

Téc em Desenvolvimento
de Sistemas Bilingue

Desenvolver Código

Orientado a Objetos

Paradigmas de programação

Os paradigmas de programação indicam formas de pensar e organizar o código, determinando regras que buscam garantir:

- Legibilidade
- Consistência
- Testabilidade
- Fácil manutenção

Paradigmas de programação

Estruturada

- Utiliza estruturas condicionais e loops para controlar o fluxo de execução do programa
- Impede controle absoluto do fluxo de execução do programa

Orientada a objetos

- Utiliza objetos para modelar e controlar os dados e suas manipulações
- Restringe acesso a variáveis e rotinas a partes específicas do código

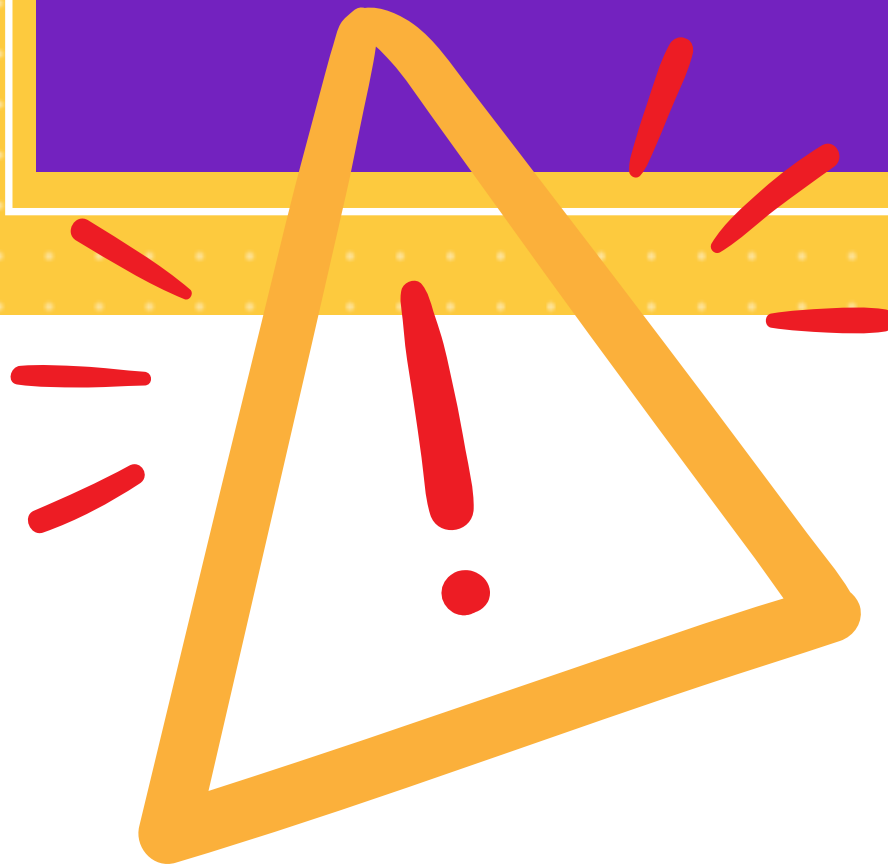
Orientação a Objetos

- A POO é um dos paradigmas mais difundidos e conhecidos
- Criada por volta dos anos 60 para ajudar a lidar com a complexidade crescente dos programas
- Utiliza principalmente o conceito de objetos como unidade central de estruturação de dados e rotinas
- Os objetos podem ser criados de forma a abstrair conceitos do mundo real, simplificando o entendimento do código por pessoas

Objetos

- No contexto da POO, cada objeto é definido por um conjunto de:
 - **Atributos:** informações (dados) do objeto
 - **Métodos:** rotinas (funções) que operam sobre os atributos do objeto, ou contextualmente relacionadas
- Os objetos são criados a partir de uma classe, que define quais atributos ele deve ter, e quais são os métodos que ele terá quando for criado

Objetos



Atenção!

Nesse contexto, o conceito de objetos é um pouco diferente do que conhecemos como “objeto” no Javascript. Os objetos do JS podem ter atributos e métodos, mas não precisam ser criados a partir de algum “template” (como as classes).

Classes

- Define o formato de um objeto
 - Atributos (propriedades)
 - Métodos (funções)
- Possuem as implementações dos métodos e podem possuir valores padrão para os atributos
- No Typescript, por definir as propriedades e métodos, representa também um **tipo**, que pode ser usado para definir variáveis. Chamamos de Tipos Abstratos de Dados.

Abstração

- Forma de criar representações mais simples de coisas complexas, escondendo essa complexidade com interfaces mais simples.
- Esconde detalhes de implementação, facilitando o uso e garantindo consistência, mas limitando possibilidades
- É possível criar múltiplos níveis de abstração
- Boas abstrações tornam o código mais legível, reaproveitável, consistente e expansível, mas são difíceis de fazer e requerem muito estudo e iteração

Exemplos

Cachorro

nome: string
peso: number

latir(): void
comer(quantidade: number): void



Exemplos

Coruja

nome: string
peso: number

chirriar(): void
comer(quantidade: number): void
voar(tempo: number): void



Exemplos

User

nome: string
tarefas: Tarefa[]

adicionarTarefas(): void
removerTarefa(): void



Declarando uma classe

- A declaração da classe representa o TAD. Ela é um **modelo** para objetos daquele tipo.
- A classe pode ser usada como um **tipo** no TypeScript
- É possível criar diversas **instâncias** da classe. São **objetos** que necessariamente possuem todos aqueles atributos e métodos

```
export class Dog {  
  name: string;  
  weight: number;  
  
  bark(): void {  
    console.log("Au Au 🐕");  
  }  
  
  eat(quantity: number): void {  
    console.log("the dog has eaten " + quantity);  
  }  
}
```

```
export class Owl {  
  name: string;  
  weight: number;  
  
  chirp(): void {  
    console.log("Hu Hu 🦉");  
  }  
  
  eat(quantity: number): void {  
    console.log("the owl has eaten " + quantity);  
  }  
  
  fly(quantity: number): void {  
    console.log("the owl has flown for " + quantity + " minutes");  
  }  
}
```

Inicializando (Construindo)

- Podemos ter ações que devem ser executadas ao criar uma instância da classe
- Para isso, existe o construtor
- Deve ser declarado com a função **constructor**
- Pode receber parâmetros, que devem ser passados no momento de criar a instância
- Toda classe possui um constructor. Quando não o explicitamos, ele existe como construtor vazio

This

- A palavra **this** é usada como uma referência para a instância corrente, ou seja, para o objeto instanciado atualmente na memória
- A referência this é usada para referenciar os atributos e métodos daquela instância
- Normalmente usada no construtor e nos métodos para acessar e manipular os atributos do objeto

Novo Objeto

- Utilizamos a keyword **class** para declarar uma classe, que será nosso modelo para construir objetos (instâncias)
- O corpo da classe pode conter uma função chamada **constructor**, que recebe como parâmetros os atributos dessa classe, referenciados pela keyword **this**
- A criação de novas instâncias é feita com a keyword **new**

Exercício 1

- Vamos pensar em um **RPG**. Para tal precisamos construir **duas classes**, ambos podem atacar e sofrer dano:
 - **Guerreiro** - inicia com 100 de vida
 - Nome
 - Tipo
 - Força
 - Saúde
 - **Monstro** - inicia com 50 de força e 10 de vida
 - Nome
 - Força
 - Saúde

Encapsulamento

- o encapsulamento refere-se à prática de agrupar os membros (propriedades e métodos) de uma classe e controlar o acesso a esses membros.
- é uma metáfora emprestada da biologia, onde uma cápsula (ou casca) envolve e protege seu conteúdo.
- Da mesma forma, na programação orientada a objetos, encapsulamento refere-se a empacotar (ou encapsular) dados e os métodos que os manipulam em uma única unidade, conhecida como classe.

Encapsulamento

- Boas abstrações escondem detalhes da implementação, disponibilizando somente o necessário para o uso correto do objeto
- Isso significa expor somente métodos específicos
- Os atributos, em geral, são todos escondidos e disponibilizados por meio de métodos

public & private

- É possível declarar atributos e métodos de classe com as keywords **public** e **private**
- Por padrão, as variáveis são públicas, o que significa que elas podem ser acessadas fora da classe
- Variáveis privadas só podem ser acessadas de dentro da própria classe (usando a keyword **this**)


```
export class Dog {  
  private name: string;  
  private weight: number;  
  
  constructor(name: string, weight: number){  
    this.name = name;  
    this.weight = weight;  
  }  
  
  bark(): void {  
    console.log("au au 🐶");  
  }  
  
  eat(quantity: number): void {  
    console.log("the dog has eaten " + quantity);  
  }  
}
```

private name: string;

(property) Dog.name: string

Property 'name' is private and only accessible within class 'Dog'. ts(2341)

const [Peek Problem](#) No quick fixes available

```
dog1.name = "Snoopy";
```

getters & setters

- É comum tornar todos os atributos privados e controlar o acesso por métodos públicos
- Esses métodos são chamados de **getters** (para pegar o atributo) e **setters** (para definir)
- Isso garante consistência e extensibilidade

```
public getName() {  
    return this.name  
}
```

```
public setName(newName: string) {  
    this.name = newName  
}
```

Exercício 3

- Utilize o public e private para encapsular suas classes