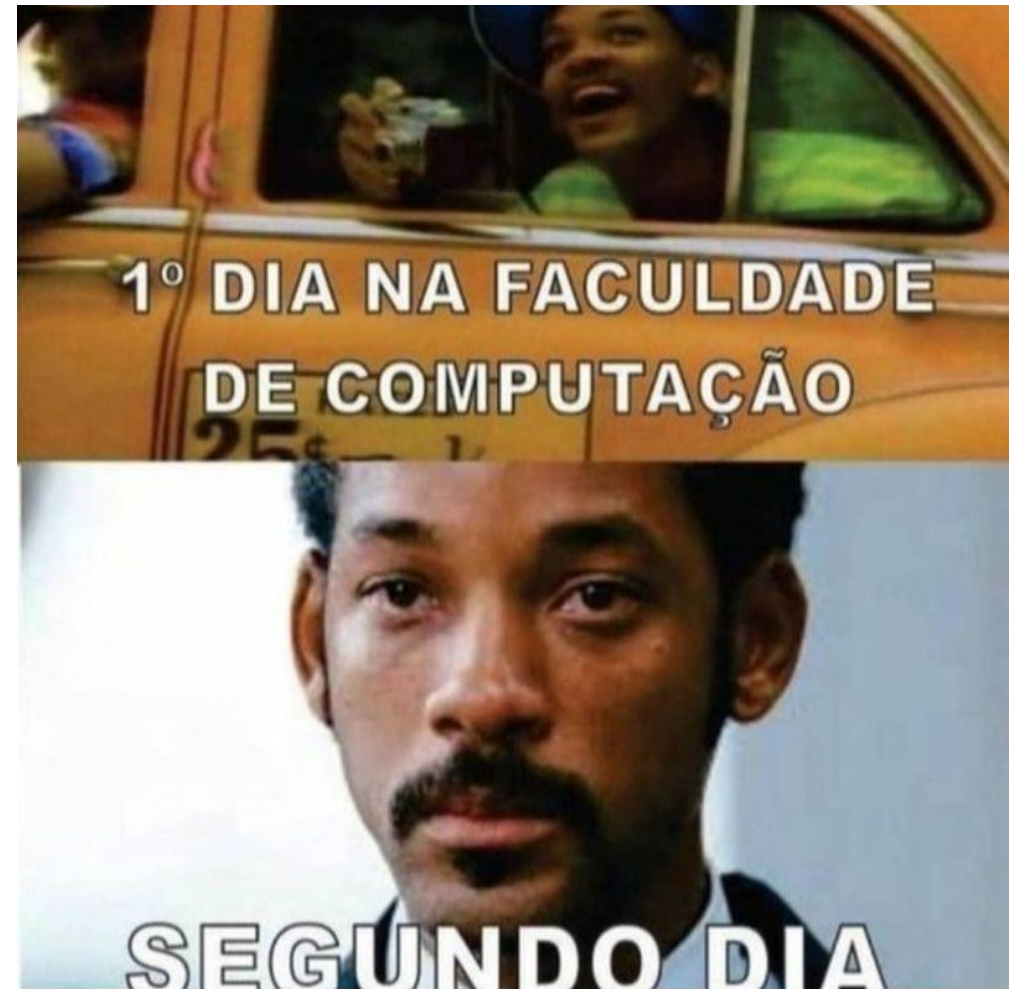
**Fluente** em diversos idiomas. **Desconhece assuntos de linguística**. Incapaz de generalizar e abstrair a partir de estruturas linguísticas presentes nos idiomas que ele domina e, portanto, para ele, é sempre igualmente **penoso aprender um novo idioma**. Tudo é sempre novidade.

## 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 ???

"E se eu deixar o curso e montar um bar ao lado da faculdade?"



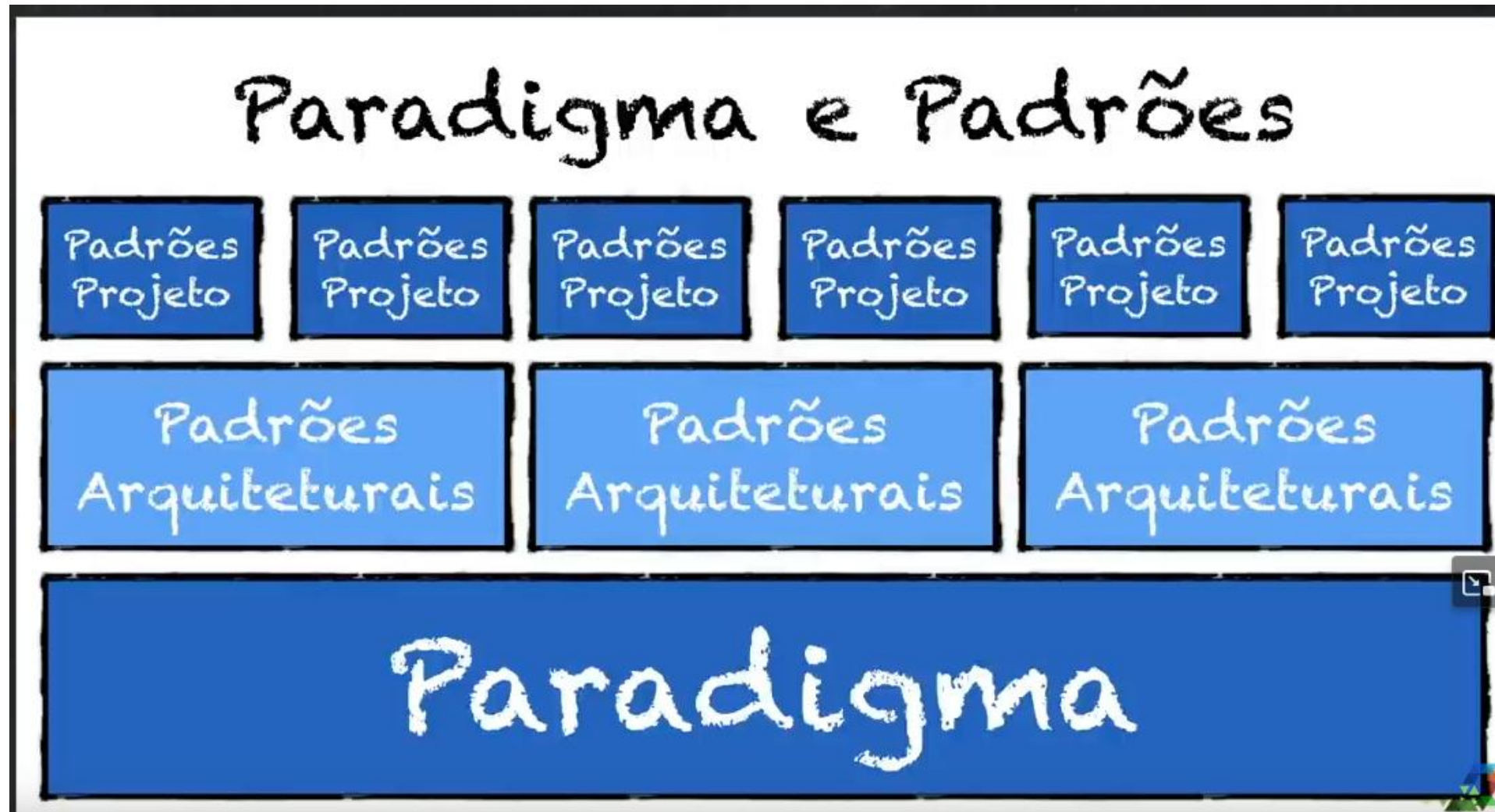
<https://www.youtube.com/watch?v=nlvtKEL8JzA>

**Tem o teu**

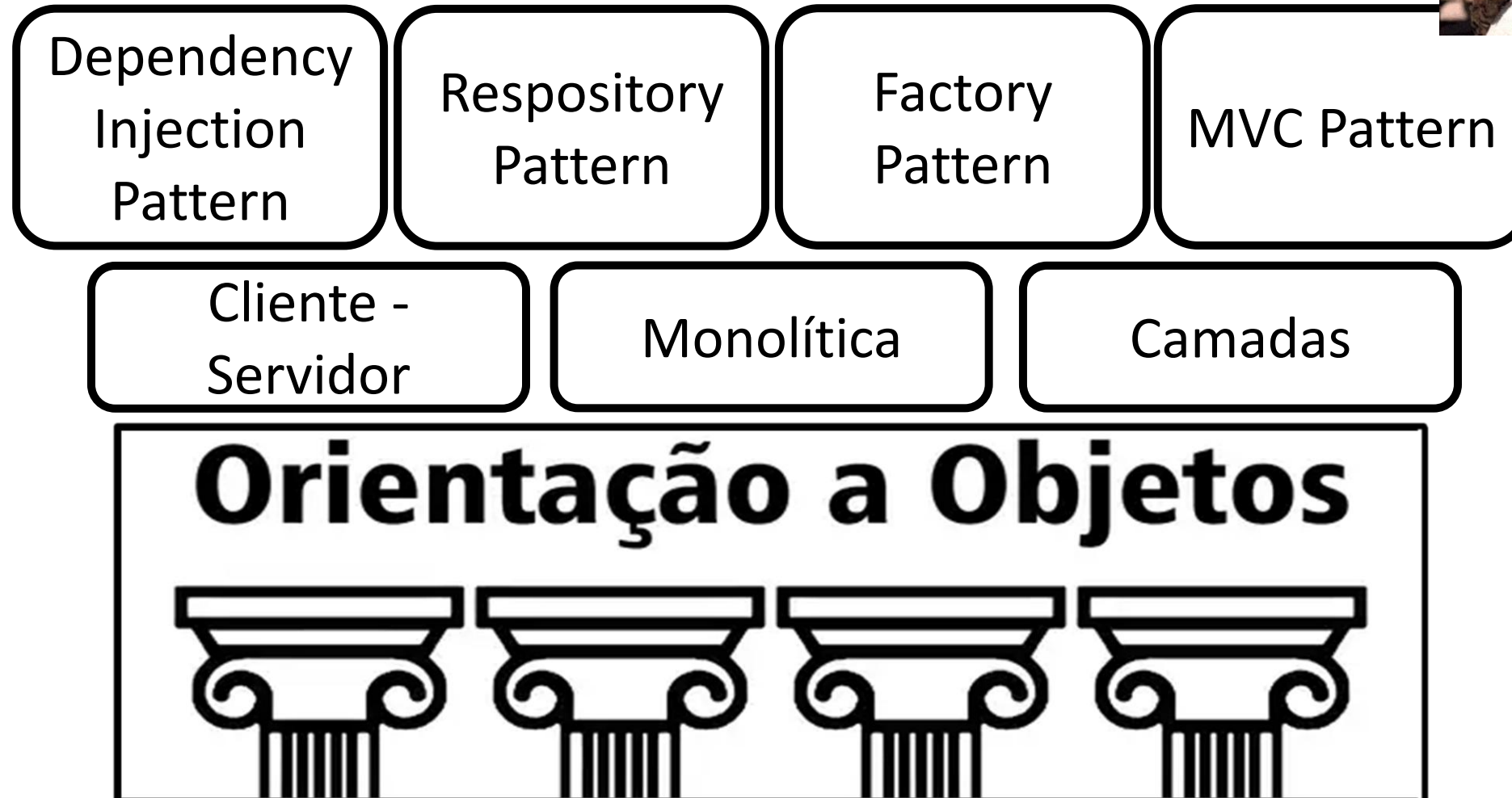
Agora sim! Vamos falar do nosso semestre!

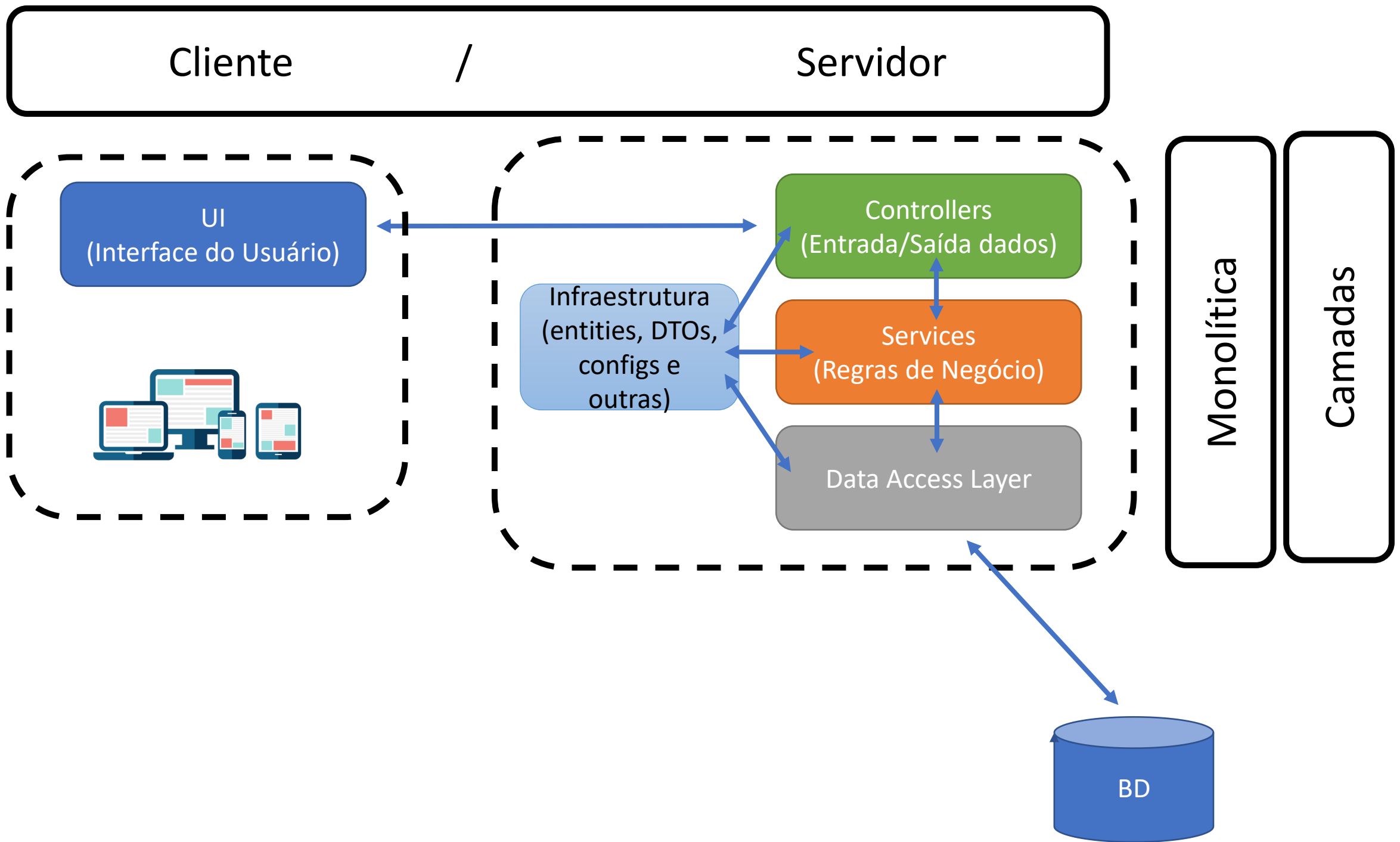


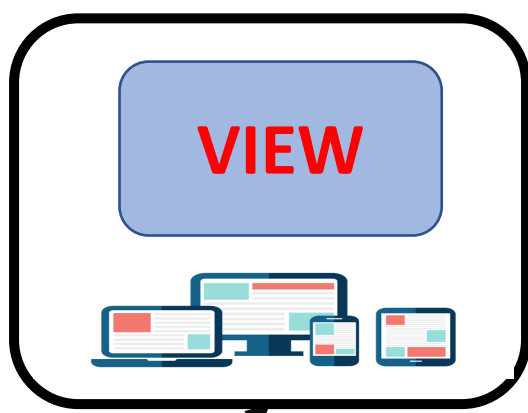
O que iremos estudar?



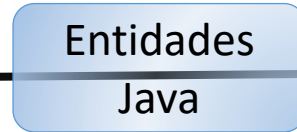
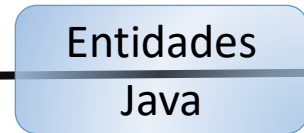
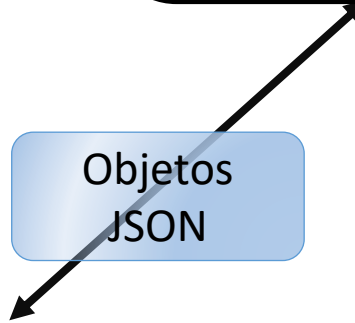
Sabia que vocês já utilizaram esses conceitos no semestre passado ????



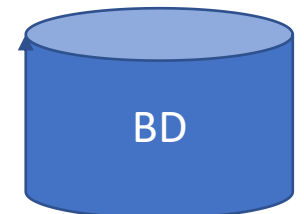
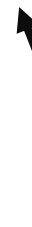
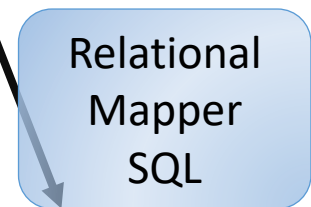




MVC Pattern  
Model-View-Controller

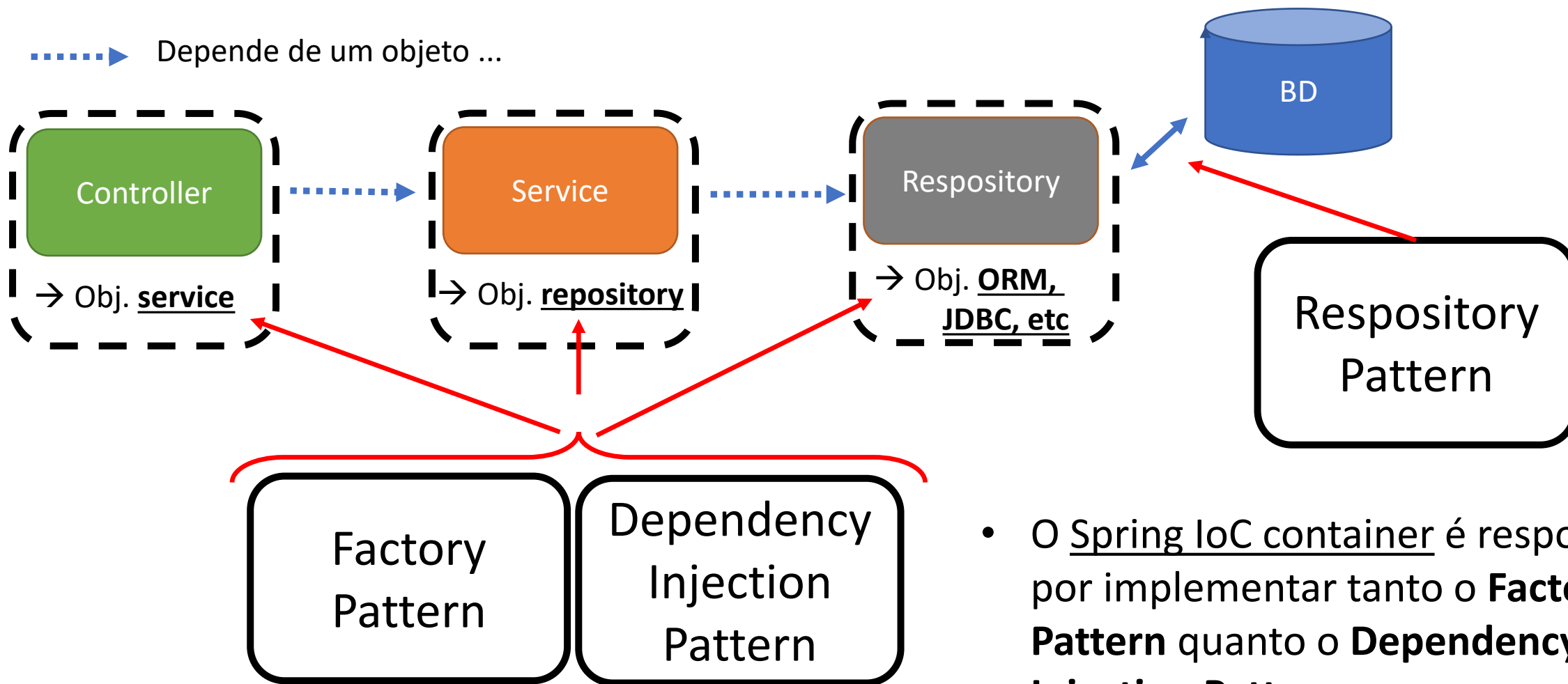


MODEL





.....> 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>{
```



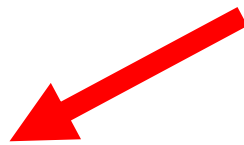
- Com a anotação [@Service](#), indicamos que o Spring IoC container deve ser responsável por criar e gerenciar o ciclo de vida dos objetos dessa classe, implementando o **Factory Pattern** e tirando a responsabilidade do desenvolvedor.
- Dessa forma, o container instanciará os beans conforme necessário 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

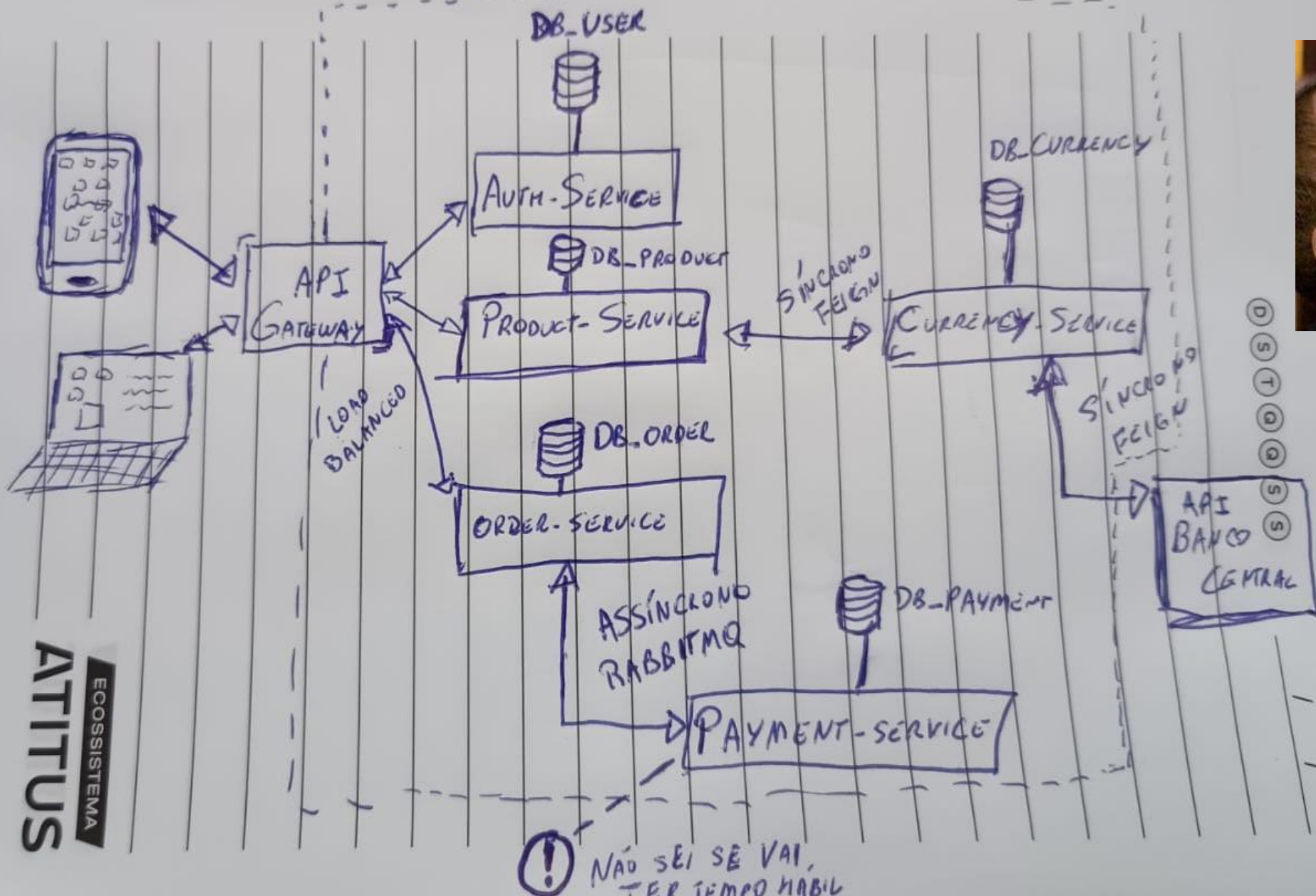
## **3 - Microservices**

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

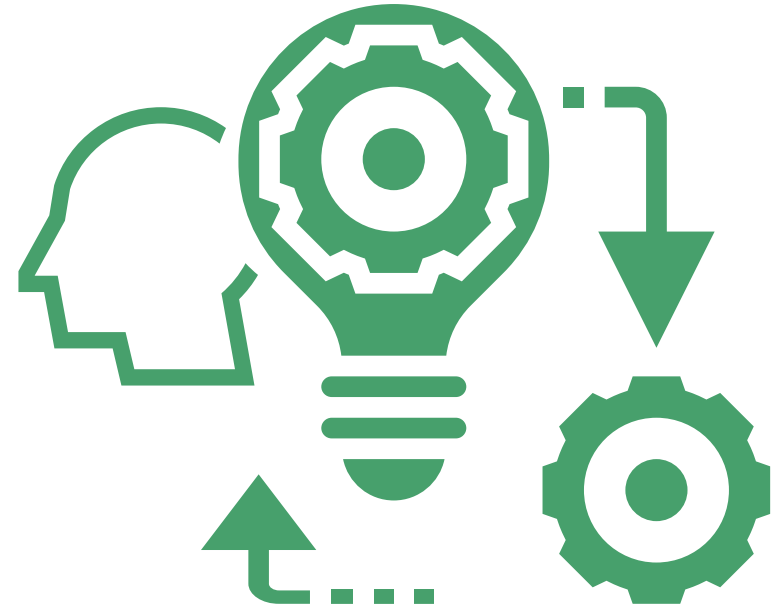
# Atividade do Semestre – Grupos

- Cada grupo deverá adaptar o código base trabalhado em aula para construir o backend da sua aplicação, conforme o tema definido.
  - A solução final deve ser **funcional, alinhada com a proposta do grupo e seguir os princípios, boas práticas e padrões arquiteturais abordados durante o semestre.**
- ◆ *Service Registry Pattern*
  - ◆ *Self Registration Pattern*
  - ◆ *Externalized Configuration Pattern*
  - ◆ *API Gateway Pattern*
  - ◆ *Server-Side Discovery Pattern*
  - ◆ *Authentication Gateway Filter*
  - ◆ *Integration with External Service*
  - ◆ *Resilience Patterns*
  - ◆ *Client-Side Discovery Pattern*
  - ◆ *Secure Resource Access*
  - ◆ *Caching & Fault Tolerance*
  - ◆ *Service Composition*
  - ◆ *Client-Side Discovery Pattern*
  - ◆ *Service per Container Pattern*
  - ◆ *Database per Service Pattern*
  - ◆ *Self Registration Pattern*









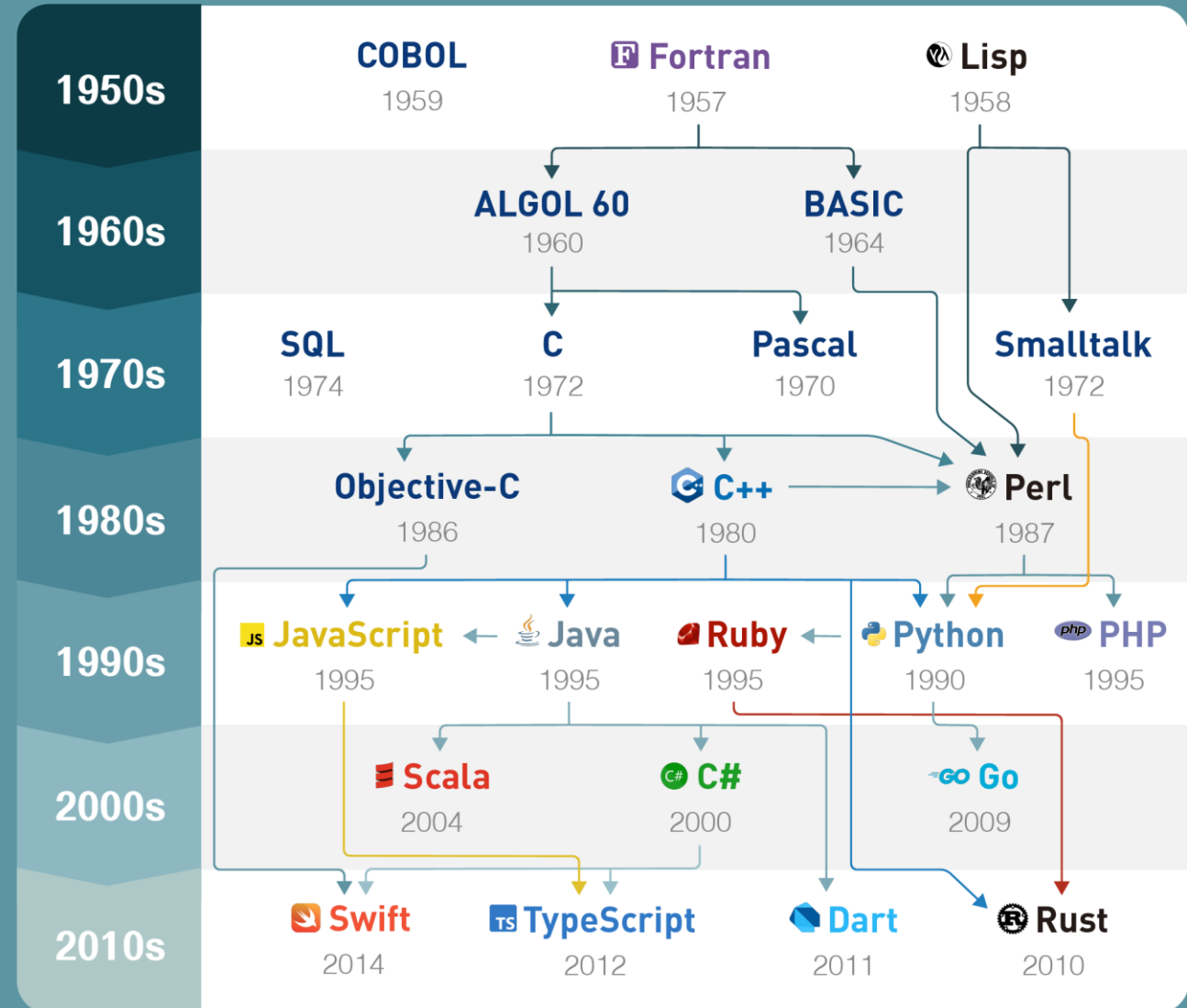
# Paradigmas de Programação

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.

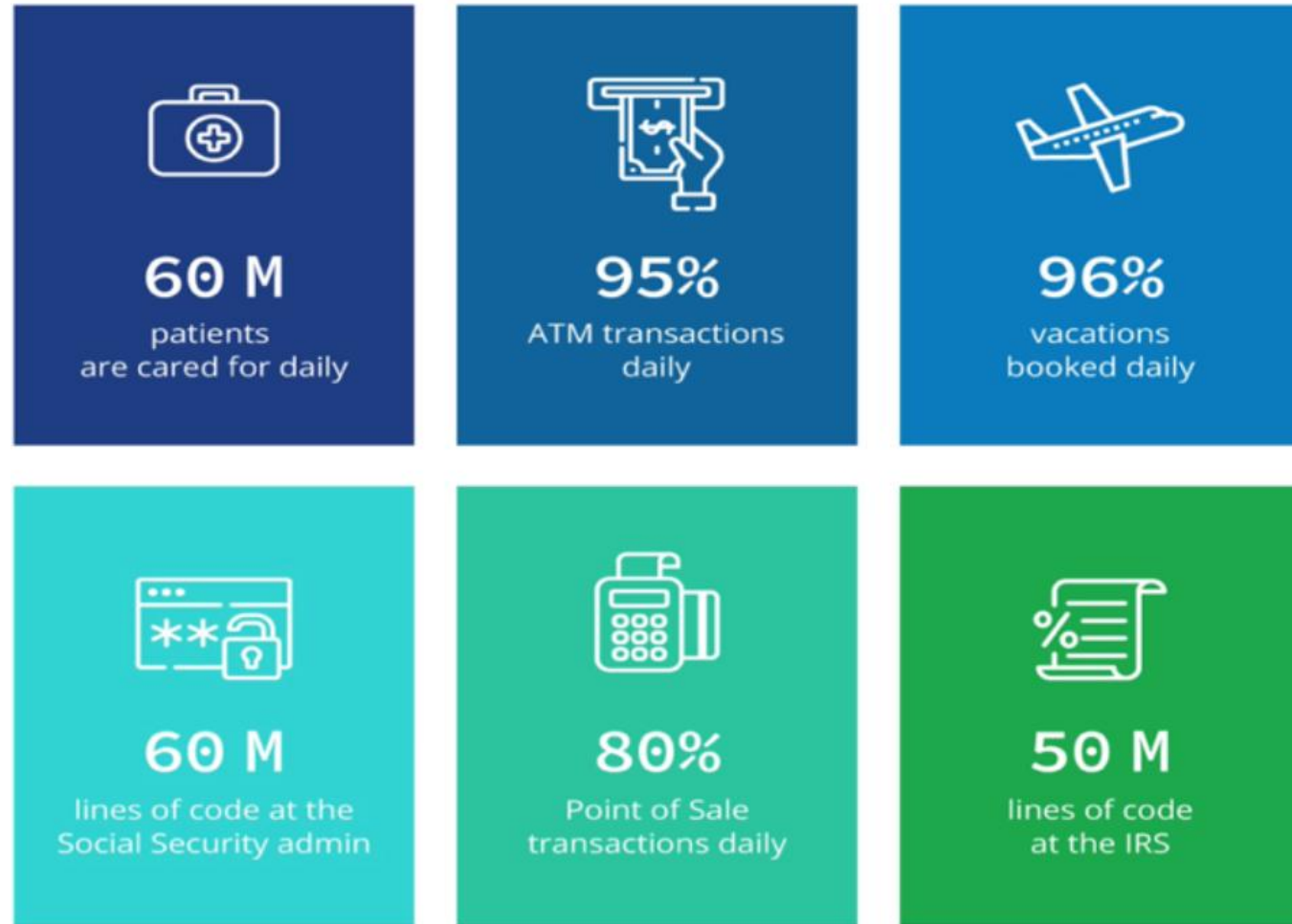
A linguagem mais utilizada, tradicionalmente, é o **COBOL** (*Common Business Oriented Language*), que usa a língua inglesa como base de sua sintaxe e suporta o estilo de programação **imperativo**.

- 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**.

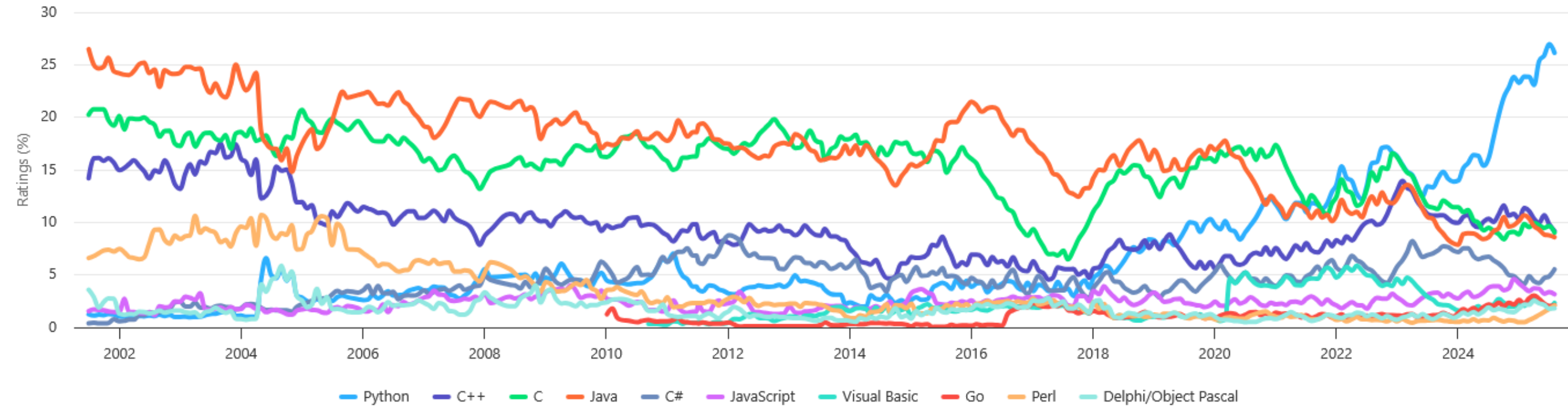
Este tipo de programação tem seu foco na interatividade, e as linguagens utilizadas possuem suporte para a **manipulação de eventos** (sistema-usuário). Geralmente, as linguagens utilizadas dão suporte, também, à **programação orientada a objeto**.

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














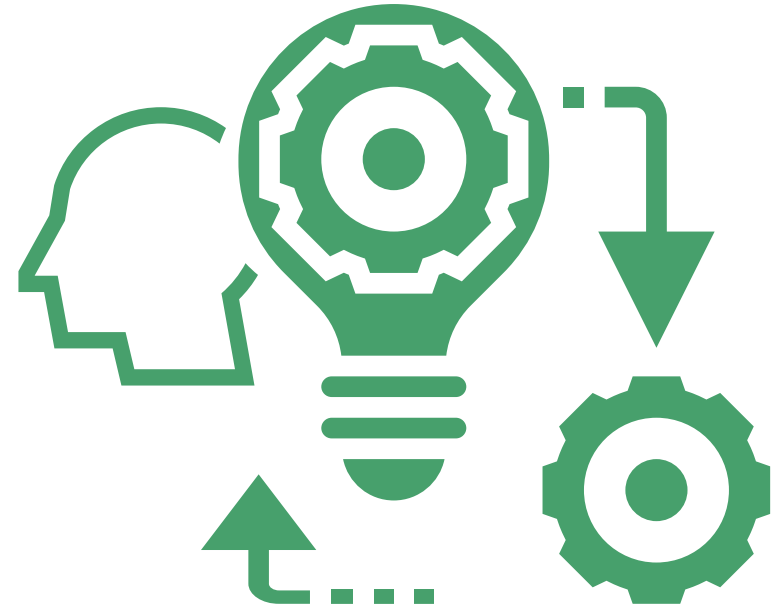
## TIOBE Programming Community Index

Source: [www.tiobe.com](http://www.tiobe.com)





Aug 2025	Aug 2024	Change	Programming Language		Ratings	Change
1	1			Python	26.14%	+8.10%
2	2			C++	9.18%	-0.86%
3	3			C	9.03%	-0.15%
4	4			Java	8.59%	-0.58%
5	5			C#	5.52%	-0.87%
6	6			JavaScript	3.15%	-0.76%
7	8	^		Visual Basic	2.33%	+0.15%
8	9	^		Go	2.11%	+0.08%
9	25	^^		Perl	2.08%	+1.17%
10	12	^		Delphi/Object Pascal	1.82%	+0.19%
11	10	v		Fortran	1.75%	-0.03%



# Paradigmas de Programação

Qual a Definição?

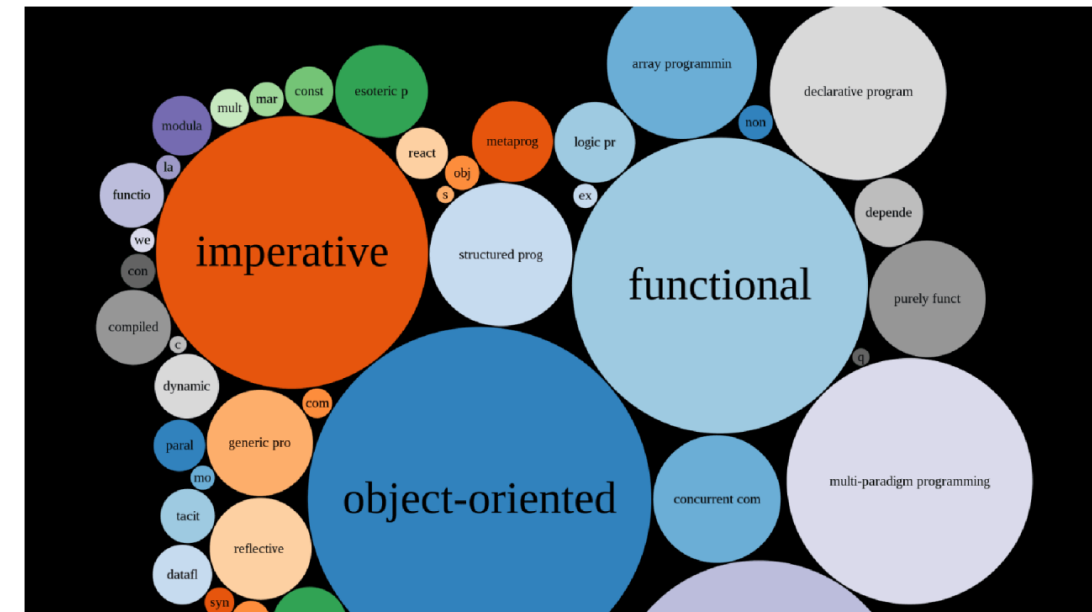
Porque estudar se eu já sei programar?

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.

- 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.**

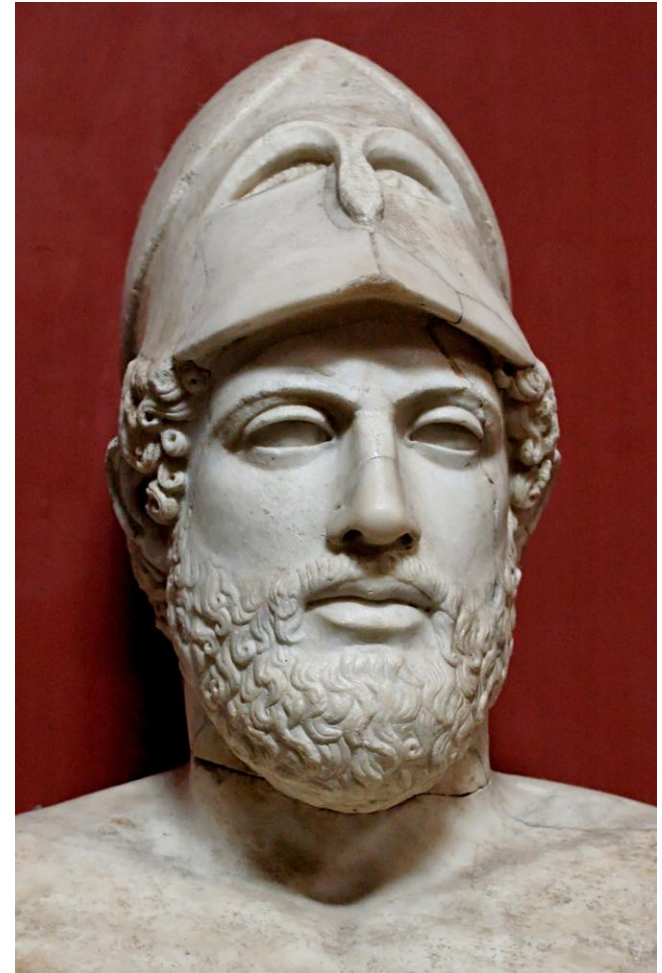
**“Vocabulário” mais rico = melhor expressão de “ideias”.**

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

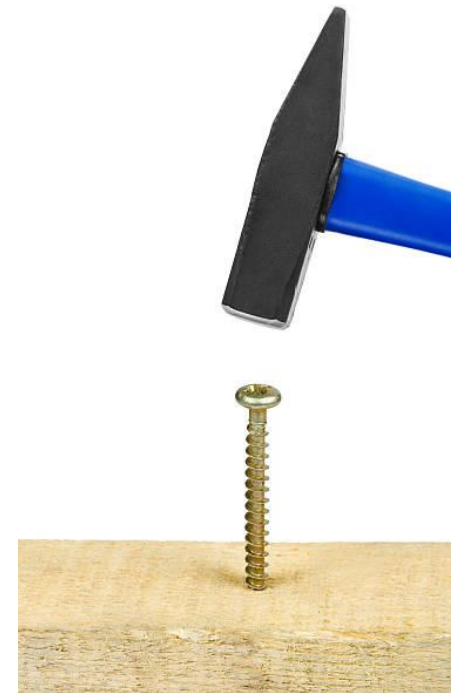


- Mas se você conhece um pouco do Paradigma Funcional, saberá que pode melhorar esse código com funções Lambdas

```
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

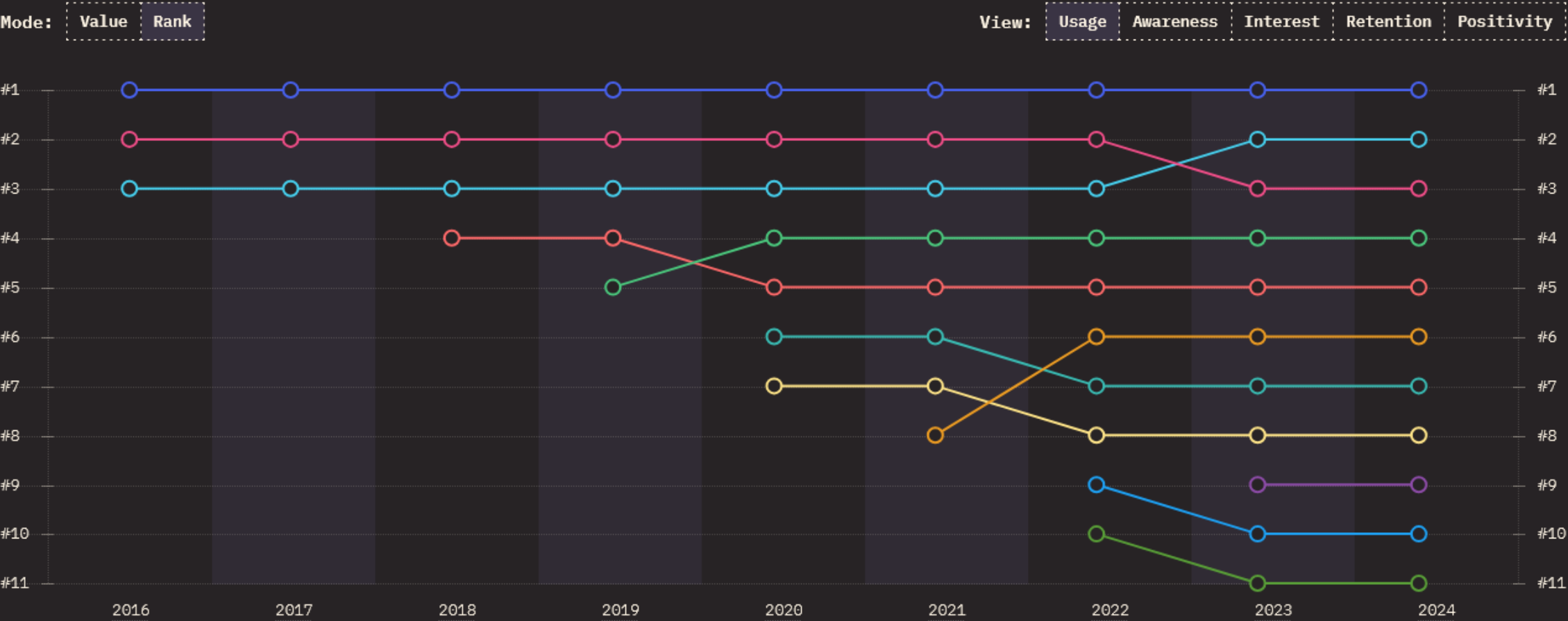
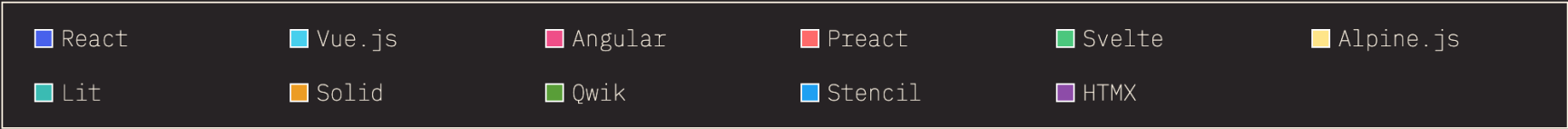
- Geralmente, um programador sem formação mais profunda, tende a escolher a sua **linguagem de preferência** para todo e qualquer projeto em que participa, mesmo que ela **não seja a escolha mais adequada**.
- Ao conhecer uma **faixa mais ampla de paradigmas e linguagens**, o desenvolvedor torna-se capacitado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

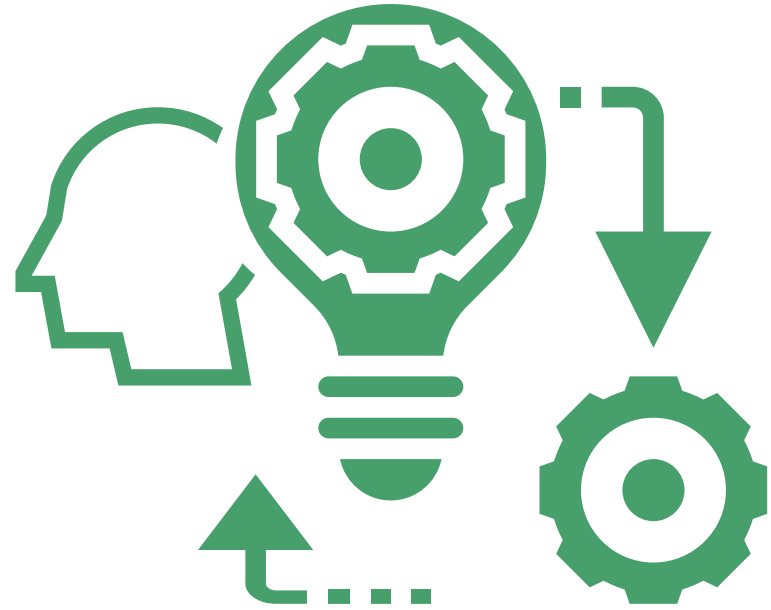


# Aumento da habilidade para aprender novas linguagens e/ou novos frameworks

- 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

# FRONT-END FRAMEWORKS RATIOS OVER TIME





# 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[...];
```

The diagram is a hierarchical flowchart on a dark blue background. At the top is a light blue rounded rectangle containing the text 'Paradigmas de programação'. Two pink arrows point downwards from this box to two separate pink rounded rectangles: 'imperativo' on the left and 'declarativo' on the right. From the 'imperativo' box, a vertical white line descends with three horizontal white arrows pointing to the right, labeled 'programação orientada a objetos', 'programação procedural', and '...'. Similarly, from the 'declarativo' box, a vertical white line descends with three horizontal white arrows pointing to the right, labeled 'programação funcional', 'programação lógica', and '...'.

imperativo

declarativo

programação  
orientada a objetos

programação  
procedural


...

programação  
funcional

programação  
lógica

...



Aspecto	Paradigma Imperativo	Paradigma Declarativo
Definição	Foca em <b><u>como fazer algo</u></b> , detalhando o passo a passo.	Foca em <b><u>o que deve ser feito</u></b> , 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';
```

# Entregáveis

Vamos lá. Está na hora!!!



