# Lógica de Programação

Pseudo código

O pseudocódigo consiste em um texto estruturado com comandos escritos em linguagem humana, no qual se apoia a criação dos algoritmos computacionais. É uma forma de representar código, sejam **algoritmos, funções ou outros processos**, usando uma combinação de linguagem natural e elementos que se parecem com linguagem de programação.

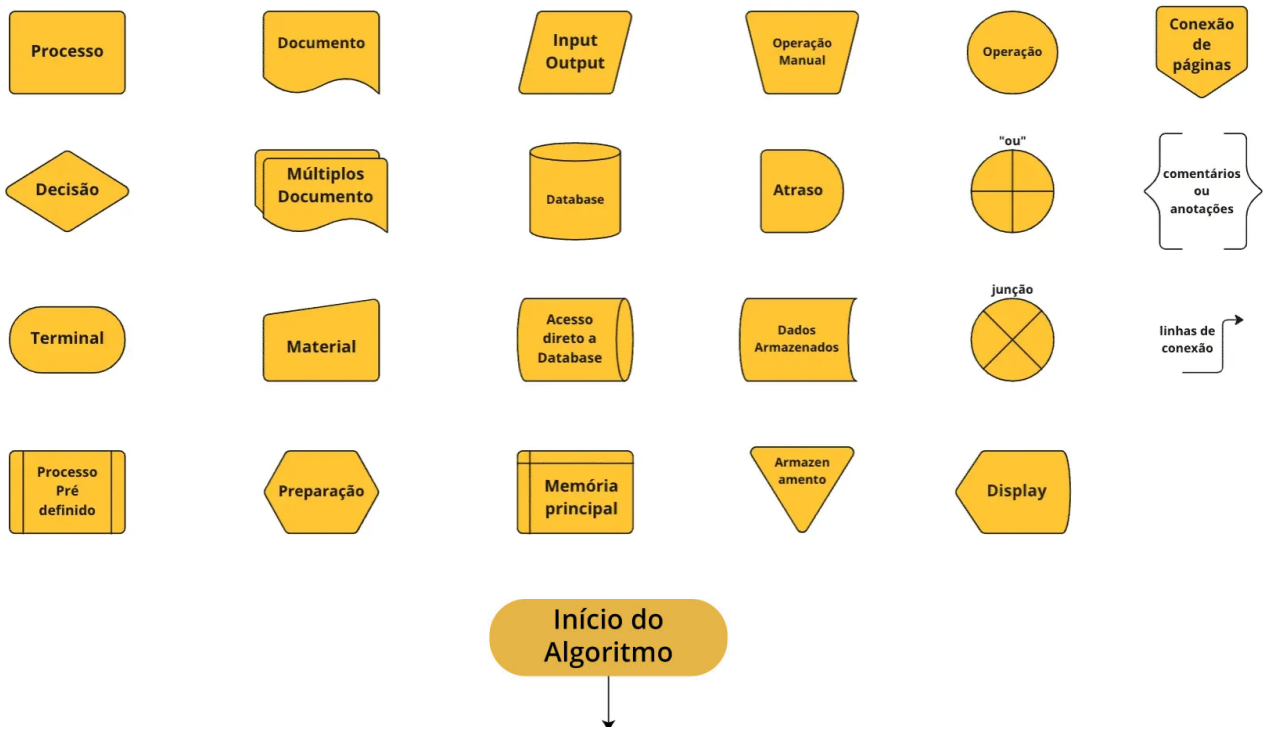
Exemplo em portugol:

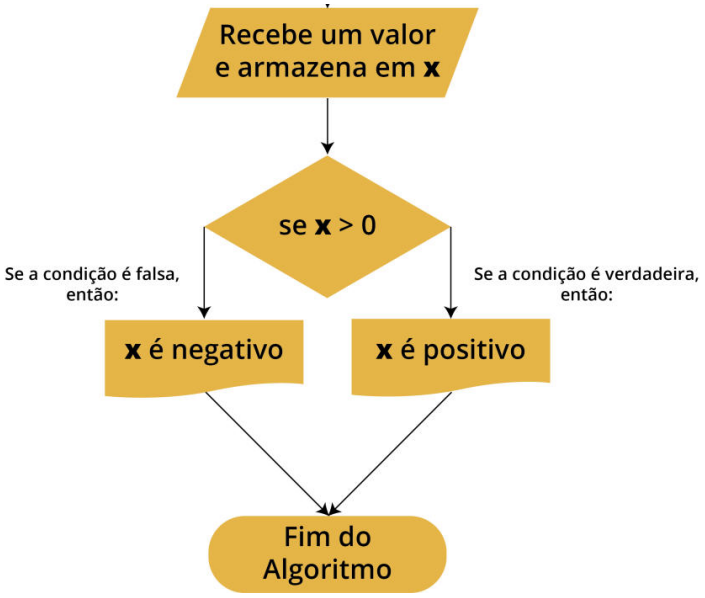
```
programa
{
    funcao inicio ()
    {
        inteiro menor, idade

        escreva("Informe sua idade: ")
        leia(idade)

        se (idade < 18)
        {
            escreva("Você é menor de idade")
```

Fluxograma faz o mesmo que o pseudo código mas de forma gráfica; são representações gráficas de processos que ajudam a visualizar o fluxo de atividades e decisões em um sistema.






Algoritmo representa uma sequência finita e bem definida de instruções elementares, de acordo com a solução de determinado problema. Cada sequência finita pode ser executada mecanicamente em um tempo finito. Assim as características de um algoritmo são: finitude, clareza e a capacidade de ser executado em um tempo determinado.

Em um algoritmo, os resultados de operações de entrada de dados geralmente são armazenados em variáveis, que ocupam posições na memória. Isso permite que os dados sejam utilizados posteriormente no algoritmo para processamento, cálculos ou outras operações.

Tabela verdade

Como calcular as linhas que uma tabela verdade deve ter, em função do número de variáveis?

Com a fórmula $2^v$ onde $v$ são as variáveis, exemplo:

Considere que uma expressão lógica envolva candidato (C), cargo político (P), votos (V) e ganhador (G). Para avaliar se uma dada expressão é verdadeira ou não, um Técnico deve usar uma Tabela da Verdade, que contém uma lista exaustiva de situações possíveis envolvendo as 4 variáveis. A Tabela da Verdade deve ter 4 colunas (uma para cada variável) e 16 linhas, pois $2^4 = 16$.

Variáveis

Como definir, modificar e imprimir o conteúdo de variáveis em pseudocódigo:

```
inteiro variavel = 0;
variavel = 1 + 2;
imprime(variavel);
```

Em algumas linguagens que utilizam a programação estruturada, as variáveis locais,

usadas apenas dentro do escopo da função em que são declaradas, não permitem que dados importantes sejam acessados pelas demais funções do programa, isso se chama escopo das variáveis.

```
funcao funcaoA() {
    inteiro x
    x := 10
    escreva("Valor de x em funcaoA: ", x, "\n")
}

funcao funcaoB() {
    inteiro x
    x := 20
    escreva("Valor de x em funcaoB: ", x, "\n")
}

inicio
    funcaoA()
    funcaoB()
fimalgoritmo
```

Em programação estruturada, funções são blocos de código que executam uma tarefa específica e podem ser reutilizadas em diferentes partes de um programa.

## ## Programação Estruturada

**Programação estruturada** é um paradigma de desenvolvimento de software que se baseia no uso de estruturas de controle bem definidas, como sequências, decisões condicionais e laços de repetição, para organizar o fluxo de execução de um programa. Ela visa melhorar a legibilidade, manutenibilidade e confiabilidade dos programas, ao evitar o uso de comandos de desvio, como o `GOTO`, que podem tornar o código confuso.

A programação estruturada surgiu na década de 1960, como resposta à crescente complexidade dos programas e à dificuldade de manutenção e compreensão de códigos que usavam extensivamente o comando `GOTO`, gerando o "código espaguete". Um dos marcos da programação estruturada foi o artigo de Edsger Dijkstra, em 1968, intitulado "Go To Statement Considered Harmful", no qual ele argumentava que o `GOTO` deveria ser evitado para melhorar a clareza do código.

> *Uma característica marcante da programação estruturada é o não uso de comandos de desvio, como, por exemplo, o GOTO (ou transferência unilateral de controle) em que o fluxo do programa fica confuso (difícil de entender e não linear. Em vez disso, ela promove o uso de estruturas de controle como **sequências**, **decisões condicionais** (`if`, `switch`) e **laços de repetição** (`for`, `while`) para organizar o fluxo do programa de maneira clara e previsível, promovendo um fluxo de controle mais disciplinado e*

> *hierárquico.*

A adoção desse paradigma foi impulsionada pelo desenvolvimento de linguagens de programação como Algol, Pascal, e posteriormente C, que incorporaram essas práticas e ajudaram a definir a base para o desenvolvimento de software moderno.

**Principais características da programação estruturada**

1. **Divisão do programa em módulos**: Programas são divididos em blocos menores e independentes (funções, procedimentos, sub-rotinas), cada um com uma responsabilidade específica. Isso facilita a leitura e a manutenção do código.
2. **Uso de estruturas de controle bem definidas**:
   - **Sequência**: Execução das instruções em ordem linear.
   - **Decisão condicional**: Estruturas como `if`, `else`, e `switch`, que permitem ao programa tomar diferentes caminhos de execução com base em condições.
   - **Laços de repetição**: Estruturas como `for`, `while`, e `do-while`, que permitem a repetição de blocos de código até que uma condição seja satisfeita.
3. **Hierarquia de chamadas**: As funções e procedimentos são organizados de forma hierárquica, com funções de alto nível controlando funções de nível mais baixo.
4. **Facilidade de depuração**: Como o código é organizado em módulos e o fluxo de controle é claro, a depuração de erros se torna mais fácil e sistemática.

**Modularidade** é um princípio fundamental da programação estruturada que consiste em dividir um programa em partes menores e independentes chamadas **módulos**. Cada módulo é responsável por uma tarefa ou funcionalidade específica do programa. Isso facilita a organização, o desenvolvimento e a manutenção do código, tornando-o mais claro e compreensível.

> *Na programação estruturada, a redução dos níveis de complexidade é alcançada pela divisão do programa em módulos funcionais. No entanto, esses módulos são organizados de forma hierárquica. A hierarquia permite que módulos de alto nível chamem módulos de nível inferior, estabelecendo uma estrutura clara de controle e fluxo de dados.*

Módulos/funções hierárquicas ocorre em códigos que possuem a função `main()` que tem um nívem mais alto pois é a função principal, responsável por controlar o fluxo do programa. Outras funções, como por exemplo `calcularImposto()` tem um nível intermediário, sendo chamadas por `main()` para realizar tarefas específicas. Elas estão em um nível mais baixo na hierarquia, pois são subordinadas ao controle do `main()`.

Essa estrutura facilita o entendimento do código, já que as funções têm responsabilidades bem definidas e são chamadas de forma organizada e hierárquica, indo do nível mais geral (controle) ao mais específico (operações).

> *O resultado de uma função pode ser usado como operando em uma expressão aritmética de um programa ou algoritmo de programação.*

Certo. Isso é comum em muitas linguagens de programação, onde o valor retornado por uma função pode ser diretamente utilizado em cálculos ou outras operações. `resultado = multiplicacao(soma(2, 3), 4)` em `soma(2, 3)` tem-se o que o enunciado diz.

**Depuração é** o processo de identificar, isolar e corrigir erros ou falhas no código. Por meio da depuração, é possível investigar a ocorrência de erros no programa. É uma atividade que pode ocorrer durante e após os testes. A depuração geralmente ocorre após a compilação, quando o programa já está sendo executado.A depuração pode ser executada por meio de pontos de parada (breakpoints).

**Breakpoints** são pontos definidos no código de um programa onde a execução é interrompida durante a depuração. Eles permitem que o desenvolvedor pause a execução do programa em um estado específico para investigar variáveis, a lógica do código e a sequência de execução. Isso facilita a identificação de bugs e o entendimento do comportamento do programa.

**Testes** ocorrem para verificar se um software funciona conforme o esperado. Eles envolvem a execução do código com várias entradas para garantir que o sistema se comporte corretamente.

Tabela comparativa entre depuração e TEstes:

| **Aspecto** | **Testes** | **Depuração** |
|---|---|---|
| **Definição** | Verificação do comportamento do software. | Processo de identificar e corrigir erros. |
| **Objetivo** | Garantir que o software funcione conforme esperado. | Encontrar a causa de erros e corrigi-los. |
| **Fases** | Ocorre durante o ciclo de desenvolvimento (testes unitários, de integração, etc.). | Pode ocorrer a qualquer momento, após a detecção de um erro. |
| **Métodos** | Execução de casos de teste com entradas variadas. | Análise do código, uso de breakpoints, inspeção de variáveis. |
| **Foco** | Comportamento funcional do software. | Lógica e estrutura do código. |
| **Timing** | Antecede o lançamento do software. | Ocorre conforme necessário após a identificação de falhas. |

## ### Estruturas

> *Com a finalidade de minimizar a complexidade dos programas, a programação estruturada ~~permite o uso de um número ilimitado de estruturas de controle~~ dentro de cada módulo*

Um dos princípios centrais da programação estruturada é a **simplicidade** e a **clareza** do código, o que implica em limitar a complexidade dentro de cada módulo. Por isso, um número elevado de estruturas de controle dentro de um módulo deve ser evitado, e funções longas ou complexas devem ser divididas em partes menores para manter a legibilidade e a simplicidade.

- Estruturas de **seleção (controle condicional)**: *if...else, switch*
- Estruturas de **iteração (repetição, loop)**: *while*, *for*
  - *O comando* `for` *("para") é ideal para contagem controlada e iteração sobre listas ou arrays. Sua sintaxe inclui a inicialização de uma variável, uma condição de continuação e uma atualização, tornando-o menos propenso a erros de loop infinito. É amplamente utilizado em algoritmos que requerem um número fixo de repetições.*
  - As estruturas de repetição `while` ("enquanto") e `do...while` ("repita até") são usadas quando não se conhece o número exato de iterações. A estrutura "repita até" sempre executa o bloco de código pelo menos uma vez, pois a condição é verificada após a execução do bloco.
- `continue` é usado para pular a iteração atual de um loop e continuar com a próxima iteração. Ele não interrompe o loop, mas simplesmente faz com que a execução avance para a próxima repetição.
- `break` é utilizado para interromper a execução de um loop ou de um bloco de controle, e fazer com que a execução continue a partir da próxima linha após o loop ou bloco.
- Variável de Controle é uma ou mais variáveis que controlam a execução do loop. Por exemplo, um contador que é incrementado a cada iteração.É importante garantir que a condição que determina o término do loop seja atualizada a cada iteração para evitar loops infinitos.

> *A temperatura média corporal de uma pessoa costuma variar entre 36 °C e 37,3 °C (**normal**); mais do que isso caracteriza a **febrícula** (até 37,8 °C), a **febre** (acima de 37,8 °C até 39 °C) e a **febre alta** (mais de 39 °C). Tendo como referência isso, julgue o item seguinte, relativo a construção de algoritmos e estruturas de controle, seleção, repetição e desvio.*
>
> *Em um algoritmo desenvolvido para identificar um dos cenários (normal ou anormal) referentes à temperatura corporal de uma pessoa, serão necessárias pelo menos **quantas** estruturas do tipo se-então-senão?*

Resposta: 4

1. Se a temperatura estiver entre 36 °C e 37,3 °C, então é normal.
2. Se a temperatura estiver entre 37,4 °C e 37,8 °C, então é febrícula.
3. Se a temperatura estiver entre 37,9 °C e 39 °C, então é febre.
4. Se a temperatura for maior que 39 °C, então é febre alta.

> *A estrutura de controle IF, que pode ser classificada como do tipo ~~iteração~~, determina o caminho que o algoritmo deve seguir, de acordo com determinada condição.*

Errado. A estrutura de controle IF é classificada como uma estrutura de seleção, não de iteração.

> *Um programa que utiliza uma estrutura de controle do tipo `repita até` para realizar certo conjunto de instruções continuará sendo executado enquanto o resultado do teste de controle for falso.*

> *Um loop que sempre se repetirá em um determinado número de vezes é representado pelo código ~~SE ENTÃO SENÃO FIM SE~~.*

Errado. A estrutura "SE ENTÃO SENÃO FIM SE" representa uma estrutura de controle condicional (ou seleção), não um loop. Um loop que se repete um determinado número de vezes é geralmente representado por estruturas como "FOR" ou "WHILE".

## ### Recursão

Uma função recursiva é uma função que se chama novamente dentro de sua própria definição para resolver subproblemas. É composta por:

- A **condição/caso base** é uma condição que determina quando a função deve parar de se chamar recursivamente. Geralmente, é um caso simples que pode ser resolvido diretamente (por exemplo, o fatorial de 0 ou 1 é 1).
- O **caso recursivo** em uma função recursiva é a parte da função que realiza a chamada à própria função, permitindo que o problema seja dividido em subproblemas menores.

Quando uma função é chamada, suas variáveis locais e o ponto de retorno são armazenados em uma **pilha de chamadas**. Cada chamada cria uma nova entrada na pilha. Quando a condição base é atingida, as chamadas começam a ser resolvidas e retiradas da pilha na ordem inversa em que foram chamadas.

```
// Função f1: cálculo do fatorial recursivamente
function f1(N) {
    if (N <= 1) {
```

```
        return 1; // Base da recursão
    } else {
        return N * f1(N - 1); // Chamada recursiva
    }
}
// Exemplo de uso
console.log(f1(5)); // Saída: 120
```

A recursividade possibilita a escrita de um código mais enxuto, com maior legibilidade, embora seja conceitualmente seja mais complexo.

Abaixo uma comparação entre recursividade e iteratividade (ou loop):

| **Critério** | **Funções Recursivas** | **Funções Iterativas** |
|---|---|---|
| **Definição** | Funções que se chamam a si mesmas | Funções que usam estruturas de repetição |
| **Estrutura** | Possui condição base e chamadas recursivas | Utiliza laços (`for`, `while`, etc.) |
| **Abordagem** | Top-down (decompõe o problema) | Bottom-up (constrói a solução passo a passo) |
| **Uso de Memória** | Utiliza a pilha de chamadas | Utiliza espaço de memória fixo |
| **Desempenho** | Pode ser menos eficiente em termos de tempo e memória | Geralmente mais eficiente |
| **Legibilidade** | Pode ser mais intuitiva para problemas recursivos | Pode ser mais complexa para problemas complexos |
| **Complexidade** | Pode simplificar a lógica para problemas naturais (como árvores) | Pode ser mais complicada para gerenciar estados |
| **Risco de Erros** | Suscetível a estouro de pilha se não bem gerenciada | Menos suscetível a erros de pilha, mas pode ter loops infinitos |

Embora muitos algoritmos recursivos tenham equivalentes iterativos, nem todos os algoritmos iterativos possuem uma versão recursiva e vice-versa.

# Paradigma de orientação a objetos

Uma **classe** é um modelo ou um template que define um tipo de objeto. Ela encapsula dados (propriedades) e comportamentos (métodos) que os objetos desse tipo terão.

Um **método** é uma função definida dentro de uma classe que descreve um comportamento que os objetos dessa classe podem executar.

As **propriedades** (ou atributos) são variáveis que armazenam o estado ou as características de um objeto. Elas são definidas dentro da classe e podem ser acessadas e modificadas pelos métodos.

Uma **instância** é um objeto criado a partir de uma classe. Quando uma classe é instanciada, ela cria um novo objeto que possui suas próprias cópias das propriedades definidas na classe.

> *Os valores atribuídos aos atributos do objeto fazem dele um objeto único.*

> *Na programação orientada a objetos, as instâncias são criadas a partir de uma classe e compartilham os métodos e atributos dessa classe, ~~assim como os conteúdos desses atributos~~.*

Errado, pois ao criar o objeto os métodos e atributos são usados da classe, mas o conteúdo ou valor dos atributos não é compartilhado.

Um **método construtor** é um método especial que é chamado automaticamente quando uma nova instância de uma classe é criada. Ele é usado para inicializar as propriedades do objeto.

Em programação orientada a objetos, uma **classe abstrata** é uma classe que não pode ser instanciada diretamente e serve como um modelo para outras classes. Para que uma classe seja considerada abstrata, é necessário que ela contenha pelo menos um **método abstrato**. Um método abstrato é um método que é declarado na classe abstrata, mas não possui uma implementação (ou corpo). As subclasses que herdam da classe abstrata são obrigadas a implementar esses métodos abstratos.

**Interfaces** em programação orientada a objetos são contratos que definem um conjunto de métodos que uma classe deve implementar. Elas permitem que diferentes classes compartilhem um comportamento comum sem a necessidade de uma hierarquia de classes.

> *Quatro principais conceitos em que a programação orientada a objetos é baseada: abstração; encapsulamento; herança; polimorfismo. Os pilares da orientação a objetos são classe, objeto, atributo e método.*

## Abstração

**Abstração** em programação orientada a objetos (POO) é o princípio que permite representar conceitos complexos de forma simplificada, focando apenas nas características essenciais de um objeto, enquanto oculta os detalhes desnecessários. A abstração ajuda a reduzir a complexidade do sistema, permitindo que os desenvolvedores se concentrem no que é relevante para a aplicação em vez de se perderem em detalhes de implementação.

> *Na programação orientada a objetos, a abstração é a característica que permite a utilização de um código ou função sem a necessidade de conhecer detalhes sobre sua implementação.*

> *Na arquitetura orientada a objeto, os dados e as operações são encapsulados para facilitar a manipulação das informações.*

> *Em programação orientada a objetos, ~~abstração~~ é a propriedade que permite que um método de determinado nome tenha comportamentos distintos, em função de diferentes parâmetros recebidos.*

Errado. A definição apresentada se refere ao polimorfismo, não à abstração.

## Encapsulamento

O encapsulamento é um dos princípios fundamentais da programação orientada a objetos e se refere à prática de esconder os detalhes internos de uma classe e expor apenas o que é necessário através de uma interface pública. Isso permite que um programa seja dividido em partes menores (ou classes), mas o objetivo do encapsulamento é reduzir a dependência entre essas partes, promovendo a modularidade e a manutenção do código.

> *Encapsulamento identifica o princípio de ocultar os detalhes internos de funcionamento de uma classe e expor apenas o que é necessário por meio de interfaces públicas.*
>
> *Uma classe encapsula os dados e o processamento que os manipula, ocultando a informação e reduzindo o impacto de efeitos colaterais associados a modificações.*

> *O encapsulamento permite que um programa seja dividido em várias partes menores; ~~contudo, as partes tornam-se dependentes umas das outras em relação à implementação e em relação ao trabalho realizado~~.*

Errado. Na verdade, o encapsulamento busca minimizar essa dependência, permitindo

que as partes do programa possam ser alteradas ou substituídas sem afetar outras partes, desde que a interface pública permaneça a mesma.

# ## Polimorfismo

Polimorfismo é um dos princípios fundamentais da programação orientada a objetos e que significa "muitas formas". O **polimorfismo** permite que diferentes classes implementem métodos com o mesmo nome, mas com comportamentos distintos. Isso significa que você pode chamar o mesmo método em diferentes objetos, e cada objeto pode responder de maneira apropriada, dependendo de sua classe específica.

> *Em orientação a objetos, o conceito utilizado para descrever os vários comportamentos que um método possui, visando a um melhor aproveitamento de partes de um código, é denominado polimorfismo.*
>
> *Polimorfismo é a capacidade de escrever métodos que se comportem corretamente para objetos de tipos diferentes.*
>
> *No paradigma da orientação a objetos, o polimorfismo permite que várias operações distintas possuam o mesmo nome, desacoplando, assim, os objetos uns dos outros, tornando-os mais independentes.*

Assinatura de um método refere-se à combinação do seu nome e dos parâmetros que ele aceita. A assinatura é o que distingue um método de outro dentro de uma mesma classe ou entre métodos em classes relacionadas. Abaixo os métodos tem assinaturas diferentes mesmo tendo o mesmo nome:

```
Plain text
void calcular(int a) { ... }
void calcular(double a) { ... }
void calcular(int a, int b) { ... }
```

Existem duas formas principais de polimorfismo:

1. **Polimorfismo de Sobrecarga (Overloading)** : Ocorre quando duas ou mais funções ou métodos têm o mesmo nome, mas diferentes assinaturas (número ou tipo de parâmetros). O compilador determina qual método chamar com base nos argumentos fornecidos.

Por exemplo, você pode ter um método `soma` que aceita dois inteiros e outro que aceita dois números de ponto flutuante.

> *Na programação orientada a objetos, o mecanismo pelo qual dois métodos de mesma classe podem ter o mesmo nome, desde que suas listas de parâmetros sejam diferentes, é chamado de sobrecarga.*

1. **Polimorfismo de Sobrescrita (Overriding)** : Ocorre quando uma subclasse fornece uma implementação específica de um método que já está definido na sua superclasse. Isso permite que a subclasse altere o comportamento do método herdado.

Por exemplo, se você tem uma classe `Animal` com um método `fazerSom`, e subclasses `Cachorro` e `Gato` que implementam `fazerSom` de maneiras diferentes (latido e miado, respectivamente), você pode chamar `fazerSom` em um objeto `Animal` e obter o som apropriado, dependendo do tipo real do objeto.

> *A partir de uma classe derivada de uma superclasse, podem-se invocar métodos que tenham a mesma assinatura, mas comportamentos distintos, ou seja, em que haja alteração do funcionamento interno de um método herdado de um objeto pai. Na orientação a objetos, isso é possível por meio do polimorfismo.*

# ## Herança

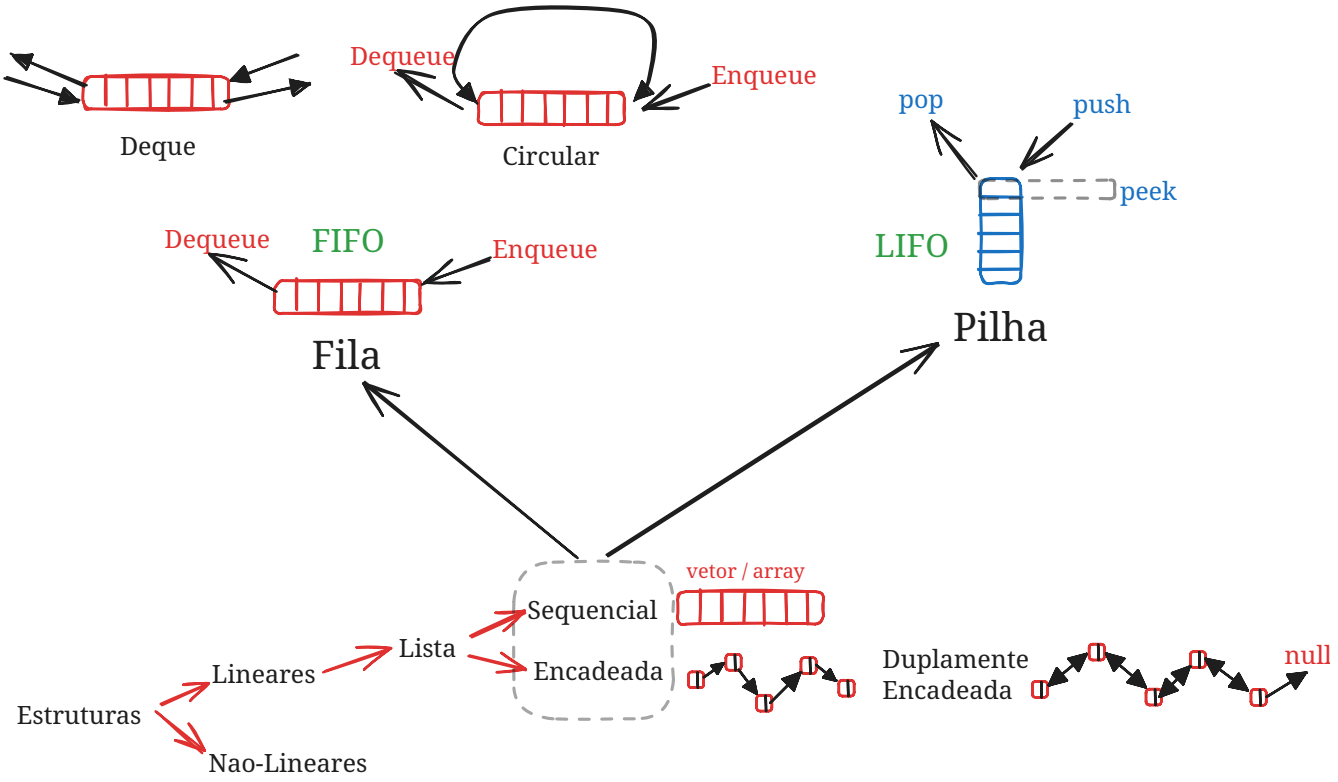
Herança é um dos conceitos fundamentais da programação orientada a objetos (POO) que permite que uma classe (chamada de subclasse ou classe derivada) herde atributos e métodos de outra classe (chamada de superclasse ou classe base). Esse mecanismo promove a reutilização de código e a criação de hierarquias de classes, facilitando a organização e a manutenção do software.

Em muitas linguagens de programação orientadas a objetos, a palavra-chave `extends` é utilizada para indicar que uma classe está herdando de outra. No entanto, a sintaxe pode variar de uma linguagem para outra.

> *Em orientação a objetos, o mecanismo de se criar novas classes a partir de uma classe existente é denominado herança.*
>
> *Herança é uma característica do paradigma orientado a objetos, a qual possibilita que haja hierarquia de classes, de forma que as alterações em uma classe-pai possam ser imediatamente propagadas para a classe-filha.*
>
> *Na orientação a objetos, os conceitos de superclasse e subclasse estão relacionados a herança.*

# Estrutura de Dados

Estruturas de dados são formas de organizar e armazenar dados em um computador de maneira que possam ser utilizados de forma eficiente. Elas são fundamentais para a programação e o desenvolvimento de algoritmos, pois influenciam diretamente a eficiência e a complexidade das operações realizadas sobre os dados.




**Tipos de Estruturas de Dados**

1. **Estruturas Lineares**: Os elementos são organizados em uma sequência linear. Exemplos incluem:
   - **Vetor**: Uma coleção de elementos de mesmo tipo, armazenados em posições contíguas na memória.
   - **Lista**: Uma coleção de elementos que pode ser armazenada de forma contígua (como um vetor) ou não contígua (como uma lista encadeada).
2. **Estruturas Não Lineares**: Os elementos não seguem uma sequência linear. Exemplos incluem árvores e grafos.

## ## Lista

**Lista** é uma estrutura de dados que armazena uma coleção de elementos em uma sequência linear. Podem ser sequencial ou encadeada, e a principal diferença entre essas duas implementações é a **forma como os elementos são armazenados e acessados**, mas em ambas as estruturas, os elementos estão organizados em uma sequência.

> *Em um programa, uma estrutura do tipo lista armazena, de maneira seqüencial ou*

> *encadeada, os elementos dispostos um após o outro.*

- Uma lista sequencial é uma estrutura de dados que armazena elementos em uma sequência contígua de memória, como em vetor ou array. Os elementos são acessados por meio de índices, que representam suas posições na lista.
- Uma lista encadeada é uma estrutura de dados onde cada elemento (ou nó) contém um valor e um ponteiro (ou referência) para o próximo nó na sequência, como uma lista simplesmente ou duplamente encadeada. Isso permite que os elementos não estejam armazenados em locais contíguos de memória. A inserção e remoção de elementos em uma lista encadeada são mais eficientes do que em uma lista estática, pois não é necessário mover os elementos existentes; basta ajustar os ponteiros.
  - Em listas encadeadas o último elemento aponta para `null` (ou `None`, dependendo da linguagem), indicando que não há mais elementos.
- Uma lista duplamente encadeada é uma variação da lista encadeada, onde cada nó contém referências para o nó anterior e para o próximo. Sua principal vantagem é que se pode percorrer a lista em ambas as direções (para frente e para trás).

> *Listas encadeadas e duplamente encadeadas são estruturas de dados, nas quais os objetos estão organizados em ordem linear.*
>
> *Em uma lista duplamente encadeada, cada elemento deve apontar para o elemento anterior e para o próximo.*
>
> *Uma estrutura do tipo lista, em que é desejável percorrer o seu conteúdo nas duas direções indiferentemente, é denominado lista duplamente encadeada.*

## ## Fila

Uma Fila é uma estrutura de dados linear que segue o princípio FIFO (First In, First Out), ou seja, o primeiro elemento a entrar é o primeiro a sair. Suas operações incluem: **Enqueue a**diciona um elemento ao final da fila e **Dequeue** remove o elemento do início da fila. Pode ser implementada usando alocação sequencial (vetores) ou alocação encadeada (listas encadeadas).

- Deque (Double-Ended Queue) é uma fila que permite a inserção e remoção de elementos em ambas as extremidades (início e fim)
- Fila circular **é** uma variação da fila que conecta o final de volta ao início, mas ainda assim permite apenas inserções no final e remoções no início.

> *Sempre que houver uma remoção na estrutura de dados denominada fila, o elemento removido será aquele que está na estrutura há mais tempo.*
>
> *A estrutura de dados do tipo fila utiliza o conceito de FIFO, ou seja, os elementos são*

> *atendidos, sequencialmente, na ordem em que são armazenados.*

> *Os elementos de uma fila poderão ser retirados somente na ordem inversa em que foram inseridos, ou seja, respeitando-se o conceito ~~last in, first out~~.*

A afirmação está incorreta. Em uma fila, os elementos são retirados na ordem em que foram inseridos, respeitando o conceito FIFO (First In, First Out)

> *Se os processos em um tribunal forem analisados e numerados de acordo com a ordem em que chegam ao protocolo — sendo o primeiro processo que chega o primeiro a ser analisado —, ~~é mais adequado associá-los a uma lista linear dinâmica do tipo pilha do que a uma lista linear do tipo fila~~.*

Errado. A afirmação está incorreta. Se os processos em um tribunal são analisados na ordem em que chegam, onde o primeiro processo a chegar é o primeiro a ser analisado, isso se alinha ao conceito de FIFO (First In, First Out), que é característico de uma fila, não de uma pilha.

## ## Pilha

Uma Pilha é uma estrutura de dados linear que segue o princípio LIFO (Last In, First Out), ou seja, o último elemento a entrar é o primeiro a sair. Push adiciona um elemento ao topo da pilha, pop remove o elemento do topo da pilha e peek acessa o elemento do topo sem removê-lo.

> *Pilha é uma estrutura de dados do tipo lista linear, em que as operações TOP, PUSH e POP são realizadas no topo da pilha e, por isso, são denominadas LIFO.*
>
> *Se os elementos X, Y, W, Z, nessa ordem, forem colocados em uma pilha e excluídos um de cada vez, eles serão removidos na ordem invertida: Z, W, Y, X.*
>
> *Em um sistema operacional, a estrutura de dados utilizada para organizar chamadas de funções recursivas por meio da inserção ou remoção de elementos via operações como push e pop é denominada pilha.*

> *O técnico José implementou uma estrutura de dados linear na qual os elementos são organizados de modo a constituir uma sequência lógica. Na estrutura implementada por José, os elementos não são, necessariamente, armazenados de forma contígua na memória do computador, e o primeiro elemento a entrar é o último a sair. A estrutura de dados implementada por José é caracterizada especificamente como pilha com alocação encadeada.*

> *Pilhas são tipos de estruturas de dados que ~~permitem a remoção direta de qualquer elemento de sua estrutura~~.*

Errado. Pilhas são estruturas de dados que seguem o princípio LIFO (Last In, First Out), onde apenas o elemento que foi adicionado por último pode ser removido diretamente.

> *Nas estruturas de pilhas, a inserção de um novo item ou a remoção de um item já existente ~~ocorre tanto na extremidade de baixo quanto no topo da pilha~~.*

A afirmação está incorreta. Em uma pilha, a inserção (push) e a remoção (pop) de itens ocorrem apenas na extremidade do topo da pilha.

> *Em estruturas do tipo pilha, a inserção de um novo item é realizada em uma única extremidade: ~~na base~~.*

Errado. Em uma pilha, a inserção de um novo item (operação chamada de **push**) é realizada na extremidade do **topo** da pilha, não na base.

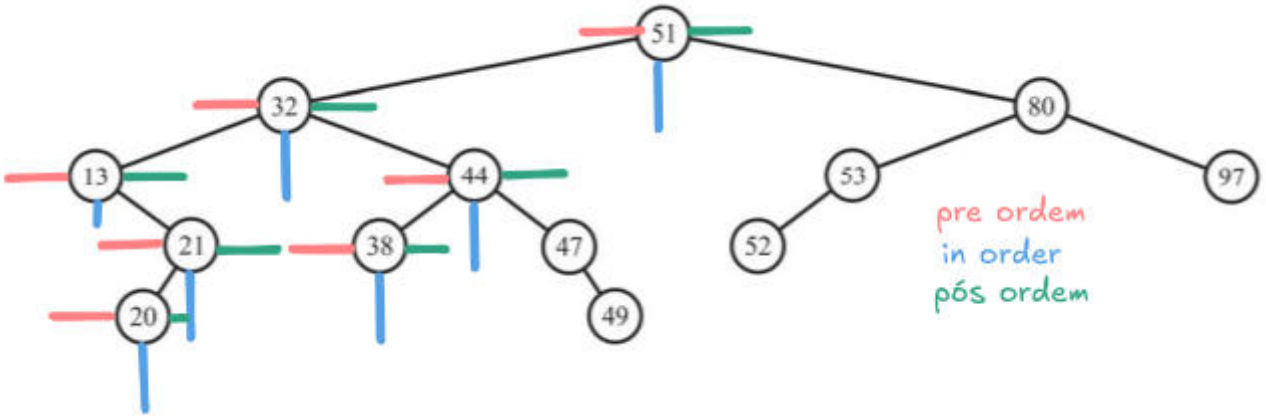
> *Uma pilha oferece as operações ~~pop para inserir~~ um elemento da pilha e ~~push para remover~~ o elemento no seu início.*

Errado. É pop para retirar e push para inserir.

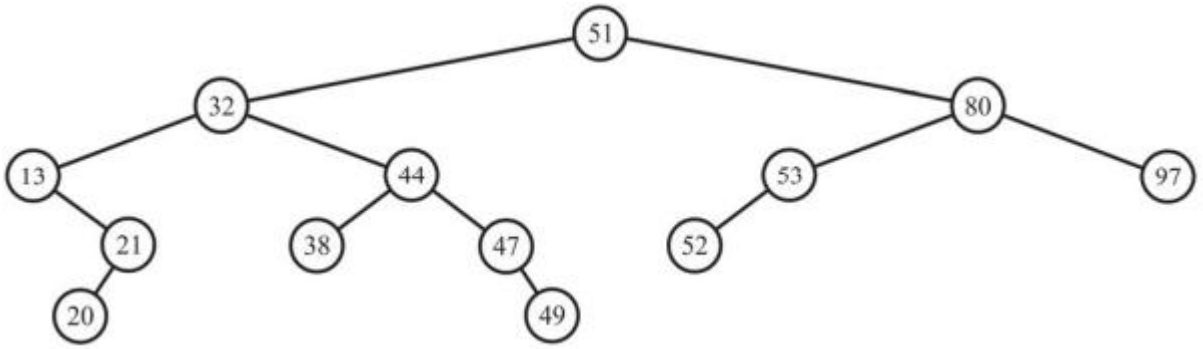
## ## Árvore

Uma árvore é uma estrutura de dados hierárquica composta por nós, onde cada nó contém um valor e referências para seus filhos. O nó superior é chamado de raiz, e os nós sem filhos são chamados de folhas. Cada nó pode ter zero ou mais filhos, e cada nó (exceto a raiz) tem exatamente um pai.

Tem as operações de inserção, remoção, busca e os percursos, onde se percorre todos os nós da árvore em uma ordem específica (pré-ordem, em-ordem, pós-ordem). Abaixo mostra-se as formas de percorrer uma árvore:



- **Árvore binária em que c**ada nó tem no máximo dois filhos (esquerdo e direito).
- **Árvore binária de busca (BST) é u**ma árvore binária onde os nós à esquerda de um nó têm valores menores e os à direita têm valores maiores.

> *Assinale a opção em que é apresentada a sequência correta de números correspondente à varredura da árvore binária representada a seguir quando esta estiver sendo percorrida em profundidade por meio da utilização da técnica do tipo pré-ordem.*
>
> 
> 

51, 32, 13, 21, 20, 44, 38, 47, 49, 80, 53, 52, 97

> *Que estrutura de dados como deve-se representar esse diretório abaixo:*
>
> *Dados*
> *├── Técnicos*
> *│ ├── Árvores*
> *│ ├── Hash*
> *│ ├── Recursão*
> *│ ├── Filas*
> *│ └── Pilhas*
> *└── Práticos*
> *├── Programas*
> *│ ├── AFazer*
> *│ └── Prontos*
> *│ ├── Eleições*
> *│ └── Urnas*
> *└── ComErro*
> *├── Corretos*
> *├── Urgentes*
> *├── Pendentes*
> *└── Antigos*

Para representar esses diretórios de um computador, a estrutura de dados a ser usada é a árvore, pois ela reflete a hierarquia dos dados de forma eficiente, permitindo múltiplos filhos por nó.

> *Uma estrutura do tipo árvore binária é caracterizada por não ter elemento algum (árvore vazia) ou ter um elemento denominado raiz, com dois ponteiros para duas estruturas diferentes.*

A afirmação está correta. Uma árvore binária é uma estrutura de dados que pode ser:

1. **Vazia:** Não contém nenhum elemento.
2. **Não vazia:** Contém um elemento chamado **raiz**. A raiz é o nó superior da árvore e pode ter até dois filhos, que são representados por ponteiros (ou referências) para duas subárvores diferentes, chamadas de subárvore esquerda e subárvore direita.

> *Uma árvore binária deve ter, no mínimo, ~~3 nós~~.*

Errado. Uma árvore binária pode ter 0, 1, 2 ou mais nós, dependendo da sua estrutura.

# HTML

> *O HTML é uma linguagem de marcação que estrutura o conteúdo da web e apresenta informações na tela do usuário.*

> *HTML5 suporta integralmente conteúdo de áudio e vídeo incorporado em navegadores, através das tags `<audio>` e `<video>`.*

> *HTML5 permite que o JavaScript forneça funcionalidade dinâmica para páginas da web usando APIs como Canvas 2D, Drag and Drop, Web Storage e Messaging.*

> *HTML5 foi projetado para garantir a compatibilidade com o código de marcação legado, a fim de evitar quebrar a web.*

> *A opção em que é indicado atributo HTML usado para especificar o destino de um link é `href` .*

> *Em HTML5, a tag `<output>` ~~fornece uma indicação ao usuário do que pode ser inserido no campo.~~*

Errado. A tag `<output>` é usada para representar o resultado de um cálculo ou outra forma de saída, como o resultado de um script. Ela é comumente usada em conjunto com elementos de formulário, como `<input>` e `<form>`, para exibir o resultado de uma operação.

Exemplo: Suponha que você tenha um formulário com dois campos de entrada numéricos, e você queira exibir a soma desses dois números. Você pode usar a tag `<output>` para isso:

Plain text

```
<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">
  <input type="number" id="a" name="a" value="0"> +
  <input type="number" id="b" name="b" value="0"> =
  <output name="x" for="a b"></output>
</form>
```

> *Cada elemento no HTML pode ou não fazer parte de um grupo de elementos com características similares. Representam categorias de elementos do HTML:*

A) Flow content. ~~B) Form content.~~ C) Phrasing content. D) Metadata content.

**Categoria de elementos do HTML** (Flow Phrasing e Metadata Content). Essas categorias ajudam a organizar e estruturar o conteúdo de uma página web, permitindo que os desenvolvedores entendam como os diferentes elementos podem interagir entre si.

- **Flow content** é uma categoria que inclui a maioria dos elementos HTML que podem ser usados em um documento. Elementos de flow content podem conter outros elementos de flow content, bem como elementos de phrasing content.
  - Exemplos de elementos de flow content incluem `<div>`, `<p>`, `<header>`, `<footer>`, `<section>`, e muitos outros. Essa categoria é essencial para a estruturação do conteúdo em uma página web.
- **Phrasing content** é uma categoria que inclui elementos que podem ser usados dentro de um fluxo de texto. Esses elementos geralmente são usados para formatar ou estilizar partes do texto.
  - Exemplos de elementos de phrasing content incluem `<span>`, `<a>`, `<strong>`, `<em>`, e `<img>`. Eles podem ser inseridos dentro de elementos de flow content e são usados para criar conteúdo mais rico e interativo.
- **Metadata content** é uma categoria que inclui elementos que fornecem informações sobre o documento, mas que não são diretamente exibidos na página. Esses elementos são geralmente encontrados dentro da seção `<head>` do HTML.
  - Exemplos de elementos de metadata content incluem `<title>`, `<meta>`, `<link>`, e `<style>`. Eles são usados para definir informações como o título da página, descrições, palavras-chave, e links para estilos ou scripts.

> *Dentre todas as categorias de modelos de conteúdo do HTML, existem dois tipos de elementos: elementos de linha eelementos de bloco. Os elementos de linha marcam, na maioria das vezes, textos, elementos de formulários, imagens. Alguns exemplos: a, strong, em, img, input, abbr, span. Já os elementos de blocos são como caixas, que dividem o conteúdo em seções do layout. Analise as afirmativas a seguir.I. Os elementos de linha não podem conter outros elementos de linha. FII. Os elementos de linha nunca podem conter elementos de bloco. VIII. Os elementos de bloco sempre podem conter elementos de linha. VIV. Os elementos de bloco podem sempre conter elementos de bloco F*

- Os **elementos de bloco** são aqueles que ocupam toda a largura disponível de seu contêiner, criando uma nova "linha" antes e depois do elemento. Eles são usados para estruturar o layout da página e dividir o conteúdo em seções.
  - Elementos de Bloco: `<div>`, `<p>`, `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, `<ul>`, `<ol>`, `<section>`, `<article>`
- Os **elementos de linha**, por outro lado, ocupam apenas o espaço necessário para seu conteúdo e não iniciam uma nova linha antes ou depois. Eles são frequentemente usados para formatar partes do texto ou para incluir pequenos

elementos dentro de um bloco.
    - Elementos de Linha: `<span>`, `<a>`, `<strong>`, `<em>`, `<img>`, `<input>`

| Situação | Pode? | Observação |
|---|---|---|
| Elemento de linha dentro de elemento de linha | DEPENDE | Dependendo da categoria que ele se encontra. Por exemplo: o elemento A não pode conter o elemento LABEL. Mas o inverso é possível |
| Elemento de bloco dentro de elemento de linha | NÃO | |
| Elemento de linha dentro de elemento de bloco | SIM | |
| Elemento de bloco dentro de elemento de bloco | DEPENDE | Dependendo da categoria que ele se encontra. Por exemplo, um parágrafo não pode conter um DIV. Mas o inverso é possível |

Assim sendo:

- I. Os elementos de linha não podem conter outros elementos de linha. F
- II. Os elementos de linha nunca podem conter elementos de bloco. V
- III. Os elementos de bloco sempre podem conter elementos de linha. V
- IV. Os elementos de bloco podem sempre conter elementos de bloco F

Plain text

```
<!DOCTYPE html>
<html>
<head>
    <style>
        a:after {
            content: " (" attr(href) ")";
        }
    </style>
</head>
<body>
    <p>
        <a href="www.tre-rs.jus.br">TRE/RS</a>
    </p>
</body>
</html>
```

Com relação ao CSS e HTML descritos no código, o resultado apresentado será: `TRE/RS`

`(http://www.tre-rs.jus.br)`

A regra CSS `a:after { content: " (" attr(href) ")"; }` faz o seguinte:

1. A regra se aplica a todos os elementos `<a>` (links) na página.
2. O `:after` é um pseudo-elemento que permite adicionar conteúdo após o conteúdo de um elemento selecionado. Neste caso, ele adiciona conteúdo após o texto do link.
3. A propriedade `content` é usada para definir o que será adicionado.
4. A função `attr(href)` é utilizada para pegar o valor do atributo `href` do link.

> *Entre os novos elementos do HTML5, o elemento `<progress>` define o progresso de uma tarefa.*

`<progress>` : Este elemento é usado para representar o progresso de uma tarefa em andamento, como o progresso de um download ou de uma operação de upload.

# DHTML

DHTML, ou Dynamic HTML, é uma combinação de HTML, CSS e JavaScript que permite a criação de páginas web interativas e dinâmicas, possibilitando a manipulação do Document Object Model (DOM) para alterar conteúdo, estrutura e estilo em tempo real, sem recarregar a página.

Ele possibilita a criação de animações, transições e elementos interativos, como menus suspensos, melhorando a experiência do usuário. Embora tenha sido popular nos anos 90 e início dos anos 2000, suas funcionalidades foram amplamente incorporadas em tecnologias mais modernas, como AJAX e frameworks JavaScript.

> *A definição do termo DHTML (Dynamic HTML) é a de ser uma combinação de HTML, CSS e JavaScript para criar páginas web interativas.*

> *A linguagem DHTML, unindo as tecnologias HTML, JavaScript e uma linguagem de apresentação, significa Dynamic Hypertext Markup Language.*

> *Conteúdos de páginas web implementadas com DHTML são atualizados ~~somente com a recarga da página inteira~~.*

Errado. DHTML permite que os conteúdos de páginas web sejam atualizados dinamicamente sem a necessidade de recarregar a página inteira, utilizando JavaScript para manipular o DOM e alterar elementos em tempo real.

> *A tecnologia DHTML é uma evolução do HTML na qual são adicionados recursos dinâmicos às páginas web.*

Certo. A tecnologia DHTML é, de fato, uma evolução do HTML que combina HTML, CSS e JavaScript para adicionar recursos dinâmicos e interativos às páginas web, permitindo que o conteúdo seja atualizado em tempo real sem a necessidade de recarregar a página.

# CSS

> *Assim como o HTML, ~~o CSS é uma linguagem de marcação~~, que define a estrutura do conteúdo da página web.*

Errado. O HTML é uma linguagem de marcação que define a estrutura e o conteúdo de uma página web, enquanto o CSS (Cascading Style Sheets) é uma linguagem de estilo que define a aparência e o layout dessa página.

O CSS não define a estrutura do conteúdo, mas sim a apresentação visual desse conteúdo, como cores, fontes, tamanhos, posicionamento dos elementos, entre outras propriedades.

> *As folhas de estilo em cascata CSS consistem em um mecanismo simples ~~para adicionar dados dinâmicos em documentos web~~, visíveis por todos os tipos de navegadores web.*

Errado. O CSS é uma linguagem de estilo utilizada para descrever a apresentação (layout, cores, fontes, etc.) de documentos HTML, XML e outros tipos de documentos. Ele permite separar o conteúdo da apresentação, tornando a web mais acessível, flexível e fácil de manter.

> *Nas CSS, o termo body significa que todo o código CSS existente entre um par de chaves — “{” e “}” — se aplica ao conteúdo dentro do elemento do HTML.*

> *Uma regra CSS tem sintaxe dividida em duas partes: um seletor e uma ou mais declarações. Cada declaração é dividida em propriedade e valor.*

> *A ordem de prioridade dos estilos definidos em CSS segue a seguinte cascata decrescente:*

1. Estilos em linha (inline) - Estilos definidos diretamente no elemento HTML usando o atributo "style".
2. Estilos internos - Estilos definidos dentro da seção do documento HTML.
3. Estilos externos - Estilos definidos em uma folha de estilo CSS externa, vinculada ao documento HTML.
4. Estilos padrão do navegador - Estilos definidos por padrão pelo navegador.

Essa ordem de prioridade é conhecida como "Cascata CSS" e determina qual estilo será aplicado quando houver conflitos entre diferentes definições.

> *Uma vantagem no uso de CSS é que, com ele, é possível fazer alteração no design sem afetar o código HTML.*

> *O código CSS é armazenado em um arquivo separado do HTML, embora seja utilizado para definir o modo como as tags HTML devem **ser apresentadas visualmente**."*

> *A forma correta de se referir a uma folha de estilo externa é `<link rel="stylesheet" href="estilo.css">`*

Em `<link rel="stylesheet" type="text/css" href="estilo.css">` o atributo `type` não é mais necessário na tag `<link>` para folhas de estilo CSS. A especificação HTML5 considera que o tipo padrão para folhas de estilo é "text/css", portanto, você pode omitir esse atributo.

> *Existem três maneiras de aplicar estilos em um documento HTML:*

1. **Folha de estilo externa**: Utiliza a tag `<link>` na seção `<head>` do documento HTML.
2. **Estilo interno**: Utiliza a tag `<style>` na seção `<head>` do documento HTML.
3. **Estilo inline**: Utiliza o atributo `style` diretamente em uma tag HTML.

> *CSS é um código separado da HTML que pode afetar a aparência das tags em uma única página ou em todo um site da Web.*

O CSS pode afetar a aparência de tags HTML em uma única página web ou em todo um site da web, dependendo de como as regras CSS são definidas e aplicadas.

> *O uso extensivo de pseudoelementos no CSS3 pode melhorar a estrutura semântica de um documento HTML.*

Errado. Os **pseudoelementos** do CSS3 não têm a capacidade de melhorar a estrutura semântica de um documento HTML. Eles são utilizados para estilizar partes específicas de um elemento HTML, como o primeiro caractere de um parágrafo ( `::first-letter` ) ou a primeira linha de um parágrafo ( `::first-lin` e), por exemplo.

A estrutura semântica de um documento HTML é definida pelos elementos HTML utilizados, como `<header>, <nav>, <article>, <section>, <footer>`, entre outros. Esses elementos fornecem significado e contexto ao conteúdo da página, independentemente da sua aparência visual.

> *A propriedade `font-size` controla o tamanho do texto em CSS.*

A propriedade `font-size` é uma das propriedades CSS mais importantes e amplamente utilizadas para definir o tamanho da fonte de um elemento HTML. Ela permite que você especifique o tamanho do texto em diferentes unidades, como pixels, pontos,

porcentagem, entre outras.

> *Em CSS, comentários são inseridos como na seguinte forma.* `/* comentário */`

Os comentários em CSS são delimitados pelos caracteres `/*` no início e `*/` no final. Tudo o que estiver entre esses delimitadores será ignorado pelo navegador e não afetará o estilo da página. Serve para uma ou mais linhas.

> *De acordo com o conceito de herança em CSS, as propriedades de página com especificidade mais alta têm prioridade sobre as propriedades com especificidade mais baixa.*

Correto. Isso significa que se houver conflito entre duas regras CSS, a regra com maior especificidade será aplicada.

> *Utilizando-se o CSS3, é possível criar efeitos animados por meio do atributo transform em conjunto com scripts.*

Certo. O atributo `transform` do CSS3 permite aplicar transformações geométricas aos elementos HTML, como rotação, escala, translação e inclinação. Quando combinado com propriedades de transição e animação do CSS3, é possível criar efeitos de animação sem a necessidade de scripts.

Além disso, é possível utilizar scripts, como JavaScript, para controlar e manipular as transformações CSS de forma mais avançada, permitindo criar animações ainda mais complexas e interativas.

> *O código HTML/CSS a seguir apresenta Alo Mundo como resultado.*

```
<html>
  <p style="text-transform:capitalize"> alo mundo </p>
</html>
```

Correto. A propriedade `text-transform:capitalize` do CSS faz com que a primeira letra de cada palavra seja convertida para maiúscula, enquanto as demais letras permanecem em minúscula.

> *No código abaixo, escrito na linguagem CSS, red é um valor do tipo palavra-chave, enquanto #f00 é um valor do tipo notação funcional.*

```
p {
  color: red;
```

```
  background-color: #f00;
}
```

Errado. No código CSS apresentado, ambos os valores `red` e `#f00` são considerados valores do tipo palavra-chave.

`red` é um valor do tipo palavra-chave, que representa a cor vermelha. `#f00` é um valor do tipo notação hexadecimal, que também representa a cor vermelha.

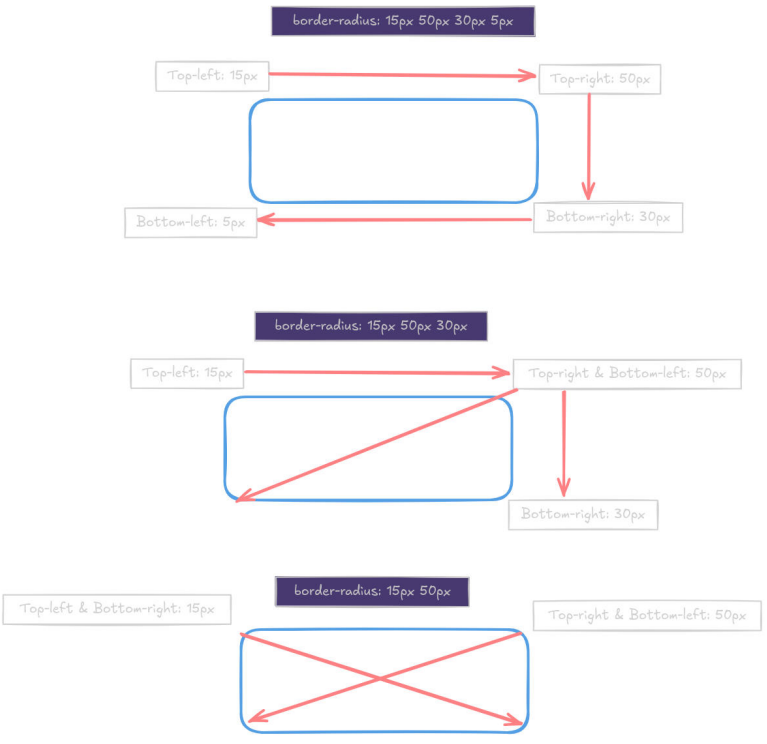
> *Quando se utiliza CSS3, para que os flex items sejam apresentados na mesma ordem em que aparecem no HTML, é necessário atribuir o valor 1 à propriedade order.*

Errado. A propriedade `order` do CSS Flexbox permite controlar a ordem de exibição dos flex items, mas o valor padrão dessa propriedade é 0. Ou seja, se nenhum valor for atribuído à propriedade `order`, os flex items serão exibidos na mesma ordem em que aparecem no HTML.

Se você atribuir o valor 1 à propriedade `order` de um flex item, esse item será exibido após os flex items com valor de `order` igual a 0 (o valor padrão). Isso fará com que o flex item seja exibido em uma ordem diferente da ordem em que aparece no HTML.

> *Ao utilizar CSS incorporado, vinculado ou inline, é possível atribuir a várias propriedades, em especial àquelas relacionadas ao dimensionamento das tags renderizadas, até quatro valores numéricos, ou ainda, utilizar quatro variações da propriedade, especificando-se a lateral ou extremidade a ser manipulada.*

A questão aborda mais especificamente a propriedade `border-radius`:



```
<div style="border:1px solid black;width:200px;height:150px;">
<!-- Parágrafos aqui...-->
</div>
```

Se no interior do contêiner a quantidade de texto exceder a área visível, para que apareça automaticamente uma barra de rolagem do lado direito do contêiner deve ser adicionada às configurações CSS a instrução `overflow:auto`.

A propriedade `overflow` em CSS controla o que acontece quando o conteúdo de um elemento excede o tamanho do seu contêiner. As opções mais comuns para `overflow` são:

- `overflow: auto;` : Adiciona uma barra de rolagem apenas quando necessário (ou seja, quando o conteúdo excede o tamanho do contêiner).
- `overflow: scroll;` : Sempre mostra uma barra de rolagem, independentemente de o conteúdo exceder ou não o tamanho do contêiner.
- `overflow: hidden;` : Oculta o conteúdo que excede o tamanho do contêiner.
- `overflow: visible;` : O conteúdo que excede o tamanho do contêiner é exibido fora dele.

```
<style>
div#block, div.box{ width:200px;height:100px;background-color:#336699}
</style>
```

As palavras block e box são, respectivamente, valores dos atributos id e class de tags div.

- `div#block` refere-se a um elemento `<div>` que possui o atributo `id` com o valor "block".
- `div.box` refere-se a um elemento `<div>` que possui a classe `class` com o valor "box".

> *No desenvolvimento web, **Sass** é uma extensão do CSS que acrescenta, entre outros, regras aninhadas, variáveis e mixins.*

Sass (Syntactically Awesome Style Sheets) é uma linguagem de extensão para CSS que permite o uso de recursos como variáveis, aninhamento de regras, mixins, funções e muito mais, facilitando a escrita e a manutenção de estilos em projetos web.

- As variáveis em Sass permitem que você armazene valores que podem ser reutilizados em todo o seu código. Isso é útil para manter a consistência e facilitar a manutenção.
  - `$primary-color: #3498db;`
- O aninhamento de regras permite que você escreva CSS de forma hierárquica,

refletindo a estrutura HTML. Isso torna o código mais legível e organizado, pois você pode aninhar seletores dentro de outros seletores.

```
.nav {
    ul {
        list-style: none;
    }
    li {
        display: inline-block;
    }
    a {
        text-decoration: none;
    }
}
```

- Mixins são blocos de código que podem ser reutilizados em diferentes partes do seu CSS. Eles permitem que você agrupe estilos que podem ser aplicados a vários seletores, e você pode até passar argumentos para personalizar o comportamento do mixin.

```
@mixin border-radius($radius) {
    border-radius: $radius;
    -webkit-border-radius: $radius; // Para compatibilidade com navegadores
antigos
}
.button {
    @include border-radius(5px);
}
```

- As funções em Sass permitem que você crie lógica e manipule valores. Você pode usar funções embutidas do Sass para realizar operações, como cálculos, ou criar suas próprias funções para retornar valores personalizados.

```
@function calculate-rem($pixels) {
    @return $pixels / 16 * 1rem;
}
body {
    font-size: calculate-rem(16px);
}
```

# JavaScript

> *O objetivo do JavaScript é deixar mais dinâmicas as aplicações web, de maneira que o usuário possa interagir e alterar o conteúdo da página.*

> *Para garantir uma correta compilação de um código escrito em JavaScript, é necessário que as variáveis sejam definidas e inicializadas antes de seu uso no código.*

Errado. Em JavaScript, não é necessário que as variáveis sejam definidas e inicializadas antes de seu uso no código. O JavaScript possui um mecanismo chamado "hoisting" que permite que as declarações de variáveis (mas não suas atribuições) sejam movidas para o topo do escopo antes da execução do código.

Isso significa que, mesmo que uma variável seja usada antes de ser declarada, o JavaScript irá reconhecê-la e atribuir a ela o valor `undefined` até que a declaração seja encontrada.

> *Em JavaScript, variáveis declaradas com `let` podem ser usadas fora do escopo em que foram criadas.*

Errado. Em JavaScript, variáveis declaradas com `let` têm um escopo de bloco, o que significa que elas são limitadas ao bloco (delimitado por chaves `{}`) em que foram declaradas.

Isso significa que uma variável declarada com `let` dentro de um bloco (por exemplo, dentro de um `if` ou de um `for`) não pode ser acessada fora desse bloco.

> *Em JavaScript, os valores Infinity e -Infinity são usados para representar os infinitos positivo e negativo.*

Correto. Esses valores especiais são usados para representar resultados de operações matemáticas que excedem os limites numéricos finitos do sistema.

> *Em js quando declarado, o evento `onchange` ocorre quando o valor de um elemento é alterado.*

Sim, isso está correto. O evento `onchange` em JavaScript é acionado quando o valor de um elemento é alterado.

Especificamente, o evento `onchange` é disparado quando:

1. O valor de um elemento de formulário (como `<input>`, `<select>` ou `<textarea>`) é

alterado e o elemento perde o foco (ou seja, o usuário sai do elemento).
2. O usuário seleciona um novo arquivo em um elemento `<input type="file">`.
3. O usuário seleciona uma nova opção em um elemento `<select>`.

> *Em js o valor `NaN` pode resultar de uma operação numérica que não resulte em um número preciso e significativo.*

O valor `NaN` é um valor especial em JavaScript que representa um valor numérico inválido. Exemplo: `console.log(5 * "hello");` Ele é um tipo de dado primitivo e pode ser verificado usando a função `isNaN()`.

O valor `NaN` é um valor especial em JavaScript que representa um valor numérico inválido. Ele é um tipo de dado primitivo e pode ser verificado usando a função `isNaN()`.

> `===` *(igualdade estrita) é o operador que expressa igualdade entre valores de mesmo tipo, enquanto que* `==` *(igualdade fraca) é um operador de comparação que realiza uma conversão de tipo implícita antes de fazer a comparação.*

- `2 == 2` Resultado: `true`
- `0 === " "` Resultado: `false` pois neste caso, `0` é um número, enquanto `" "` é uma string vazia.
- `1 == "1"` Resultado: `true`, pois o operador `==` realiza uma conversão de tipo antes da comparação.
- `0 === [ ]` Resultado: `false`
- `1000 == "1000"` Resultado: `true`

> *`var frutas = ["maçã", "laranja", "pera", "banana", "kiwi", "abacaxi", "manga", "uva"]; var posicao = frutas.indexOf("banana");` Considerando o código precedente, escrito em JavaScript, o valor da variável `posicao` após a execução do código é `3`.*

```
const array = ["Um", "Dois", "Tres"];
array.unshift("Tres");
array.splice(-1);
```

O valor final do array é:

1. `array.unshift("Tres");` Essa operação adiciona o elemento "Tres" no início do array (para inserir no final é `push()`), resultando em: `["Tres", "Um", "Dois", "Tres"]`
2. `array.splice(-1);` Essa operação remove o último elemento do array, resultando em: `["Tres", "Um", "Dois"]` Utilizar o método `pop()` teria o mesmo resultado de

`splice(-1)`

O código JavaScript a seguir, ao ser executado, apresenta o valor 3 no console de desenvolvimento do navegador web.

```
<script>
var cores = [“vermelho”, “verde”, “azul”];
console.log(cores.length);
</script>
```

A propriedade `length` de um array retorna o número de elementos que o array contém. Nesse caso, o array `cores` possui 3 elementos, portanto, o valor impresso no console será `3`.

> *A função `setTimeout` é usada para se aguardar, por um período de tempo parametrizado, a execução de uma thread com sucesso e, caso o tempo expire, ~~a função interrompe a execução do programa e gera um código de erro~~.*

Errado. A frase correta seria: A função setTimeout é usada para agendar a execução de uma função callback após um período de tempo parametrizado. Ela não interrompe a execução do programa principal, que continua a ser executado de forma assíncrona enquanto aguarda a execução da função callback.

Aqui está um exemplo em JavaScript que ilustra como a função setTimeout não interrompe a execução do programa principal:

```
console.log('Início do programa');
setTimeout(function() {
  console.log('Função callback executada após 2 segundos');
}, 2000);
console.log('Continuação da execução do programa principal');
// Saída:
// Início do programa
// Continuação da execução do programa principal
// Função callback executada após 2 segundos
```

Neste exemplo, a função `setTimeout` é chamada para executar uma função callback após 2 segundos (2000 milissegundos). No entanto, a execução do programa principal não é interrompida. Ele continua a ser executado normalmente, imprimindo a mensagem "Continuação da execução do programa principal" antes da função callback ser executada.

Isso demonstra que a função `setTimeout` é assíncrona e não bloqueia a execução do

programa principal, permitindo que o programa continue a ser executado enquanto aguarda a execução da função callback.

> *Suponha-se que um código precise tratar uma string que apresente o valor `10.5` como um número inteiro. Nessa hipótese, para suportar essa operação, deve-se usar a função `parseInt`.*

> *Em Javascript, a função `top (5,7)` é usada para encontrar o maior valor entre 5 e 7.*

Errado. Em JavaScript, não existe uma função nativa chamada `top()` que retorna o maior valor entre dois números. A função em JavaScript que retorna o maior valor entre dois (ou mais) números é a função `Math.max()`. Essa função recebe um ou mais argumentos numéricos e retorna o maior valor entre eles.

> *JavaScript trabalha com números usando os operadores aritméticos fornecidos pela própria linguagem. No entanto, a linguagem aceita operações matemáticas mais complexas por meio de um conjunto de funções e constantes definidas como propriedades do objeto Math.*

Assinale a opção que apresenta a função que permite realizar, em JavaScript, a operação matemática de arredondar para cima o número 1,17, obtendo-se o valor 2.

Aqui está uma breve explicação sobre cada uma das funções matemáticas do JavaScript mencionadas:

A. `Math.ceil()`:

- Essa função arredonda um número para cima, retornando o menor número inteiro maior ou igual ao número fornecido.
- Exemplo: `Math.ceil(3.2)` retorna `4`.

C. `Math.pow()`:

- Essa função retorna o resultado da elevação de um número à potência de outro número.
- Exemplo: `Math.pow(2, 3)` retorna `8`.

D. `Math.round()`:

- Essa função arredonda um número para o inteiro mais próximo.
- Exemplo: `Math.round(3.5)` retorna `4`.

E. `Math.sqrt()`:

- Essa função retorna a raiz quadrada de um número.
- Exemplo: `Math.sqrt(16)` retorna `4`.

Essas funções matemáticas do objeto `Math` em JavaScript são úteis para realizar operações matemáticas comuns em programas e aplicações.

> *O código Javascript a seguir altera a palavra Pergunta para Resposta quando o evento onclick é disparado.*

```
<html>
<body>
    <p id='q1'>Pergunta</p>
    <input type='button'
          value='clique'
          onclick='javascript:document.getElementById("q1").innerHTML =
"Resposta";'>
</body>
</html>
```

Correto. Essa prática é conhecida como "event handler inline", onde o código JavaScript é diretamente inserido no atributo `onclick` do elemento HTML. Essa abordagem não é considerada a melhor prática, pois mistura o conteúdo (HTML) com a lógica (JavaScript), tornando o código menos modular e mais difícil de manter. A abordagem mais recomendada atualmente é separar o HTML do JavaScript, utilizando um evento de clique no elemento e, então, manipular o DOM (Document Object Model) através de um manipulador de eventos em um arquivo JavaScript externo.

```
let s = “MPGO”;
let resultado = s.padStart(7, “*”);
console.log(resultado);
```

O que é exibido na tela é `***MPGO`

Certo. A função `padStart()` adiciona caracteres (no caso, o caractere " *") ao início da string, até que ela atinja o tamanho especificado (no caso, 7 caracteres).* A string original "MPGO" possui 4 caracteres, e a função `padStart(7, "*")` adiciona 3 caracteres `*` ao início da string, resultando em uma string final com 7 caracteres.

> *Em JavaScript, a função `eval` interpreta e executa strings com código JavaScript embutido.*

Certo. A função `eval()` em JavaScript é responsável por interpretar e executar código JavaScript embutido em uma string. Quando você chama `eval(string)`, o JavaScript irá

avaliar a string fornecida como se fosse código JavaScript e executá-la.

Exemplo:

```
let x = 5;
let expression = "x + 10";
let result = eval(expression);
console.log(result); // Output: 15
```

Neste exemplo, a string `"x + 10"` é avaliada e executada pela função `eval()`, resultando no valor `15`.

No entanto, é importante ressaltar que o uso indiscriminado de `eval()` pode ser perigoso, pois pode abrir brechas de segurança se o código a ser executado não for devidamente validado. Portanto, é recomendado evitar o uso de `eval()` sempre que possível e, em vez disso, utilizar alternativas mais seguras, como funções de primeira classe ou métodos de objetos.

# AJAX

AJAX (Asynchronous JavaScript and XML) é uma técnica de desenvolvimento web que permite a comunicação assíncrona entre o cliente e o servidor, possibilitando que o usuário interaja com a interface enquanto as requisições ao servidor são processadas em segundo plano. Utilizando JavaScript, HTML e o Document Object Model (DOM), AJAX permite atualizações dinâmicas na página, como troca de imagens, sem recarregar a interface.

> *Utilizando-se o Ajax, qualquer ação que requeira uma resposta do servidor ocorre de forma assíncrona.*
>
> *A tecnologia Ajax separa as interações com o usuário das interações com o servidor, de modo que ambas sejam executadas paralelamente. Isso significa que o usuário pode continuar interagindo com a interface da aplicação enquanto as requisições ao servidor estão sendo processadas, resultando em uma experiência mais fluida e responsiva.*
>
> *Utilizado com o Javascript, o AJAX proporciona novos usos a padrões já existentes e tem a capacidade de liberar o navegador web para outras operações enquanto atende outras requisições.*
>
> *A metodologia Ajax para desenvolvimento web engloba os componentes HTML, document object model e JavaScript.*

> *Nas aplicações SPA (single page application) que utilizam AJAX, ~~cada interação do usuário resulta em um recarregamento completo da página~~, o que garante que todas as partes da interface sejam atualizadas simultaneamente.*

Errado. Em aplicações SPA (Single Page Application) que utilizam AJAX, apenas partes específicas da interface são atualizadas, permitindo uma experiência mais fluida e rápida, sem a necessidade de recarregar toda a página. Assim, a estrutura da página fica estática, enquanto o conteúdo principal é atualizado de forma assíncrona.

> *AJAX representa técnicas utilizadas para que páginas web sejam carregadas rapidamente pelo ~~processamento na parte cliente da aplicação~~.*

Errado. AJAX não é usado especificamente para processamento no lado do cliente, mas sim para atualização assíncrona de conteúdo.

> *A ferramenta Ajax ~~permite o desenvolvimento de aplicações juntamente com as interações com o usuário e o servidor, o que faz as duas avançarem juntas~~.*

Errado. Embora a ferramenta Ajax permita o desenvolvimento de aplicações que interagem tanto com o usuário quanto com o servidor, a principal característica do Ajax é que ele separa essas interações, permitindo que elas ocorram de forma assíncrona. Isso significa que as interações com o usuário podem continuar enquanto as requisições ao servidor estão sendo processadas, em vez de avançarem juntas de forma sincronizada.

> *Em um site que utilize Ajax para enviar ao servidor dados da senha de cartão de crédito via protocolo HTTP, é possível que pessoa mal-intencionada intercepte os dados transmitidos.*

Sim, pois  HTTP é sem criptografia, sendo necessários usar a versão criptografada: HTTPS

# ## XMLHttpRequest

`XMLHttpRequest` é um objeto JavaScript que permite a comunicação assíncrona entre um cliente (geralmente um navegador web) e um servidor. Ele é uma parte fundamental da tecnologia AJAX (Asynchronous JavaScript and XML), que possibilita a atualização de partes de uma página web sem a necessidade de recarregar a página inteira.

Estes são os principais métodos de `XMLHttpRequest`:

- `open` inicializa uma nova requisição.
- `send` envia a requisição para o servidor.
- `abort` cancela a requisição atual, se estiver em andamento.

Estes são os principais eventos de `XMLHttpRequest`:

- `onreadystatechange` é acionado sempre que o estado da requisição muda (ou seja, sempre que a propriedade `readyState` é alterada).
- `onload` é acionado quando a requisição é concluída com sucesso
- `onerror` é acionado quando ocorre um erro durante a requisição, como problemas de rede ou falhas de conexão.
- `ontimeout` é acionado quando a requisição excede o tempo limite definido.
- `onabort` é acionado quando a requisição é cancelada usando o método `abort()`

Exemplo:

Plain text

```
// Cria uma nova instância do objeto XMLHttpRequest
var xhr = new XMLHttpRequest();
// Define a função a ser chamada quando a requisição for concluída
  xhr.onreadystatechange = function() {
    // Verifica se a requisição foi concluída (readyState 4) e se a
resposta foi bem-sucedida (status 200)
```

```
    if (xhr.readyState === 4 && xhr.status === 200) {
        // Captura a resposta do servidor
        var response = xhr.responseText;
        // Exibe a resposta no console
        console.log(response);
    }
};
// Inicializa a requisição: método GET, URL do recurso, e define como
assíncrona (true)
xhr.open("GET", "https://api.exemplo.com/dados", true);
// Envia a requisição
xhr.send();
```

> *Se o Ajax for usado para atualizar determinada imagem em uma página de compras, o objeto utilizado para a solicitação Ajax, na programação do evento de clique, será o XmlHttpRequest.*
>
> *`XMLHttpRequest` é uma API utilizada pela tecnologia de apresentação Ajax e fornece a recuperação de dados de uma URL, em qualquer formato textual, sem que seja necessário atualizar completamente a página apresentada ao usuário.*

> *~~Da mesma forma que as páginas HTML utilizam os objetos~~ `HttpRequest` e `HttpResponse` ~~para enviar e receber solicitações ao servidor,~~ o Ajax utiliza os objetos `XMLHttpRequest` e `XMLHttpResponse` para comunicação assíncrona.*

Errado. O Ajax utiliza o objeto `XMLHttpRequest` para enviar solicitações assíncronas ao servidor e receber respostas, mas não existe um objeto chamado `XMLHttpResponse`. A resposta do servidor é acessada através do objeto `XMLHttpRequest` após a solicitação ser concluída.

> *O Ajax — que carrega e renderiza uma página utilizando recursos de scripts que estão rodando pelo lado do cliente (navegador) — utiliza um objeto `XMLHttpRequest` e o método `open` para abrir um documento em uma linguagem de marcação, bem como para passar argumentos e capturar uma resposta.*

# Webservices

Web services são sistemas de software projetados para suportar interoperabilidade entre diferentes aplicações através da internet. Os protocolos **SOAP** e **REST** são os padrões mais utilizados na comunicação entre os sistemas por meio do *web service*, eles compõem a estrutura básica dos *web services*. A principal diferença entre eles é que o SOAP é um protocolo mais rígido e formal, enquanto o REST é um estilo arquitetural mais flexível e orientado a recursos.

> *Uma das vantagens do **REST** é a sua utilização correta dos métodos HTTP (GET, POST, PUT, DELETE) para realizar as operações de leitura, criação, atualização e exclusão de recursos, enquanto o **SOAP** utiliza o protocolo HTTP, mas define sua própria estrutura de mensagem (em XML), independente dos métodos HTTP.*

> *Web service é um modelo de computação distribuída que ~~depende dos sistemas operacionais~~, porém independe da linguagem de desenvolvimento e do hardware onde os sistemas integrados são processados.*

Errado. Web services são um modelo de computação distribuída que não dependem dos sistemas operacionais, mas sim de protocolos de comunicação padrão da Internet, como HTTP, SOAP, XML, etc. Eles permitem a integração de sistemas independentemente da linguagem de programação, sistema operacional ou hardware utilizado.

> *A interface de um webservice pode mudar ao longo do tempo sem comprometer a habilidade de interação do cliente com o serviço. A essa caraterística dá-se o nome de **acoplamento fraco.***
>
> *Para alcançar a interoperabilidade entre aplicações, é necessário adotar padrões abertos e comuns, tais quais serviços web, REST, JSON, XML, OAuth e OpenID Connect.*

## ## APIs RESTful

APIs RESTful são um tipo específico de web service que segue os princípios da arquitetura REST (Representational State Transfer). São um subconjunto dos web services, pois elas também são sistemas de software projetados para integração entre aplicações. Utilizam os métodos HTTP (GET, POST, PUT, DELETE, etc.) para realizar operações sobre recursos identificados por URIs.

No padrão REST (Representational State Transfer), cada **recurso** é identificado por uma URL única. Em um serviço RESTful, os recursos são identificados pela mesma URL, a diferença entre os métodos (como GET, POST, PUT, DELETE) é o que determina a ação a ser realizada sobre esse recurso.

> *REST (REpresentational State Transfer) ~~é um protocolo para troca de informações estruturadas~~, cujo formato de mensagem é embasado na linguagem de marcação extensível (XML).*

Errado. REST (REpresentational State Transfer) não é um protocolo, mas sim um estilo arquitetural para a construção de serviços web.

> *Uma application programming interface (API) define as regras que necessitam ser seguidas para se comunicar com outros sistemas de software, podendo ser utilizada para compartilhar recursos e fornecer serviços da Web.*
>
> *Uma característica dos serviços Web RESTful é a capacidade de transmitir dados diretamente via HTTP.*
>
> *As APIs de Web services que aderem às restrições de arquitetura REST são chamadas de APIs RESTful.*

> *Na comunicação feita entre cliente e servidor, uma API Restful ~~armazena informações de outras sessões a respeito das solicitações~~, de modo a fazer um controle confiável das suas transações.*

Errado. As APIs RESTful não armazenam informações de outras sessões a respeito das solicitações. Elas são projetadas para serem stateless, ou seja, cada solicitação é tratada de forma independente, sem manter informações sobre sessões anteriores.

> *Na representational state transfer (REST), a ausência de estado refere-se ao método de comunicação, no qual o servidor completa cada solicitação do cliente, independentemente de todas as solicitações anteriores.*
>
> *Em uma API RESTful, cada solicitação deve conter todos os dados necessários ao seu atendimento para não depender de informações armazenadas em outras sessões, o que caracteriza uma restrição de comunicação stateless.*

> *Uma API REST utiliza ~~somente o formato XML~~ para representar os recursos e as respostas do servidor.*

Errado. Uma API REST não é limitada a um único formato de representação de recursos. Embora o XML possa ser utilizado, as APIs REST geralmente suportam múltiplos formatos, como JSON, XML, HTML, e outros. O JSON, por exemplo, é um formato muito popular devido à sua simplicidade e leveza.

> *As APIs do tipo REST, também chamadas de APIs RESTful, utilizam o protocolo HTTP e oferecem suporte à criptografia TLS (Transport Layer Security), mantendo a privacidade*

> *em uma conexão HTTPS na Internet.*

## ### Métodos/Verbos HTTP

No contexto de web services REST, os verbos ou métodos HTTP são utilizados para indicar a ação que o cliente deseja realizar sobre um recurso. Cada verbo tem um significado específico e é associado a uma operação CRUD (Create, Read, Update, Delete). Aqui estão os verbos HTTP mais importantes e suas funções:

| **Método HTTP** | **Descrição** |
|---|---|
| GET | Recupera dados de um recurso. Não deve alterar o estado do recurso. |
| POST | Cria um novo recurso. Envia dados ao servidor para processamento. |
| PUT | Atualiza um recurso existente ou cria um novo recurso se ele não existir. |
| DELETE | Remove um recurso existente. |
| PATCH | Aplica modificações parciais a um recurso existente. |
| HEAD | Recupera os cabeçalhos de resposta de um recurso, sem o corpo da resposta. |

> *O REST emprega um protocolo universal, o HTTP, para oferecer um serviço web simples e aberto. Verbos HTTP são usados para realizar chamadas e indicar para o serviço que ação deve ser realizada. O verbo* `PUT` *é usado tipicamente para a atualização de um recurso existente.*

Um método é considerado **idempotente** se múltiplas chamadas ao mesmo método com os mesmos parâmetros resultam no mesmo estado do recurso. Por exemplo, chamar um método GET várias vezes não altera o recurso, enquanto chamar um método POST várias vezes pode criar múltiplos recursos.

## ### HATEOAS

Em uma API REST que segue o princípio de HATEOAS (Hypermedia as the Engine of Application State), as respostas enviadas ao cliente contêm não apenas os dados solicitados, mas também informações sobre os links (hyperlinks) que o cliente pode utilizar para navegar e interagir com os recursos da API, que indicam as próximas ações que o cliente pode realizar. Isso permite que o cliente descubra dinamicamente como interagir com a API, sem precisar ter conhecimento prévio de sua estrutura.

> *O princípio HATEOAS é utilizado pela API REST para fornecer links entre os recursos e facilitar a navegação do cliente.*

## SOAP

SOAP é um dos principais protocolos utilizados para a implementação de web services. Ele fornece uma estrutura padrão para a definição de serviços, a descrição de suas interfaces e a troca de mensagens entre clientes e servidores. Ele define um formato de mensagem baseado em XML para a comunicação entre aplicações.

> *SOAP (simple object access protocol) é um protocolo de comunicação usado para a troca de mensagens XML entre o cliente e o provedor de serviço.*
>
> *O **protocolo** utilizado para efetuar chamadas de procedimentos remotos e que é independente de plataforma é o SOAP.*
>
> *APIs do tipo SOAP utilizam protocolo com estrutura de mensagem restrita que depende de XML.*

Assim, o XML é a base fundamental para a estruturação das mensagens SOAP. O código abaixo está escrito em XML:

```
<Cliente Nome="JOAO DA SILVA" Identificacao="123456">
    <Endereco Logradouro="Rua Ativa, 123" Municipio="Recife" UF="PE" />
</Cliente>
```

### Mensagem SOAP

A mensagem SOAP serve para permitir a comunicação e integração entre diferentes sistemas e aplicações através da internet, por meio de web services.

> *As solicitações e respostas empacotadas em mensagens SOAP (Simple Object Access Protocol) são escritas em XML e contêm as informações para a execução de um web service.*

Ele possuí os seguintes elementos:

| Elemento | Descrição | Indispensável |
|---|---|---|
| `<?xml version="1.0" encoding="UTF-8"?>` | Declaração XML, obrigatória para definir a versão e a codificação do documento. | Sim |
| `<soap:Envelope>` | Elemento raiz que encapsula toda a mensagem SOAP. | Sim |
| `<soap:Header>` | Contém informações adicionais sobre a mensagem. | Não |

| Elemento | Descrição | Indispensável |
|---|---|---|
| `<soap:Body>` | Contém a definição da operação a ser executada. | Sim |
| `<soap:Fault>` | Contém informações sobre erros ou exceções. | Não |
| `<faultcode>` | Código de falha que indica o tipo de erro. | Sim |
| `<faultstring>` | Mensagem de falha que descreve o erro. | Sim |
| `<detail>` | Elemento que pode conter informações adicionais sobre o erro. | Não |
| `<errorMessage>` | Mensagem de erro que fornece detalhes sobre o problema. | Não |

> *O elemento header (cabeçalho) do envelope de uma mensagem enviada pelo SOAP é ~~um elemento indispensável~~, uma vez que ele ~~informa o endereço da mensagem~~.*

Errado. O elemento "header" (cabeçalho) do envelope de uma mensagem SOAP é um elemento dispensável e contêm informações adicionais que podem ser necessárias para o processamento da mensagem, como informações de autenticação, informações de roteamento, ou outros metadados que podem ser relevantes para a comunicação entre o cliente e o servidor.

> *No que se refere a web services, as duas formas de envio de mensagem para que um cliente possa efetuar solicitações a um web service são one-waymessaging e request-response messaging.*

- *one-way messaging* (envio de mensagem sem esperar resposta)
- *request-response messaging* (envio de mensagem e espera pela resposta).

O WSDL (Web Services Description Language) é uma linguagem baseada em XML para descrever web services.

UDDI (Universal Description, Discovery and Integration) é um padrão que permite o registro e a descoberta de serviços web. O UDDI atua como um catálogo central onde os provedores de serviços web baseados em SOAP podem registrar e publicar informações sobre seus serviços. Isso facilita a descoberta e o consumo desses serviços por outras aplicações.

Os registros UDDI podem estar disponíveis de forma:

- **pública**, quando qualquer pessoa ou aplicação possa descobrir e acessar os serviços web registrados
- **privada**, disponíveis apenas dentro de uma organização ou grupo restrito, permitindo o registro e descoberta de serviços web de forma controlada e segura.

# Noções sobre desenvolvimento e manutenção de sistemas e aplicações

Os **requisitos não funcionais** realmente estabelecem padrões sobre como o sistema deve se comportar, incluindo aspectos como desempenho, segurança, usabilidade, confiabilidade, entre outros. Podem estar relacionados às propriedades emergentes do sistema, como confiabilidade, desempenho e tempo de resposta.

- Requisitos não funcionais podem afetar a arquitetura de um sistema em vez de apenas componentes individuais. Por exemplo, para assegurar que sejam cumpridos os requisitos de desempenho, será necessário organizar o sistema para minimizar a comunicação entre os componentes.
- Um único requisito não funcional, tal como um requisito de proteção, pode gerar uma série de requisitos funcionais relacionados que definam os serviços necessários no novo sistema. Além disso, também podem gerar requisitos que restrinjam requisitos existentes.

Por outro lado, os **requisitos funcionais** definem o que o sistema deve fazer, ou seja, as funcionalidades e características específicas que o sistema deve oferecer aos usuários. Estão diretamente relacionados com os serviços específicos oferecidos pelo sistema a seus usuários.

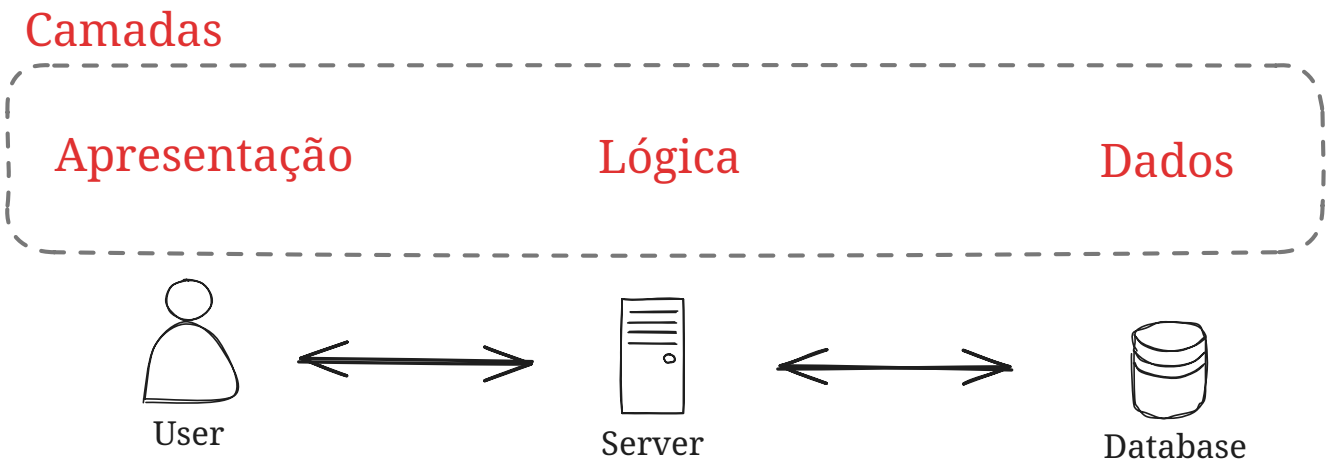
Aqui está uma comparação:

| **Aspecto** | **Requisitos Funcionais** | **Requisitos Não Funcionais** |
|---|---|---|
| **Definição** | Especificam o que o sistema deve fazer; descrevem funções e comportamentos. | Especificam como o sistema deve se comportar; descrevem atributos de qualidade. |
| **Exemplos** | O sistema deve permitir que os usuários façam login. | O sistema deve ser capaz de processar 1000 transações por segundo. |
| **Foco** | Funcionalidade e comportamento do sistema. | Qualidade, desempenho e restrições do sistema. |
| **Impacto na Arquitetura** | Geralmente afetam componentes individuais e suas interações. | Podem afetar a arquitetura do sistema como um todo. |
| **Medição** | Normalmente medidos em termos de "sim" ou "não" (ex: o sistema faz X?). | Medidos em termos de métricas (ex: tempo de resposta, taxa de erro). |

Sobre testes em desenvolvimento de software, geralmente seguem uma abordagem

que começa com testes de componentes individuais (testes unitários) antes de passar para testes de integração e, finalmente, testes do sistema como um todo. Essa abordagem permite identificar e corrigir problemas em partes específicas do sistema antes de integrá-las e testar a aplicação como um todo.

# Arquitetura cliente-servidor multicamadas




A arquitetura cliente-servidor multicamadas é geralmente organizada em três camadas principais: apresentação, lógica de negócios (ou lógica) e dados.

- A camada de **apresentação** é responsável pela interface com o usuário. O cliente (interface do usuário) interage com esta camada, que é responsável por exibir informações e coletar entradas do usuário.
- A camada de **lógica** de negócios processa as regras e a lógica da aplicação, realizando operações necessárias com base nas solicitações do cliente.
- A camada de **dados** gerencia o armazenamento e a recuperação de dados. Essa camada gerencia o armazenamento, a recuperação e a persistência dos dados.

Na arquitetura cliente/servidor multicamadas, cada camada se comunica diretamente apenas com a camada imediatamente adjacente (ou vizinha). Isso proporciona um maior isolamento entre as camadas, o que reduz a fragilidade da aplicação. Se uma camada precisar ser alterada ou atualizada, as mudanças podem ser feitas com um impacto mínimo nas outras camadas, desde que a interface de comunicação entre elas permaneça consistente. Essa separação de responsabilidades e a comunicação controlada entre as camadas contribuem para a robustez e a manutenibilidade da aplicação.

> *No modelo cliente-servidor de várias camadas, o cliente ~~realiza comunicação direta com o servidor de banco de dados~~, proporcionando eficiência no processamento das consultas.*

Errado, apresentação se comunica com a lógica que por fim se comunica com dados.

> *A arquitetura cliente/servidor multicamadas reduz a fragilidade da aplicação, fornecendo mais isolamento entre as camadas. Uma camada somente se comunica diretamente com a camada vizinha.*

> *Em arquitetura multicamadas, o servidor de aplicação nada mais é do que um programa que fica entre o aplicativo cliente e o sistema de gerenciamento de banco de dados.*

> *A arquitetura multicamada ~~mais comumente utilizada na web é a de quatro camadas (4-Tier)~~ , composta de: camada de dados; camada de rede; camada de negócio; e camada de apresentação.*

Errado. A arquitetura multicamada mais comumente utilizada na web é geralmente a de três camadas (3-Tier), que consiste em: camada de apresentação, camada de lógica de negócios e camada de dados.

> *Nessa arquitetura, quando são consideradas três camadas, ~~a primeira camada deve ser implementada por meio do servidor de aplicação.~~*

Errado. Na arquitetura cliente-servidor multicamadas com três camadas, a primeira camada, que é a camada de apresentação, é geralmente implementada no cliente (ou seja, na interface do usuário). O servidor de aplicação é responsável pela camada de lógica de negócios, que é a segunda camada.

> *Para a codificação dos módulos e componentes de uma aplicação computacional aderente a uma arquitetura do tipo cliente-servidor multicamadas, ~~deve-se utilizar uma única linguagem de programação~~, visto que isso facilita a futura manutenção desse tipo de aplicação.*

Errado. Embora utilizar uma única linguagem de programação possa facilitar a manutenção em alguns casos, em uma arquitetura do tipo cliente-servidor multicamadas, é comum e muitas vezes necessário utilizar diferentes linguagens de programação para diferentes camadas da aplicação.

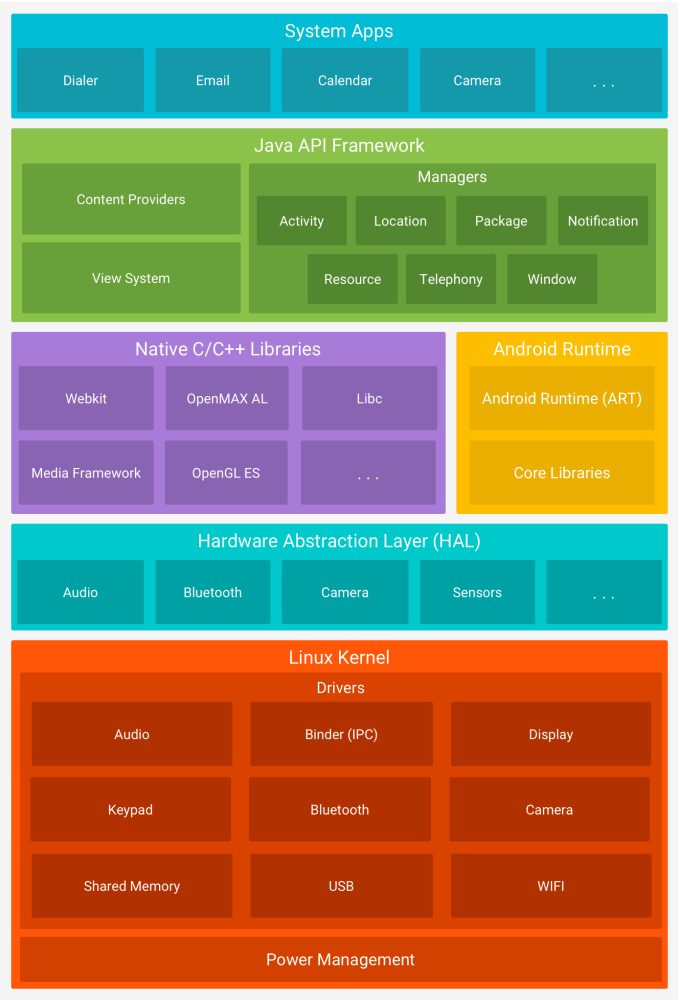
> *No desenvolvimento da interface com o usuário de uma aplicação do tipo cliente-servidor multicamadas, o conhecimento das necessidades de comunicação de dados do usuário ~~é apresentado com menor importância~~ que o conhecimento das necessidades de informações no ambiente de trabalho do usuário.*

No desenvolvimento da interface com o usuário de uma aplicação do tipo cliente-servidor multicamadas, o conhecimento das necessidades de comunicação de dados do usuário é fundamental. A interface deve ser projetada levando em consideração como os usuários interagem com os dados e quais informações são necessárias para realizar suas tarefas de forma eficiente. As necessidades de comunicação de dados são essenciais para garantir que a interface atenda às expectativas e requisitos dos usuários, facilitando a usabilidade e a eficácia da aplicação.

# Android

## ## Framework do Android

O **framework do Android** é um conjunto de bibliotecas e APIs que fornece as ferramentas necessárias para o desenvolvimento de aplicativos Android. Ele permite que os desenvolvedores acessem funcionalidades do sistema operacional, como gerenciamento de interface do usuário, acesso a dados, comunicação entre aplicativos e serviços do sistema. O framework é dividido em várias camadas:




Vamos imaginar um cenário em que um desenvolvedor está criando um aplicativo Android de jogo que utiliza gráficos avançados e interatividade. Aqui está como cada um dos componentes mencionados será utilizado na prática:

**Linux Kernel**: O desenvolvedor não interage diretamente com o Linux Kernel, mas ele fornece a base do sistema operacional Android, gerenciando recursos de hardware, como CPU e memória. O jogo se beneficiará do gerenciamento eficiente de processos e memória, permitindo que o aplicativo funcione suavemente em diferentes dispositivos.

**Hardware Abstraction Layer (HAL)** : O HAL permite que o jogo acesse recursos de hardware, como a GPU para renderização gráfica. O desenvolvedor pode usar APIs de gráficos que se comunicam com o HAL, garantindo que o jogo funcione em diferentes

dispositivos sem precisar se preocupar com as especificidades de cada hardware.

**Android Runtime (ART)** : O ART executa o código do jogo. O desenvolvedor escreve o código em Java (ou Kotlin), que é compilado para bytecode e, em seguida, convertido em código nativo pelo ART durante a instalação. Isso garante que o jogo tenha um desempenho otimizado e tempos de inicialização rápidos.

**Native C/C++ Libraries**: Para partes do jogo que exigem alto desempenho, como processamento de gráficos ou física, o desenvolvedor pode usar o Android NDK para escrever código em C/C++. Ele pode criar bibliotecas nativas que realizam cálculos intensivos e, em seguida, chamar essas bibliotecas a partir do código Java do jogo, combinando a eficiência do C/C++ com a facilidade de uso do Java.

**Java API Framework**: O desenvolvedor utiliza o Java API Framework para construir a interface do usuário do jogo, gerenciar eventos de toque e interações, e acessar recursos do sistema, como armazenamento para salvar o progresso do jogo. Ele pode usar classes como `Activity` para gerenciar a tela do jogo e `View` para desenhar elementos gráficos.

Resumo do Fluxo de Trabalho

- O **Linux Kernel** fornece a infraestrutura básica e gerenciamento de recursos.
- O **HAL** permite que o jogo acesse a GPU e outros componentes de hardware.
- O **ART** executa o código do jogo, garantindo desempenho otimizado.
- O **Native C/C++ Libraries** são usados para partes críticas de desempenho, como gráficos e física.
- O **Java API Framework** é utilizado para construir a interface do usuário e gerenciar interações.

# ## Activities

Uma **Activity** é um componente fundamental de um aplicativo Android que representa uma única tela com uma interface de usuário. Cada Activity é responsável por interagir com o usuário e pode conter elementos como botões, textos e imagens. As Activities são usadas para criar a experiência do usuário, permitindo que ele navegue entre diferentes telas do aplicativo. Cada Activity pode iniciar outras Activities, formando uma pilha de atividades que o usuário pode navegar.

## ### Activity Manager

O **Activity Manager** é um componente do framework do Android que gerencia o ciclo de vida das Activities e a navegação entre elas. Ele é responsável por iniciar, pausar, retomar e encerrar Activities, além de gerenciar a pilha de atividades em execução. O Activity Manager garante que as Activities sejam exibidas corretamente e que os recursos sejam gerenciados de forma eficiente, permitindo que os usuários alternem

entre diferentes partes do aplicativo sem problemas.

> *Na arquitetura Android, o Android Runtime ART e a camada de abstração de hardware HAL são implementados por código nativo que exige bibliotecas nativas escritas em C e C++.*

Certo. **** Na arquitetura Android, tanto o Android Runtime (ART) quanto a Hardware Abstraction Layer (HAL) são implementados em código nativo, que geralmente é escrito em C e C++. Isso permite que o sistema operacional interaja de forma eficiente com o hardware subjacente e execute aplicativos de maneira otimizada.

> *Com relação à plataforma Android, assinale a opção correta: Webkit é uma biblioteca redenrizadora de páginas para navegadores com suporte a DOM e AJAX.*

**Webkit** é um motor de renderização de código aberto utilizado para exibir conteúdo da web em navegadores, como o Safari. Ele é responsável por interpretar HTML, CSS e JavaScript, construindo a árvore DOM (Document Object Model) que representa a estrutura de uma página. Além disso, o Webkit suporta AJAX (Asynchronous JavaScript and XML), permitindo que páginas se atualizem de forma assíncrona, o que resulta em aplicações web mais responsivas e interativas. Em resumo, o Webkit é fundamental para a renderização eficiente e a interatividade em navegadores.

> *Dalvik é uma máquina virtual que executa aplicativos Android*

Dalvik é uma máquina virtual projetada para executar aplicativos Android, que são escritos em Java. Ela converte bytecode Java em um formato executável, sendo leve e eficiente em termos de uso de recursos, o que é essencial para dispositivos móveis. Dalvik também suporta multithreading, permitindo que aplicativos realizem várias operações simultaneamente. A partir do Android 5.0, Dalvik foi gradualmente substituída pelo Android Runtime (ART), que oferece melhorias em desempenho e eficiência. Em resumo, Dalvik é crucial para a execução eficiente de aplicativos no Android.

> *Dalvik foi substituído pelo Android Runtime (ART) para melhorar desempenho, inicialização de aplicativos, gerenciamento de memória e suporte a recursos modernos no Android.*

Dalvik foi substituído pelo Android Runtime (ART) a partir do Android 5.0 (Lollipop) devido a várias melhorias necessárias:

1. **Desempenho**: ART utiliza compilação antecipada (AOT), resultando em tempos de execução mais rápidos em comparação com a compilação just-in-time (JIT) do Dalvik.

2. **Inicialização Rápida**: Os aplicativos iniciam mais rapidamente em ART, pois o código já está pré-compilado.
3. **Gerenciamento de Memória**: ART possui um coletor de lixo mais eficiente, melhorando o uso da memória e reduzindo a fragmentação.
4. **Suporte a Recursos Modernos**: ART oferece melhor suporte para novas funcionalidades, incluindo a execução em dispositivos de 64 bits.
5. **Melhores Ferramentas de Depuração**: ART proporciona melhores ferramentas para depuração e profiling, facilitando a identificação de problemas de desempenho.

Em resumo, a substituição foi motivada pela necessidade de melhorar o desempenho, a eficiência e a experiência geral do usuário e desenvolvedor.

D - O Activity Manager não está na camada Libraries; ele faz parte do framework do Android e gerencia a execução de activities.

E - Content Providers não gerenciam apresentações de janelas; eles são usados para gerenciar o acesso a dados entre diferentes aplicativos.

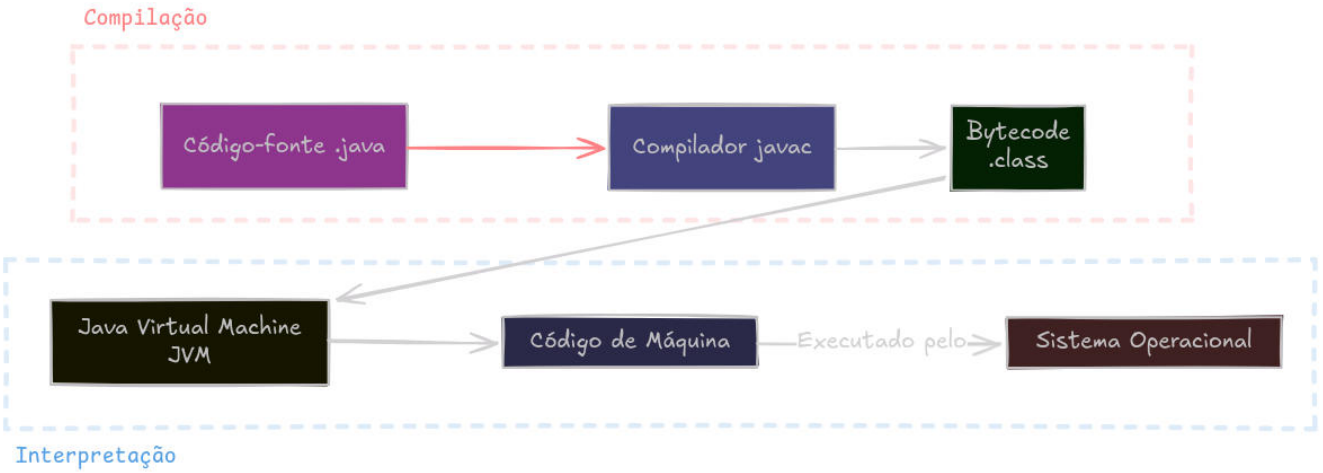
# Java

Em Java, o método principal que inicia a execução de um programa é o `main`. Esse método é o ponto de entrada para qualquer aplicação Java. A assinatura desse método é geralmente:

```
public static void main(String[] args) {
    // Código a ser executado
}
```

> *Os programas em Java podem ser tanto interpretados quanto compilados.*

A afirmação está correta. Os programas em Java podem ser tanto interpretados quanto compilados, e isso se deve à forma como a linguagem Java é projetada.




1. **Compilação**: O código-fonte Java (arquivos `.java`) é compilado pelo compilador Java (`javac`) em bytecode (arquivos `.class`). Esse bytecode é uma representação intermediária que não é específica de uma plataforma.
2. **Interpretação**: O bytecode gerado é então interpretado pela Java Virtual Machine (JVM). A JVM é responsável por executar o bytecode em tempo de execução, convertendo-o em código de máquina que pode ser executado pelo sistema operacional subjacente.

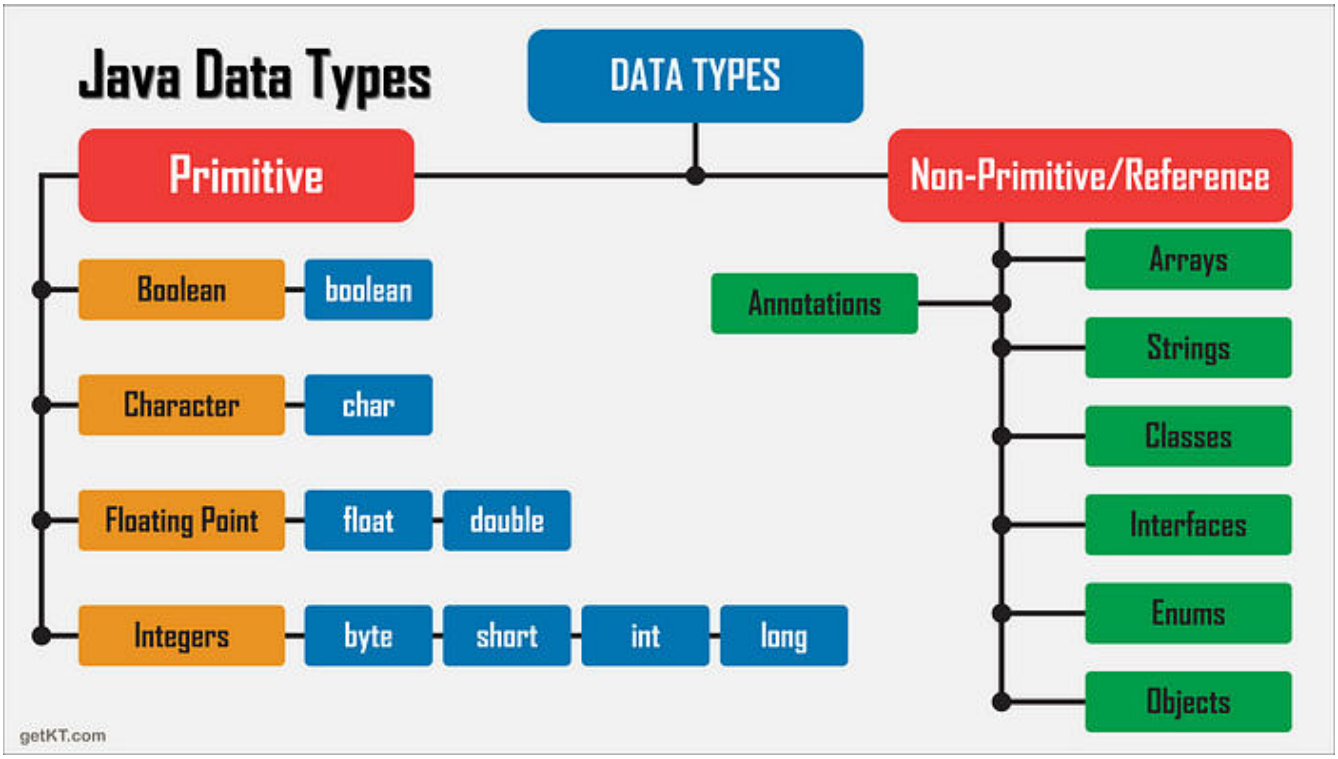
> *Na linguagem Java, existe um mecanismo próprio de gerenciamento de memória denominado garbage collector.*

Certo. Na linguagem Java, existe um mecanismo de gerenciamento de memória chamado **garbage collector** (coletor de lixo). O garbage collector é responsável por identificar e liberar a memória ocupada por objetos que não são mais referenciados ou utilizados pelo programa, ajudando a evitar vazamentos de memória e a otimizar o uso da memória.

Esse processo é automático, o que significa que os desenvolvedores não precisam gerenciar manualmente a alocação e a liberação de memória, como em algumas outras linguagens de programação.

> *Na linguagem Java, os tipos `int`, `char` e `long` são exemplos de dados elementares, e os tipos matrizes e vetores são dados estruturados da linguagem.*

Certo. Na linguagem Java, os tipos `int`, `char` e `long` são considerados tipos de dados primitivos (ou elementares). Já os tipos matrizes (arrays) e vetores (que em Java são representados por arrays) são considerados tipos de dados estruturados, pois permitem armazenar coleções de dados de forma organizada.




## ## Polimorfismo

> *O polimorfismo em Java permite processar objetos que derivam da mesma superclasse, direta ou indiretamente; cada objeto pode executar ações diferentes a partir da mesma chamada.*

- "O polimorfismo em Java permite processar objetos que derivam da mesma superclasse direta ou indiretamente"
  - **Explicação**: Isso significa que os objetos podem ser instâncias de subclasses que herdam diretamente da superclasse ou de subclasses que herdam de outras subclasses. Por exemplo, se `Cachorro` herda de `Animal` e `Labrador` herda de `Cachorro`, então `Labrador` é uma subclasse de `Animal` indiretamente.
- "cada objeto pode executar ações diferentes"

    - **Explicação**: Isso indica que, embora os objetos sejam tratados como instâncias da mesma superclasse, cada um pode ter seu próprio comportamento. Isso é possível porque cada subclasse pode sobrescrever métodos da superclasse, implementando sua própria lógica.
- "a partir da mesma chamada"
    - **Explicação**: Refere-se ao fato de que você pode chamar o mesmo método em diferentes objetos, e a implementação que será executada depende do tipo real do objeto. Por exemplo, se você chamar um método `fazerSom()` em um objeto do tipo `Animal`, o método pode se comportar de maneira diferente dependendo se o objeto é um `Cachorro` ou um `Gato`.

O trecho de código abaixo exemplifica o polimorfismo em Java através da herança e sobrescrita de métodos. A classe `Gerente` herda de `Empregado` e sobrescreve o método `recebeSalario()`, que, ao ser chamado em um objeto `Gerente`, retorna o salário base mais um bônus. Isso demonstra que o mesmo método pode ter comportamentos diferentes dependendo da classe do objeto, ilustrando o conceito de polimorfismo.

```java
class Empregado {
    private String nome;
    private double salario;
    public Empregado(String nome, double salario) {
        this.nome = nome;
        this.salario = salario;
    }
    public double recebeSalario() {
        return salario;
    }
}
class Gerente extends Empregado {
    private double bonus;
    public Gerente(String nome, double salario, double bonus) {
        super(nome, salario);
        this.bonus = bonus;
    }
    @Override
    public double recebeSalario() {
        return super.recebeSalario() + bonus;
    }
}
class FolhaPagamento {
    public static void main(String[] args) {
        Gerente gerente = new Gerente("Luiza", 5000, 1000);
        System.out.println(gerente.recebeSalario());
    }
}
```

# ## Modificadores de acesso

1. **public**: Os membros da classe (como métodos e variáveis) podem ser acessados de qualquer lugar, ou seja, de qualquer outra classe, independentemente do pacote em que estão. É como se você deixasse a porta da sua casa aberta para qualquer um entrar.
2. **private**: Os membros da classe só podem ser acessados dentro da própria classe. Ninguém de fora, nem mesmo outras classes do mesmo pacote, pode acessar. É

como se você trancasse a porta da sua casa e só você tivesse a chave.

3. **protected**: Os membros da classe podem ser acessados dentro da própria classe, em subclasses (que são classes que herdam da classe original, mesmo que estejam em pacotes diferentes) e em outras classes do mesmo pacote. É como se você tivesse uma porta que só pode ser aberta por você e por pessoas que você convidou (subclasses) ou que moram na mesma vizinhança (pacote).
4. **package-private** (ou **default**): Este é o modificador de acesso padrão em Java. Se você não especificar nenhum modificador, os membros da classe só podem ser acessados por outras classes do mesmo pacote. É como se você tivesse uma porta que só pode ser aberta por pessoas que moram na mesma rua (pacote).
    a. Um **pacote** em Java é uma forma de organizar classes e interfaces em grupos lógicos, facilitando a manutenção do código, evitando conflitos de nomes entre classes e controlando o acesso, permitindo que classes do mesmo pacote acessem membros umas das outras. Para declarar um pacote, utiliza-se a palavra-chave `package` no início do arquivo, como em `package veiculos;`, indicando que a classe pertence ao pacote `veiculos`.
5. **static**: Este modificador não controla o acesso, mas sim como um membro (método ou variável) é associado à classe em vez de a instâncias da classe. Um membro estático pertence à classe em si, e não a um objeto específico da classe. É como se você tivesse um item que pertence a todos os moradores da casa, e não apenas a um morador específico.

# ## Exceções e Erros Comuns

Exceções Comuns

1. **NullPointerException**: Ocorre quando o código tenta acessar um método ou variável de um objeto que é `null`.

```
Plain text
String str = null;
int length = str.length(); // Lança NullPointerException
```

2. **ArithmeticException**: Ocorre em operações aritméticas inválidas, como divisão por zero.

```
Plain text
int resultado = 10 / 0; // Lança ArithmeticException
```

3. **ArrayIndexOutOfBoundsException**: Ocorre quando se tenta acessar um índice de array que está fora dos limites do array.

```
Plain text
int[] array = {1, 2, 3};
int valor = array[3]; // Lança ArrayIndexOutOfBoundsException
```

4. **IndexOutOfBoundsException**: Ocorre quando se tenta acessar um índice inválido

em uma lista ou coleção.

```
List<String> lista = new ArrayList<>();
String item = lista.get(0); // Lança IndexOutOfBoundsException
```

5. **IllegalArgumentException**: Ocorre quando um método recebe um argumento inválido.

```
Thread thread = new Thread();
thread.setPriority(15); // Lança IllegalArgumentException, pois a prioridade deve estar entre 1 e 10
```

6. **NumberFormatException**: Ocorre quando se tenta converter uma string em um número, mas a string não tem o formato correto.

```
int numero = Integer.parseInt("abc"); // Lança NumberFormatException
```

7. **IOException**: Ocorre em operações de entrada/saída que falham ou são interrompidas.

Erros Comuns

1. **StackOverflowError**: Ocorre quando a pilha de chamadas de um thread se esgota, geralmente devido a recursão infinita.
2. **OutOfMemoryError**: Ocorre quando a Java Virtual Machine (JVM) não pode alocar mais memória.

# XML

XML, ou *eXtensible Markup Language*, é uma linguagem de marcação projetada para armazenar e transportar dados de forma estruturada e legível tanto por humanos quanto por máquinas. Ao contrário do HTML, que exibe dados, o XML descreve dados e sua estrutura em uma hierarquia de árvore, permitindo que os usuários definam suas próprias tags e formatos. É flexível, legível e amplamente utilizado para a troca de informações entre diferentes sistemas e plataformas, além de suportar caracteres de várias linguagens por meio do Unicode. XML é comumente empregado em aplicações web, serviços web e configuração de software.

> *XML é uma linguagem de marcação que armazena e descreve dados, permitindo que eles sejam transferidos entre diferentes aplicativos e(ou) sistemas.*

> *XML é uma linguagem de marcação que é projetada para armazenar e transportar dados enquanto que HTML é uma linguagem de marcação projetada para exibir conteúdo em navegadores da web.*

> *Em XML, a linguagem de marcação que define a estrutura e os elementos que vão compor um documento, além de suportar tipos de dados e namespaces, é chamada de **XML Schema**.*

Certo. O XML Schema é a linguagem de marcação que define a estrutura, os elementos, os tipos de dados e os namespaces que compõem um documento XML.

No contexto do XML, "tipos de dados" referem-se à definição da natureza dos valores que elementos e atributos podem ter, como inteiros, strings ou datas, permitindo a validação e a integridade dos dados em um documento XML. Já "namespaces" são utilizados para evitar conflitos de nomes entre elementos e atributos de diferentes fontes, qualificando os nomes com um URI (Uniform Resource Identifier) para diferenciá-los, como "ns1:item" e "ns2:item". Isso é especialmente útil em integrações que envolvem múltiplas fontes de dados.

Plain text

```
<Cliente Nome="JOAO DA SILVA" Identificacao="123456">
    <Endereco Logradouro="Rua Ativa, 123" Municipio="Recife" UF="PE"/>
</Cliente>
```

O trecho de código precedente está escrito em XML.

> *O código a seguir está correto de acordo com os padrões da sintaxe do XML?*

```
Plain text
<?xml version="1.0" encoding="UTF-8"?>
<menu>
    <bebida>
        <nome>Água<nome>
        <preco>R$2,50<preco>
    </bebida>
    <bebida>
        <nome>Vinho<nome>
        <preco>R$32,50<preco>
    </bebida>
</menu>
```

Errado. O código XML apresentado contém erros de sintaxe. As tags de fechamento para `<nome>` e `<preco>` estão incorretas; elas devem ser `</nome>` e `</preco>`, respectivamente.

> *Assinale a opção cujo trecho de código em XML representa corretamente uma pessoa, com seu nome e idade:* `<Pessoa Nome="Fulano" Idade="50" />`

Pessoa é uma tag de auto fechamento. A sintaxe de auto fechamento é válida e comum em XML para elementos que não têm conteúdo interno. Mesmo que a idade seja um número, em XML, os valores dos atributos devem ser sempre delimitados por aspas, independentemente de serem numéricos ou textuais.

> *XML é a tecnologia que torna possível a interoperacionalidade caracterizada na situação em que o usuário informa o seu código de endereçamento postal (CEP) em determinado site e outra plataforma de serviços retorna com o endereço correspondente a esse CEP.*

Certo. O XML (eXtensible Markup Language) é uma tecnologia que permite a troca de dados entre diferentes sistemas e plataformas, facilitando a interoperabilidade. No exemplo que você mencionou, quando um usuário informa o CEP em um site, o sistema pode enviar essa informação em formato XML para outra plataforma, que então retorna o endereço correspondente. Essa capacidade de estruturar e compartilhar dados de forma padronizada é uma das principais características do XML.

> *API ~~é um padrão XML~~ usado para desenvolver uma interface de aplicativos em dispositivos computacionais em servidores HTTP ou HTTPS.*

Errado. As partes da frase que estão erradas são:

1. **"API é um padrão XML"** : Isso é incorreto. API (Application Programming Interface)

não é um padrão XML. Uma API é um conjunto de definições e protocolos que permite a comunicação entre diferentes softwares. APIs podem usar diversos formatos de dados, incluindo XML, JSON, entre outros, mas não são definidas como um padrão XML.

2. **"usado para desenvolver uma interface de aplicativos"** : Embora APIs sejam usadas para permitir a interação entre aplicativos, a frase pode ser confusa. APIs não "desenvolvem" interfaces; elas fornecem um meio para que diferentes sistemas se comuniquem e interajam. O desenvolvimento de uma interface de usuário pode utilizar uma API, mas a API em si não é responsável por isso.

Portanto, a frase poderia ser reformulada para algo como: "Uma API é um conjunto de definições e protocolos que permite a comunicação entre aplicativos, podendo utilizar formatos como XML ou JSON em servidores HTTP ou HTTPS."

# JPA

A Java Persistence API (**JPA**) é uma especificação da plataforma Java que facilita o gerenciamento de dados relacionais em aplicações Java por meio do mapeamento objeto-relacional (ORM). Ela permite que desenvolvedores mapeiem classes Java para tabelas de banco de dados e objetos para registros, simplificando operações de persistência de dados. A JPA oferece uma linguagem de consulta chamada JPQL (Java Persistence Query Language) para realizar consultas em entidades de forma orientada a objetos, além de suportar transações para garantir a consistência dos dados.

Várias implementações, como Hibernate e EclipseLink, fornecem as funcionalidades definidas pela JPA, tornando-a uma ferramenta poderosa para o desenvolvimento de aplicações que interagem com bancos de dados relacionais.

ORM, ou Object-Relational Mapping, é uma técnica que permite a conversão de dados entre sistemas de tipos incompatíveis em programação orientada a objetos e bancos de dados relacionais, facilitando a interação entre a aplicação e o banco de dados. Com um ORM, os desenvolvedores podem trabalhar com dados como objetos, em vez de usar diretamente SQL, o que resulta em uma abstração da complexidade do banco de dados, redução de código para operações de CRUD, e gerenciamento automático de relacionamentos entre entidades.

## ## Hibernate

O **Hibernate** é uma implementação da Java Persistence API (JPA) que facilita o mapeamento objeto-relacional em aplicações Java, permitindo que desenvolvedores trabalhem com dados como objetos, sem a complexidade do SQL. Ele gerencia sessões de banco de dados, oferece suporte a caching para melhorar o desempenho, permite consultas através da Hibernate Query Language (HQL) e integra-se com o gerenciamento de transações, além de ser compatível com diversos sistemas de gerenciamento de banco de dados, tornando o desenvolvimento de aplicações mais eficiente e portável.

**Session**: No Hibernate e na Java Persistence API (JPA), uma `session` é uma interface que representa uma conexão entre a aplicação e o banco de dados, gerenciando o ciclo de vida das entidades e permitindo operações de persistência, como criar, ler, atualizar e excluir objetos. Ela mantém um cache de primeiro nível para otimizar o acesso a entidades e deve ser fechada após o uso para liberar recursos.

**Transaction:** Uma `transaction` (transação) refere-se a uma unidade de trabalho que é executada de forma atômica em um banco de dados. Isso significa que uma transação deve ser completada com sucesso em sua totalidade ou não ser aplicada de forma alguma, garantindo a integridade dos dados. As transações são fundamentais para manter a consistência e a confiabilidade dos dados em sistemas que realizam operações

de leitura e escrita em bancos de dados.

No Hibernate, a interface `Transaction` é usada para controlar o ciclo de vida das transações. Os principais métodos dessa interface incluem:

1. `begin()` : Inicia uma nova transação.
2. `commit()` : Confirma as alterações feitas durante a transação, persistindo-as no banco de dados.
3. `rollback()` : Desfaz todas as alterações feitas durante a transação, revertendo o estado do banco de dados para o que era antes do início da transação.
4. `isActive()` : Verifica se a transação está ativa.

O uso adequado de transações é crucial para garantir que operações críticas, como inserções, atualizações e exclusões, sejam realizadas de maneira segura e consistente.

> *Em desenvolvimento web, é comum a utilização de classes, tecnologias ou mesmo ferramentas e frameworks, como aquela que utiliza sua própria linguagem de consulta, o que facilita a etapa de desenvolvimento quanto ao acesso a bancos de dados e SQL. Esse exemplo enfatiza uma das vantagens do Hibernate.*

> *O Hibernate implementa ORM (object-relational mapping) entre a camada de acesso a dados do aplicativo Java e o banco de dados relacional, de modo que o aplicativo Java faz uso das APIs do Hibernate para carregar e armazenar seus dados de domínio.*

> *O Hibernate, um framework de mapeamento objeto relacional (ORM), cria uma camada persistência na solução desenvolvida, o que permite ligar os objetos aos bancos de dados relacionais. Entre seus serviços, o Hibernate provê um meio de se controlar transações, por meio de métodos de suas interfaces session e transaction, tendo ainda suporte a herança e polimorfismo. É distribuído sob a licença LGPL, o que permite seu uso em projetos comerciais ou open source.*

> *No desenvolvimento de uma aplicação em que ocorram persistências a dados usando Hibernate, é necessário criar uma sessão para fazer a conexão com o banco de dados por meio de um objeto session; este objeto, que ~~é instanciado apenas uma vez na aplicação, deve se manter instanciado durante todo o tempo de execução~~.*

As partes incorretas da frase são:

1. **"este objeto, que é instanciado apenas uma vez na aplicação"** : O objeto `Session` não deve ser instanciado apenas uma vez. Em vez disso, uma nova sessão deve ser criada para cada operação de interação com o banco de dados.

2. **"deve se manter instanciado durante todo o tempo de execução"** : O `Session` não deve ser mantido instanciado durante todo o tempo de execução da aplicação. O padrão recomendado é abrir uma sessão, realizar as operações necessárias e, em seguida, fechá-la.

Essas correções são importantes para garantir um gerenciamento adequado de recursos e evitar problemas de desempenho e concorrência na aplicação.

> *Empresa de desenvolvimento que opte pela utilização do Hibernate em seus sistemas ~~enfrentará dificuldades~~ à medida que seus projetos forem crescendo, ~~devido ao fato de o Hibernate ser considerado inapropriado para a execução de trabalhos em uma arquitetura altamente escalável~~.*

Errado. O Hibernate é uma ferramenta robusta e amplamente utilizada para mapeamento objeto-relacional e pode ser adequada para aplicações em arquiteturas escaláveis. Embora existam considerações de desempenho e complexidade que devem ser gerenciadas à medida que os projetos crescem, o Hibernate oferece recursos como caching, suporte a transações e otimizações que podem ajudar a lidar com a escalabilidade. Além disso, muitas empresas de grande porte utilizam o Hibernate em sistemas escaláveis com sucesso. No entanto, é importante que os desenvolvedores estejam cientes das melhores práticas e considerações de desempenho ao usar o Hibernate em aplicações de grande escala.

> *No framework Hibernate, é comum que uma instância de uma classe persistente tenha três estados específicos: transient, persistent, detached.*

No Hibernate, uma instância de uma classe persistente pode estar em um dos três estados:

1. **Transient**: O objeto foi criado, mas não está associado a uma sessão e não foi persistido no banco de dados.
2. **Persistent**: O objeto está associado a uma sessão e suas alterações são rastreadas pelo Hibernate, sendo que ele está persistido no banco de dados.
3. **Detached**: O objeto foi persistido anteriormente, mas a sessão que o gerenciava foi fechada, portanto, ele não está mais associado a uma sessão, mas ainda representa uma linha no banco de dados.

Saber sobre os estados de uma instância de uma classe persistente no Hibernate (transient, persistent e detached) é crucial para o gerenciamento eficaz do ciclo de vida das entidades, otimização de desempenho e manutenção da consistência dos dados. Compreender esses estados ajuda os desenvolvedores a realizar operações de persistência de forma adequada, evita problemas de inconsistência ao modificar objetos detached, e é essencial para o gerenciamento de transações. Além disso, esse

conhecimento facilita o debugging e a manutenção das aplicações, garantindo que elas funcionem de maneira eficiente e confiável.

> *Com relação à JPA (Java persistence API), julgue os itens subsequentes.*

Ao se declarar uma coluna que seja a chave primária de uma tabela, é necessário utilizar a anotação `@Id`.

Certo. Ao declarar uma coluna que será a chave primária de uma tabela em uma entidade JPA, é necessário utilizar a anotação `@Id`. Essa anotação indica que o campo correspondente é a chave primária da entidade, permitindo que o JPA gerencie corretamente a persistência e a identificação dos objetos no banco de dados.

# JDBC

JDBC (Java Database Connectivity) é uma API da plataforma Java que permite a interação entre aplicações Java e bancos de dados relacionais, fornecendo um conjunto de classes e interfaces para executar operações de banco de dados, como consultas e manipulação de dados, de forma independente do sistema de gerenciamento de banco de dados utilizado. A API possibilita estabelecer conexões com diferentes bancos de dados, executar instruções SQL através de objetos como `Statement` e `PreparedStatement`, processar resultados com `ResultSet`, gerenciar erros e controlar transações, tornando-se uma ferramenta essencial para o desenvolvimento de aplicações Java que necessitam de persistência de dados.

Aqui está uma tabela comparativa entre JDBC e JPA:

| **Característica** | **JDBC** | **JPA** |
|---|---|---|
| **Nível de Abstração** | Baixo nível (API de baixo nível) | Alto nível (API de alto nível) |
| **Modelo de Programação** | Baseado em SQL (consultas SQL diretas) | Baseado em objetos (mapeamento objeto-relacional) |
| **Complexidade** | Mais complexo, requer gerenciamento manual de conexões e transações | Menos complexo, gerenciamento automático de entidades e transações |
| **Manipulação de Dados** | Requer escrita de SQL para operações CRUD | Usa entidades Java e JPQL para operações CRUD |
| **Gerenciamento de Estado** | Não gerencia o estado das entidades | Gerencia o estado das entidades (transient, persistent, detached) |
| **Caching** | Não possui suporte nativo a caching | Suporte a caching de primeiro e segundo nível |
| **Transações** | Gerenciamento manual de transações | Gerenciamento automático de transações |
| **Portabilidade** | Dependente do driver JDBC específico | Mais portátil, pois é independente do banco de dados (desde que o provedor JPA suporte) |
| **Uso Comum** | Acesso direto a bancos de dados | Aplicações que requerem persistência de dados de forma orientada a objetos |

# JSF e Facelets

JavaServer Faces (JFS) é uma especificação da plataforma Java EE que simplifica o desenvolvimento de interfaces de usuário para aplicações web em Java, permitindo a criação de componentes reutilizáveis, como botões e formulários. O JFS gerencia o estado dos componentes entre requisições HTTP, facilita a integração com a lógica de negócios, oferece um mecanismo de navegação para transições entre páginas e suporta AJAX para atualizações dinâmicas da interface, melhorando a experiência do usuário. Em resumo, o JFS proporciona uma abordagem baseada em componentes que torna a criação e manutenção de interfaces de usuário mais eficiente e organizada.

> *Java Server Faces (JFS) é uma implementação voltada para interface de aplicações web com o usuário, com um modelo de programação dirigida a eventos.*

Certo. JavaServer Faces (JSF) é uma tecnologia voltada para a construção de interfaces de usuário em aplicações web, utilizando um modelo de programação orientado a eventos. Isso significa que os desenvolvedores podem criar componentes de interface que respondem a ações do usuário, como cliques de botão ou envios de formulários, facilitando a construção de aplicações interativas e dinâmicas.

> *JSF (JavaServer Faces) é uma tecnologia que oferece a separação entre as camadas de apresentação e de comportamento para aplicativos web.*

Certo. JSF (JavaServer Faces) realmente oferece uma separação entre as camadas de apresentação e de comportamento em aplicativos web, permitindo que os desenvolvedores se concentrem na lógica de negócios e na interface do usuário de forma independente.

> *O JSF é um framework web embasado em interface gráfica, capaz de renderizar componentes e manipular eventos em aplicações web no padrão Java EE, no qual os componentes JSF são orientados a eventos. O JSF fornece, ainda, mecanismos para conversão, validação, execução de lógica de negócios e controle de navegação.*

## ## Facelets

Facelets é uma tecnologia de visualização que faz parte da especificação JavaServer Faces (JSF) e permite a criação de interfaces de usuário em aplicações web utilizando XHTML. Ela facilita a construção de páginas de forma declarativa, promovendo a reutilização de componentes e a criação de templates para layouts consistentes. Facelets integra-se perfeitamente com JSF, aproveitando seus recursos de gerenciamento de estado e eventos, e suporta a inclusão de recursos dinâmicos, como arquivos de estilo e scripts, tornando as páginas mais interativas e personalizáveis. Em resumo, Facelets é uma ferramenta poderosa para desenvolver interfaces ricas e

modulares em aplicações web.

> *Facelets é uma linguagem de declaração que faz parte da especificação JSF e que permite, com uso de XHTML, a criação de páginas web.*