SENAC/SÃO LEOPOLDO

estruturas de DADOS

BY RAFA

rafael_c_alves@hotmail.com

estruturas de DADOS

- REVISÃO DE POO
- ESTRUTURAS DE DADOS

ENUMERADORES

ENUMERADORES

Numérico e explicito

```
export enum Level {
  BLUE = 0,
  YELLOW = 10,
  ORANGE = 20,
  RED = 30
}
```

Numérico e implicito

```
export enum Level {
  BLUE,    \\ 0
  YELLOW, \\ 1
  ORANGE, \\ 2
  RED     \\ 3
}
```

Numérico e explicito incremental

```
export enum Level {
  BLUE = 1,  \\ 1
  YELLOW,    \\ 2
  ORANGE,    \\ 3
  RED        \\ 4
}
```

```
export enum Level {
  BLUE = "Blue",
  YELLOW = "Yellow",
  ORANGE = "Orange",
  RED = "Red"
}
```

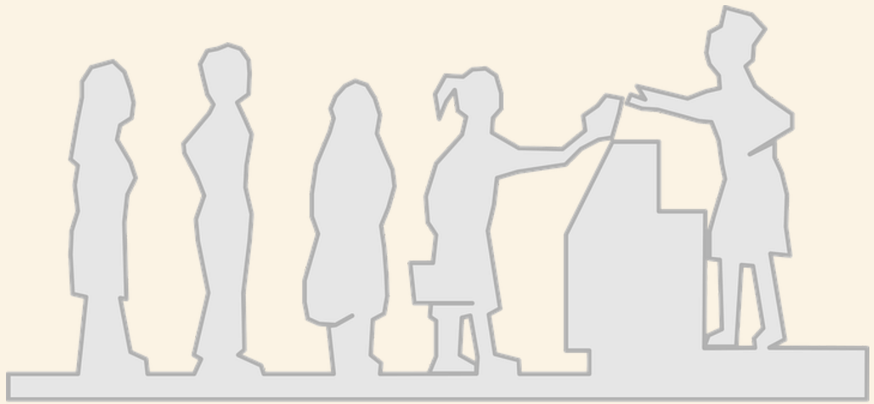
String

ENUMERADORES

```
export enum Level {
  BLUE = 1,
  YELLOW,
  ORANGE,
  RED
}
```

```
export class Survivor {
  protected level: Level;

  getLevel(): Level {
    return this.level;
  }

  levelUp(): void {
    this.level++;
  }
}
```

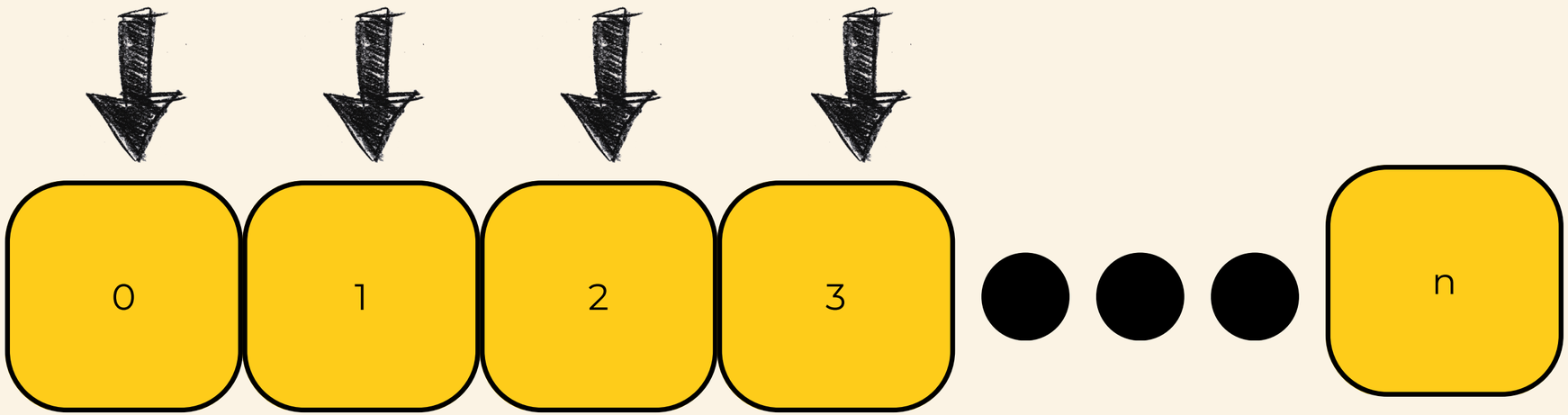
ESTRUTURAS DE DADOS



O que são?

Tipos não-primitivos de organização de dados para atender aos diferentes requisitos de processamento

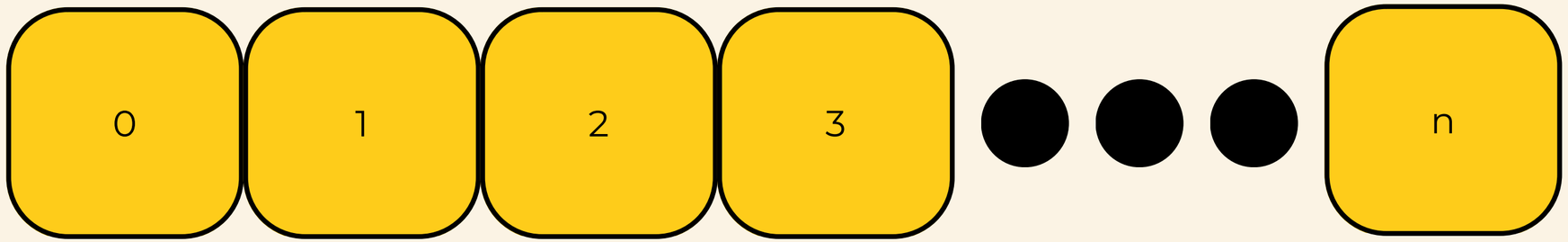
ARRAY



Posições bem definidas
Tamanho limitado (nem sempre)
Itens podem ser adicionados em qualquer posição

LISTA

Array de "luxo"

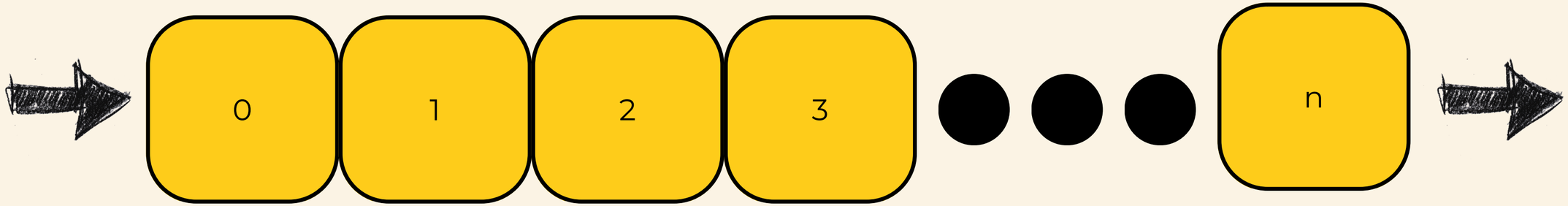


Posições bem definidas
Tamanho se adapta à posição ocupada de maior índice
Itens podem ser adicionados em qualquer posição
Não possui ordem de saída definida

```
export interface IList {
  add(item: string): void;
  remove(index: number): void;
  get(index: number): void;
  set(index: number, item: string): void;
  contains(item: string): boolean;
  size(): number;
  isEmpty(): boolean;
}
```

FILA

Primeiro a Chegar Primeiro a Sair (PCPS)
First In, First Out (FIFO)

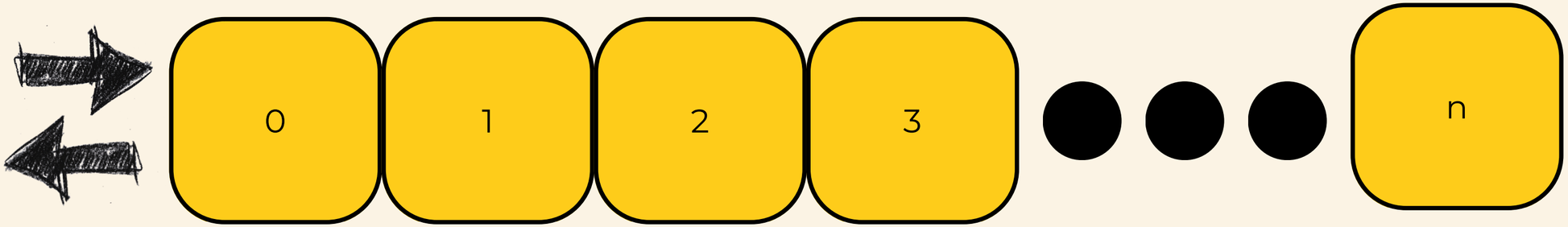


Posições bem definidas
Tamanho se adapta à posição ocupada de maior índice
Itens podem ser adicionados apenas no menor índice (zero)
Saída deve ser feita pelo maior índice ocupado

```
export interface IQueue {
  enqueue(item: string): void;
  dequeue(): string;
  size(): number;
  isFull(): boolean;
}
```

PILHA

Primeiro a Chegar Último a Sair (PCUS)
First In, Last Out (FILO)



Posições bem definidas
Tamanho se adapta à posição ocupada de maior índice
Itens podem ser adicionados apenas no menor índice (zero)
Saída deve ser feita pelo menor índice ocupado

```
export interface IStack {
  push(item: string): void;
  pop(): string;
  size(): number;
  isFull(): boolean;
}
```

INFORMAÇÃO EXTRA

5 kyu **Vector class** ✓

Create a Vector object that supports addition, subtraction, dot products, and norms. So, for example:

```
a = new Vector([1, 2, 3])
b = new Vector([3, 4, 5])
c = new Vector([5, 6, 7, 8])

a.add(b)      # should return a new Vector([4, 6, 8])
a.subtract(b) # should return a new Vector([-2, -2, -2])
a.dot(b)      # should return 1*3 + 2*4 + 3*5 = 26
a.norm()      # should return sqrt(1^2 + 2^2 + 3^2) = sqrt(14)
a.add(c)      # throws an error
```

If you try to add, subtract, or dot two vectors with different lengths, ***you must throw an error***!

Also provide:

- a `toString` method, so that using the vectors from above, `a.toString() === '(1,2,3)'` (in Python, this is a `__str__` method, so that `str(a) == '(1,2,3)'`)
- an `equals` method, to check that two vectors that have the same components are equal

Note: the test cases will utilize the user-provided `equals` method.

5 kyu **Vector class** ✓

Create a Vector object that supports addition, subtraction, dot products, and norms. So, for example:

```
a = new Vector([1, 2, 3])
b = new Vector([3, 4, 5])
c = new Vector([5, 6, 7, 8])

a.add(b)      # should return a new Vector([4, 6, 8])
a.subtract(b) # should return a new Vector([-2, -2, -2])
a.dot(b)      # should return 1*3 + 2*4 + 3*5 = 26
a.norm()      # should return sqrt(1^2 + 2^2 + 3^2) = sqrt(14)
a.add(c)      # throws an error
```

If you try to add, subtract, or dot two vectors with different lengths, ***you must throw an error***!

Also provide:

- a `toString` method, so that using the vectors from above, `a.toString() === '(1,2,3)'` (in Python, this is a `__str__` method, so that `str(a) == '(1,2,3)'`)
- an `equals` method, to check that two vectors that have the same components are equal

Note: the test cases will utilize the user-provided `equals` method.

5 kyu **Vector class**

Create a Vector object that supports addition, subtraction, dot products, and norms. So, for example:

```
a = new Vector([1, 2, 3])
b = new Vector([3, 4, 5])
c = new Vector([5, 6, 7, 8])

a.add(b)      # should return a new Vector([4, 6, 8])
a.subtract(b) # should return a new Vector([-2, -2, -2])
a.dot(b)      # should return 1*3 + 2*4 + 3*5 = 26
a.norm()      # should return sqrt(1^2 + 2^2 + 3^2) = sqrt(14)
a.add(c)      # throws an error
```

If you try to add, subtract, or dot two vectors with different lengths, ***you must throw an error***!

Also provide:

- a `toString` method, so that using the vectors from above, `a.toString() === '(1,2,3)'` (in Python, this is a `__str__` method, so that `str(a) == '(1,2,3)'`)
- an `equals` method, to check that two vectors that have the same components are equal

Note: the test cases will utilize the user-provided `equals` method.

```
export class Vector {
  components: Array<number>;
  // implementação do constructor

  add(vector: Vector): Vector {
    if(vector.components.length !=
         this.components.length) {
      // tratamento do erro
    }

    let output: number[] = [];
    for (const i in this.components){
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

5 kyu **Vector class**

Create a Vector object that supports addition, subtraction, dot products, and norms. So, for example:

```
a = new Vector([1, 2, 3])
b = new Vector([3, 4, 5])
c = new Vector([5, 6, 7, 8])

a.add(b)      # should return a new Vector([4, 6, 8])
a.subtract(b) # should return a new Vector([-2, -2, -2])
a.dot(b)      # should return 1*3 + 2*4 + 3*5 = 26
a.norm()      # should return sqrt(1^2 + 2^2 + 3^2) = sqrt(14)
a.add(c)      # throws an error
```

If you try to add, subtract, or dot two vectors with different lengths, ***you must throw an error***!

Also provide:

- a `toString` method, so that using the vectors from above, `a.toString() === '(1,2,3)'` (in Python, this is a `__str__` method, so that `str(a) == '(1,2,3)'`)
- an `equals` method, to check that two vectors that have the same components are equal

Note: the test cases will utilize the user-provided `equals` method.

```
export class Vector {
  components: Array<number>;
  // implementação do constructor

  add(vector: Vector): Vector {
    if(vector.components.length !=
          this.components.length) {
      // tratamento do erro
    }

    let output: number[] = [];
    for (const i in this.components){
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

Opção 1:

```
export class Vector {
  components: Array<number>;
  // implementação do constructor
  // constructor(components: number[])

  add(vector:Vector):Vector{
    if(vector.components.length !=
         this.components.length) {
      console.log("Tamanhos diferentes")
      return new Vector([]);
    }
    let output: number[] = [];
    for (const i in this.components){
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

Opção 2:

```
export class Vector {
  components: Array<number>;
  // implementação do constructor
  // constructor(components: number[])

  add(vector:Vector):Vector{
    if(vector.components.length !=
         this.components.length) {
      throw new Error("Tamanhos diferentes")
    }
    let output: number[] = [];
    for (const i in this.components){
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

Opção 1:

```
export class Vector {
  components: Array<number>;
  // implementação do constructor
  // constructor(components: number[])

  add(vector:Vector):Vector{
    if(vector.components.length !=
         this.components.length) {
      console.log("Tamanhos diferentes")
      return new Vector([]);
    }
    let output: number[] = [];
    for (const i in this.components){
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

Opção 2:

```
export class Vector {
  components: Array<number>;
  // implementação do constructor
  // constructor(components: number[])

  add(vector:Vector):Vector{
    if(vector.components.length !=
         this.components.length) {
      throw new Error("Tamanhos diferentes")
    }
    let output: number[] = [];
    for (const i in this.components){
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

Opção 1:

```
export class Vector {
  components: Array<number>;
  // implementação do constructor
  // constructor(components: number[])

  add(vector:Vector):Vector{
    if(vector.components.length !=
         this.components.length) {
      console.log("Tamanhos diferentes")
      return new Vector([]);
    }
    let output: number[] = [];
    for (const i in this.components){
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

Opção 2:

```
export class Vector {
  components: Array<number>;
  // implementação do constructor
  // constructor(components: number[])

  add(vector:Vector):Vector{
    if(vector.components.length !=
         this.components.length) {
      throw new Error("Tamanhos diferentes")
    }
    let output: number[] = [];
    for (const i in this.components){
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

Opção 2:

```
export class Vector {
  components: Array<number>;
  // implementação do constructor
  // constructor(components: number[])

  verifyLength(vector:Vector): boolean {
    if(vector.components.length !=
         this.components.length) {
      throw new Error("Tamanhos diferentes")
    }
  }
  add(vector:Vector): Vector {
    this.verifyLength(vector)
    let output: number[] = []
    for (const i in this.components) {
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

Opção 1:

```
export class Vector {
  components: Array<number>;
  // implementação do constructor
  // constructor(components: number[])

  hasSameSize(vector:Vector): boolean {
    if(vector.components.length ===
         this.components.length) {
      console.log("Tamanhos diferentes")
      return true
    }
    return false
  }
  add(vector:Vector): Vector {
    if(!this.hasSameSize(vector))
      return new Vector([])
    let output: number[] = []
    for (const i in this.components) {
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

Opção 1:

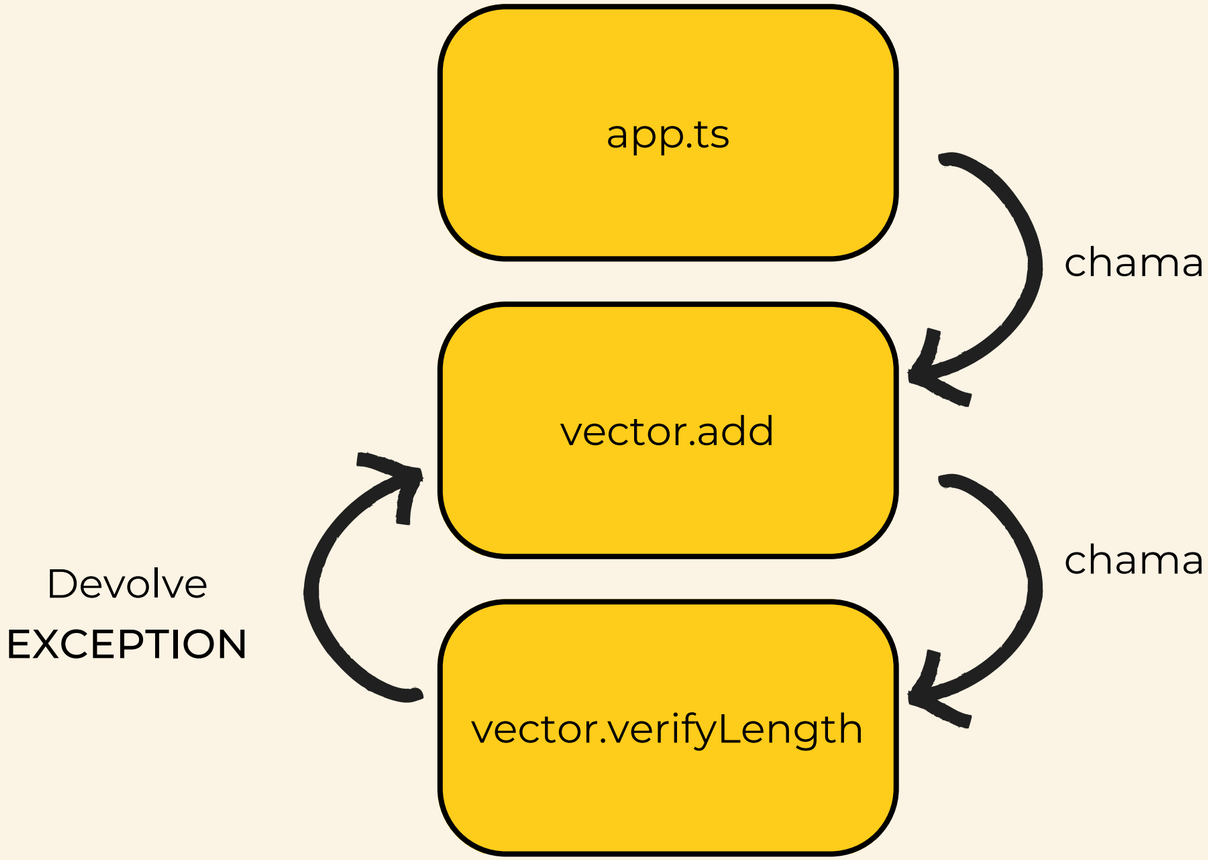
```
export class Vector {
  components: Array<number>;
  // implementação do constructor
  // constructor(components: number[])

  hasSameSize(vector:Vector): boolean {
    return vector.components.length ===
         this.components.length
  }
  add(vector:Vector): Vector {
    if(!this.hasSameSize(vector))
      return new Vector([])
    let output: number[] = []
    for (const i in this.components) {
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

Opção 2:

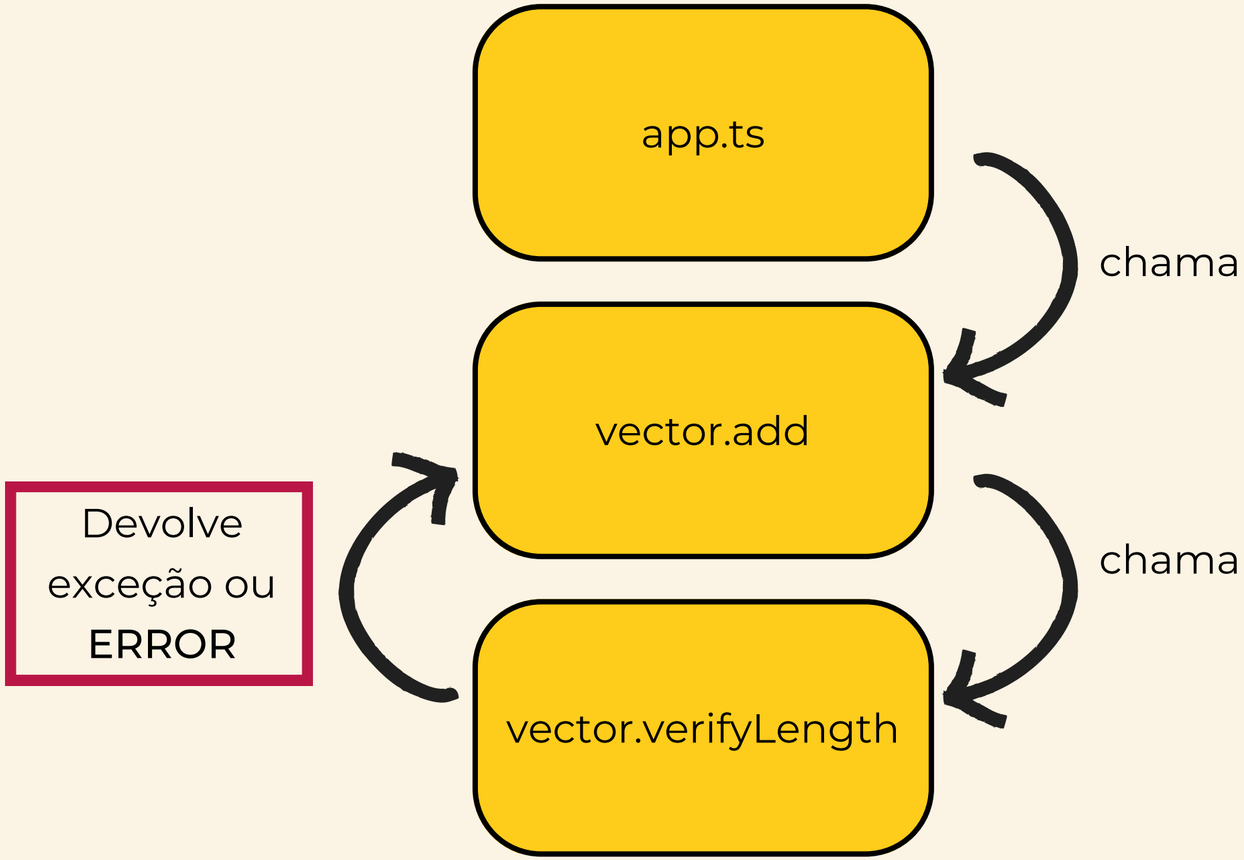
```
export class Vector {
  components: Array<number>;
  // implementação do constructor
  // constructor(components: number[])

  verifyLength(vector:Vector): boolean {
    if(vector.components.length !=
         this.components.length) {
      throw new Error("Tamanhos diferentes")
    }
  }
  add(vector:Vector): Vector {
    this.verifyLength(vector)
    let output: number[] = []
    for (const i in this.components) {
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

EXCEÇÕES

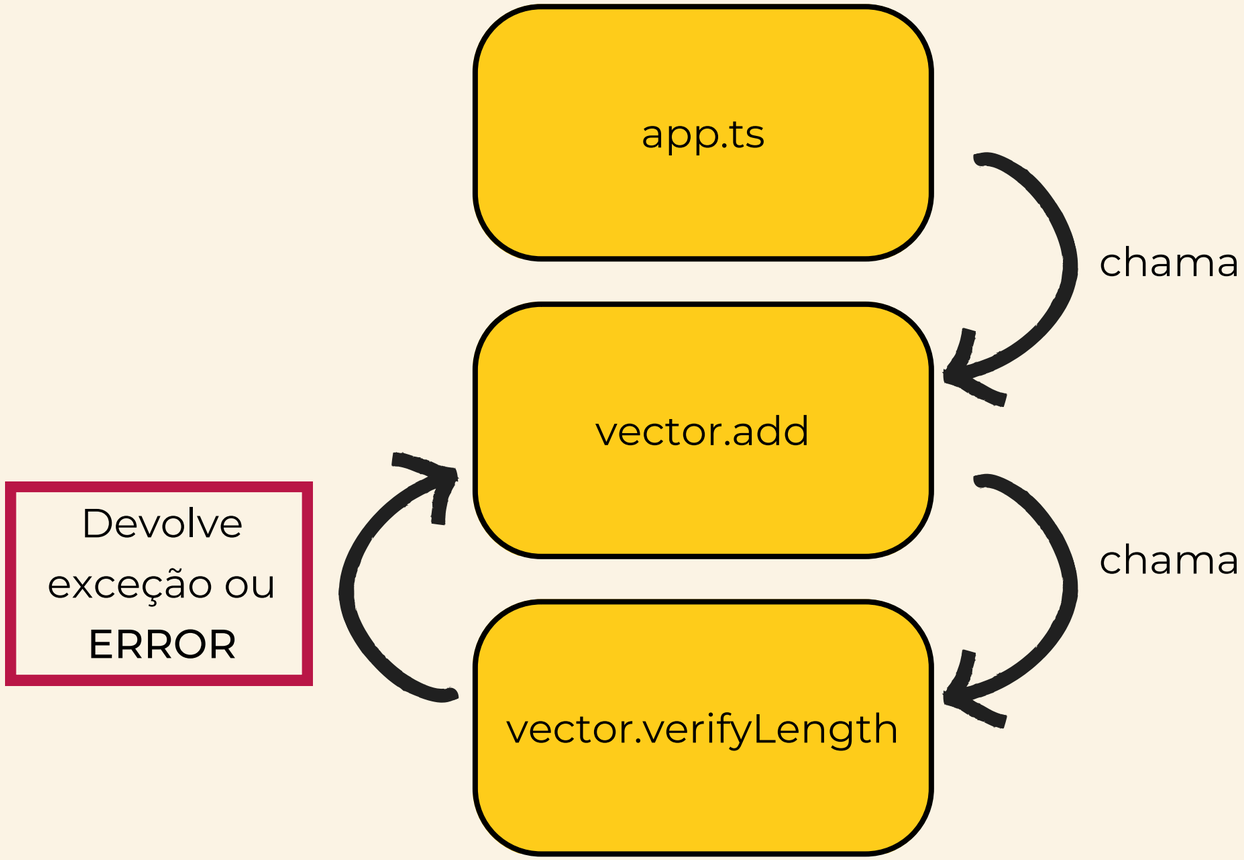
EXCEÇÕES


app.ts
chama
vector.add
chama
Devolve
EXCEPTION
vector.verifyLength

EXCEÇÕES


app.ts
chama
vector.add
chama
vector.verifyLength
Devolve exceção ou ERROR

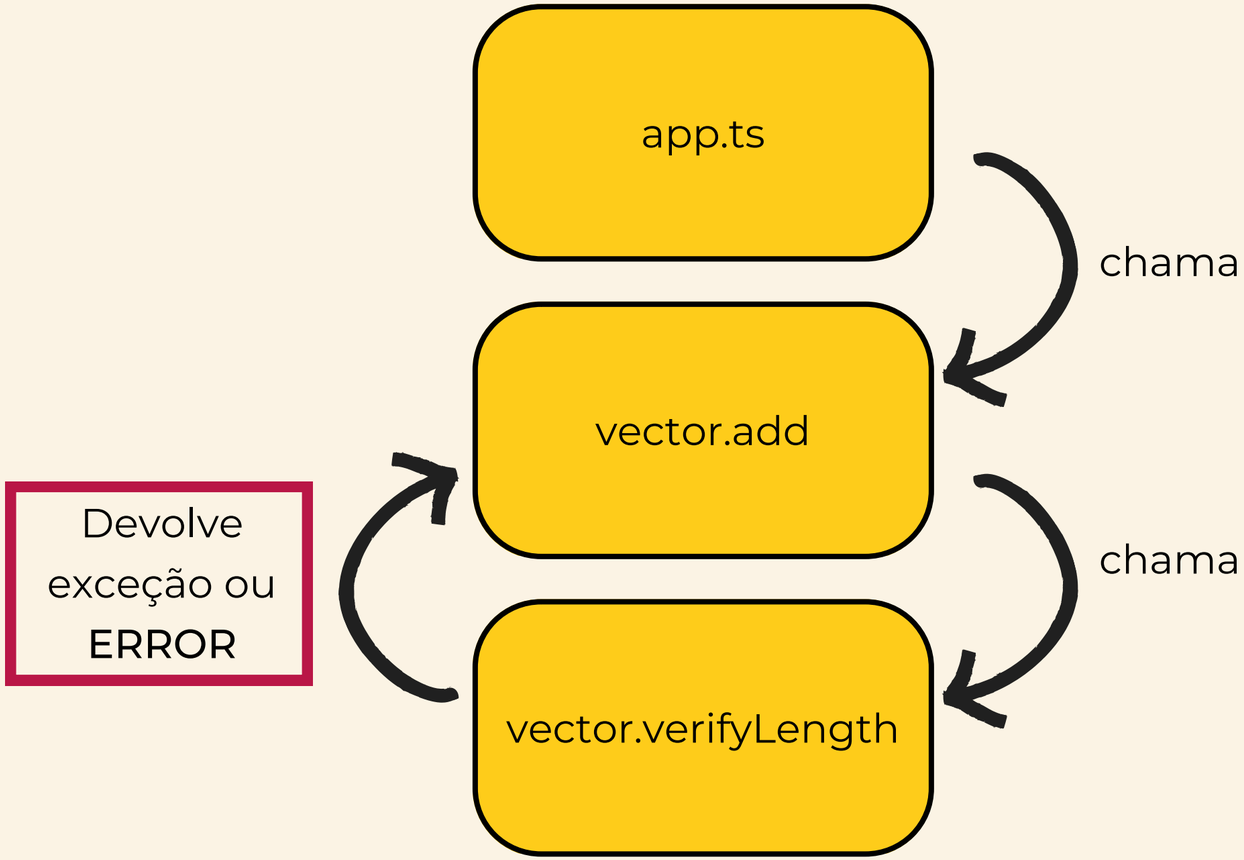
EXCEÇÕES



O que são?

Evento que ocorrem durante a execução de programas que fogem do fluxo esperado de suas instruções.

EXCEÇÕES



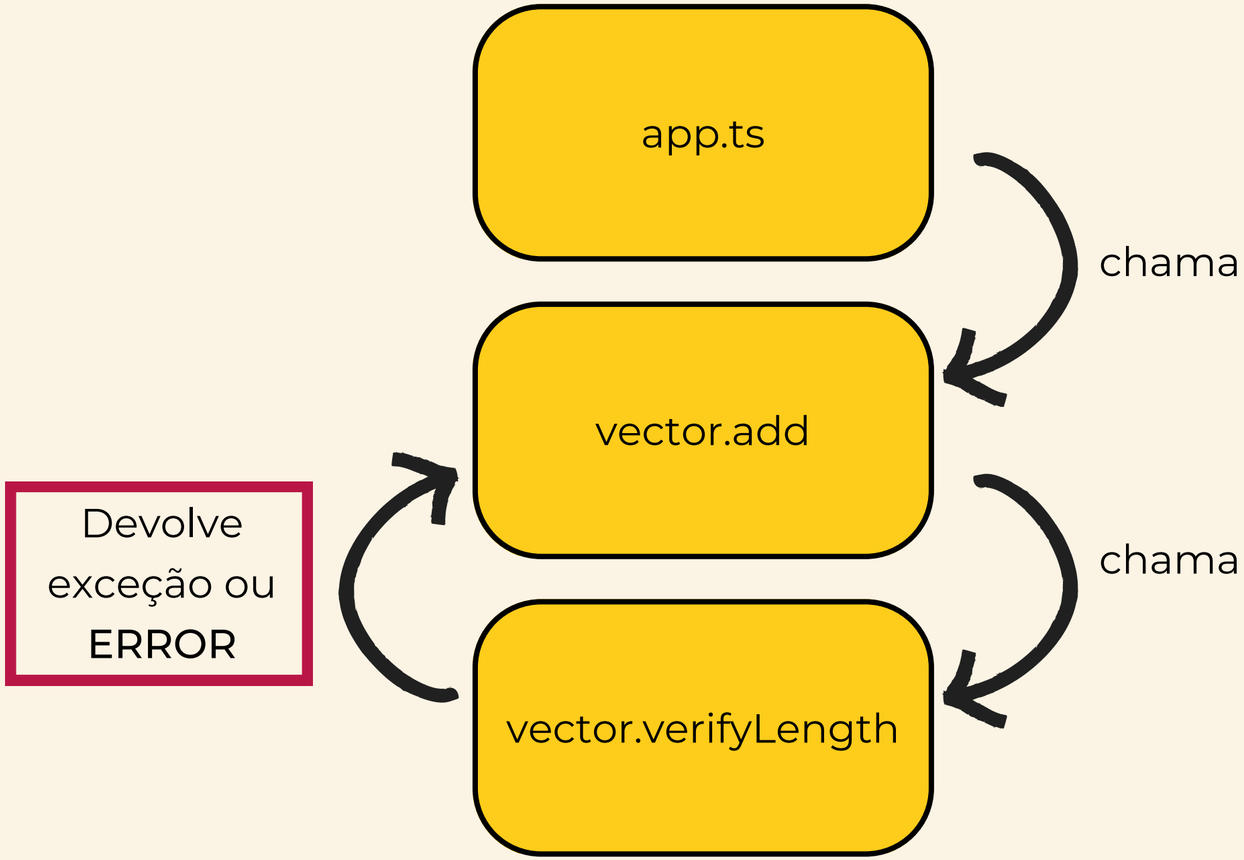
O que são?

Evento que ocorrem durante a execução de programas que fogem do fluxo esperado de suas instruções.

Pra que servem?

Ao disparar uma exceção, o programa permite que outras partes do próprio programa ou até mesmo outros programas compreendam a falha no fluxo esperado

EXCEÇÕES



O que são?

Evento que ocorrem durante a execução de programas que fogem do fluxo esperado de suas instruções.

Pra que servem?

Ao disparar uma exceção, o programa permite que outras partes do próprio programa ou até mesmo outros programas compreendam a falha no fluxo esperado

Como tratar?

São disparadas a partir do uso do termo "throw" e podem ser identificadas e devidamente tratadas usando "try...catch"

EXCEÇÕES

```
export class Vector {
  components: Array<number>;
  // implementação do constructor
  // constructor(components: number[])

  verifyLength(vector:Vector): boolean {
    if(vector.components.length !=
         this.components.length) {
      throw new Error("Tamanhos diferentes")
    }
  }
  add(vector:Vector): Vector {
    this.verifyLength(vector)
    let output: number[] = []
    for (const i in this.components) {
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

```
// código chamando a classe Vector

let vector1: Vector = new Vector([1,2,3])
let vector2: Vector = new Vector([1,2,3,4])

console.log(vector1.add(vector2))
// Erro é apresentado no console
```

EXCEÇÕES

```
export class Vector {
  components: Array<number>;
  // implementação do constructor
  // constructor(components: number[])

  verifyLength(vector:Vector): boolean {
    if(vector.components.length !=
         this.components.length) {
      throw new Error("Tamanhos diferentes")
    }
  }
  add(vector:Vector): Vector {
    this.verifyLength(vector)
    let output: number[] = []
    for (const i in this.components) {
      output.push(this.components[i] +
                  vector.components[i])
    }
    return new Vector(output);
  }

  // implementação de outros métodos
}
```

```
C:\Program Files\nodejs\node.exe .\out\app.js
Uncaught Error Error: Arrays de tamanhos diferentes
```

```
// código chamando a classe Vector

let vector1: Vector = new Vector([1,2,3])
let vector2: Vector = new Vector([1,2,3,4])

console.log(vector1.add(vector2))
// Erro é apresentado no console
```

```
// código chamando a classe Vector

let vector1: Vector = new Vector([1,2,3])
let vector2: Vector = new Vector([1,2,3,4])

try {
  console.log(vector1.add(vector2))
  console.log("Deu tudo certo!")
} catch (e) {
  console.log("Ops! Algum erro ocorreu")
} finally {
  console.log("Sempre vou ser executado")
}
```

EXCEÇÕES CUSTOMIZADAS

```typescript
interface Error {
    name: string;
    message: string;
    stack?: string;
}

interface ErrorConstructor {
    new(message?: string): Error;
    (message?: string): Error;
    readonly prototype: Error;
}
```

```typescript
export class ApplicationError extends Error {
  name: string = "Erro de aplicação";

  constructor(message:string) {
    super(message);
  }

  toString(){
    console.log(`${this.name}: ${this.message}`)
  }
}
```

```
C:\Program Files\nodejs\node.exe .\out\app.js
Uncaught ApplicationError Erro de aplicação: Arrays de tamanhos diferentes
```

EXCEÇÕES CUSTOMIZADAS

```
export class ApplicationError extends Error {
  name: string = "Erro de aplicação";

  constructor(message:string) {
    super(message);
  }

  toString(){
    console.log(`${this.name}: ${this.message}`)
  }
}
```

```
// código chamando a classe Vector

let vector1: Vector = new Vector([1,2,3])
let vector2: Vector = new Vector([1,2,3,4])

try {
  console.log(vector1.add(vector2))
  console.log("Deu tudo certo!")
} catch (e) {
  if(e instanceof ApplicationError) {
    console.log("Erro app")
  } else {
    console.log("Erro generico")
  }
  console.log("Ops! Algum erro ocorreu")
} finally {
  console.log("Sempre vou ser executado")
}
```

"instanceof" utilizado para identificar se o objeto é uma instancia da classe ou interface informada

TIPOS GENÉRICOS

TIPOS GENÉRICOS

```
// declaração Array

let array: Array<number> = new Array(10)
```

Tamanho do Array

Definição do tipo a ser utilizado

TIPOS GENÉRICOS

```
export interface IList {
  add(item: string): void;
  remove(): void;
  get(index: number): void;
  set(index: number, item: string): void;
  contains(item: string): boolean;
  size(): number;
  isEmpty(): boolean;
}
```

```
export interface IQueue {
  enqueue(item: string): void;
  dequeue(): string;
  size(): number;
  isFull(): boolean;
}
```

```
export interface IStack {
  push(item: string): void;
  pop(): string;
  size(): number;
  isFull(): boolean;
}
```

TIPOS GENÉRICOS

```
export interface IStack<T> {
  push(item: T): void;
  pop(): T;
  size(): number;
  isFull(): boolean;
}
```

TIPOS GENÉRICOS

```
export interface IStack<T> {
  push(item: T): void;
  pop(): T;
  size(): number;
  isFull(): boolean;
}
```

Declaração do tipo genérico

TIPOS GENÉRICOS

```
export interface IStack<T> {
  push(item: T): void;
  pop(): T;
  size(): number;
  isFull(): boolean;
}
```

Argumento do tipo genérico declarado

Declaração do tipo genérico

TIPOS GENÉRICOS

```
export interface IStack<T> {
  push(item: T): void;
  pop(): T;
  size(): number;
  isFull(): boolean;
}
```

Argumento do tipo genérico declarado

Declaração do tipo genérico

Retorno do tipo genérico declarado

TIPOS GENÉRICOS

```
export interface IStack<T> {
  push(item: T): void;
  pop(): T;
  size(): number;
  isFull(): boolean;
}
```

```
export class Stack<T> implements IStack<T> {
  push(item: T): void {
  }
  pop(): T {
  }
  size(): number {
  }
  isFull(): boolean {
  }
}
```

TIPOS GENÉRICOS

```
export interface IStack<T> {
  push(item: T): void;
  pop(): T;
  size(): number;
  isFull(): boolean;
}
```

O precisa ter a mesma referência no contexto da classe. Pode ser diferente da referência usada nas classes/interfaces

```
export class Stack<E> implements IStack<E> {
  push(item: E): void {
  }
  pop(): E {
  }
  size(): number {
  }
  isFull(): boolean {
  }
}
```

TIPOS GENÉRICOS

O precisa ter a mesma referência no contexto da classe. Pode ser diferente da referência usada nas classes/interfaces

```
export interface IStack<T> {
  push(item: T): void;
  pop(): T;
  size(): number;
  isFull(): boolean;
}
```

```
export class Stack<Type> implements IStack<Type> {
  push(item: Type): void {
  }
  pop(): Type {
  }
  size(): number {
  }
  isFull(): boolean {
  }
}
```

```
// declaração stack

let stack: Stack<number> = new Stack([1,2,3]);
```

TIPOS GENÉRICOS

DROGAS MAIS PESADAS

TIPOS GENÉRICOS - MÚLTIPLOS TIPOS

```
export interface IPair<T,E> {
  left(): T;
  right(): E;
}
```

```
export class Pair<T,E> implements IPair<T,E> {
  left(): T {
  }
  right(): E {
  }
}
```

```
// declaração pair

let pair: Pair<number,string> = new Pair(1,"Valor");
```

DESAFIO

Use exceções caso as estruturas estejam cheias e seja executada a tentativa de nova inserção

Implementar as interfaces de estruturas de dados apresentadas.

```
export interface IList<T> {
  add(item: T): void;
  remove(): void;
  get(index: number): void;
  set(index: number, item: T): void;
  contains(item: T): boolean;
  size(): number;
  isEmpty(): boolean;
}
```

```
export interface IQueue<T> {
  enqueue(item: T): void;
  dequeue(): T;
  size(): number;
  isFull(): boolean;
}
```

```
export interface IPair<T,E> {
  left(): T;
  right(): E;
}
```

```
export interface IStack<T> {
  push(item: T): void;
  pop(): T;
  size(): number;
  isFull(): boolean;
}
```

OBRIGADO!

rafael_c_alves@hotmail.com

RAFAEL CORRÊA