Objetivo:

I. Exceções.

Observação: antes de começar, crie um projeto para reproduzir os exemplos. Dica, use os passos da Aula 2 para criar o projeto de exemplo.

I. Exceções

O tratamento de exceções é o processo de captura e tratamento de erros que ocorrem durante a execução de um programa. O tratamento de exceções é realizado usando o bloco `try`...`catch`. O bloco `try` é usado para envolver o código que pode gerar uma exceção, e o bloco `catch` contém o código que será executado se um erro for capturado. No exemplo a seguir, a chamada da função `imc` precisa estar em um bloco `try`, pelo fato dela lançar exceção. Se ocorrer a exceção, o bloco `catch` será chamado para tratar a exceção.

O bloco `finally` contém o código que será executado independentemente de ocorrer ou não uma exceção. O bloco `finally` é opcional e é usado para liberar recursos ou executar ações que devem ser realizadas mesmo em caso de exceção.

```
try {
    const resultado = imc(70, 0);
    // esta instrução não será executa se for lançada uma exceção
    console.log("Resultado:", resultado);
} catch (e:any) { // o erro lançado será recebido no parâmetro e
   // esta instrução será executa se for lançada uma exceção
   // a propriedade message possui a mensagem do objeto Error
    console.log("Exceção:", e.message);
} finally {
    console.log("Passa por aqui");
}
```

A instrução `throw` é usada para lançar uma exceção em tempo de execução. Um erro em tempo de execução não ocorre em tempo de compilação – logo não pode ser detectado pelas ferramentas de análise estática do VS Code.

A exceção é criada usando um objeto do tipo `Error`. Um objeto do tipo `Error` não deve ser retornado por `return`, mas pela instrução `throw`.

A instrução `throw` indicará que ocorreu um erro ou uma condição excepcional e interromperá o fluxo normal do programa, transferindo o controle para o bloco `catch` mais próximo que pode lidar com a exceção.

No exemplo a seguir, qualquer instrução `throw` interromperá o fluxo e fará com que a instrução `return` não seja executada.

```
function imc(peso: number, altura: number): number {
    if (peso <= 0) {
        throw new Error("Peso incorreto");
    }
    else if (altura <= 0) {
        throw new Error("Altura incorreta");
    }
    // esta instrução não será executa se for lançada uma exceção
    return peso / altura**2;
```

```
}
```

> O uso de exceções deve ser feito com cuidado e somente em situações apropriadas. É recomendável lançar exceções apenas para erros e situações excepcionais e não abusar do uso de exceções para controlar o fluxo do programa.

Podemos definir nossos próprios tipos de dados `Error`, basta estender a classe `Error`. No exemplo a seguir o tipo `ImcError` foi personalizado para ter a instrução que gerou a exceção.

```
class ImcError extends Error {
    constructor(message:string, public instrucao:string) {
        super(message);
    }
}

function imc(peso: number, altura: number): number {
    if (peso <= 0) {
        throw new ImcError("Peso incorreto", `${peso}<=0`);
    }
    else if (altura <= 0) {
        throw new ImcError("Altura incorreta", `${altura}<=0`);
    }
    return peso / altura**2;
}

try {
    const resultado = imc(70, 0);
    console.log("Resultado:", resultado);
} catch (error:any) {
    console.log("Exceção:", error.message);
    console.log("Instrução:", error.instrucao);
} finally {
    console.log("Passa por aqui");
}
```

Exercícios

Observação: crie um único projeto para fazer todos os exercícios, assim como você fez nas aulas anterior.

Exercício 1: Adicionar instruções try ... catch no código a seguir para que ambas as chamadas sejam executadas.

Requisito: a função calcular não poderá ser modificada.

```
function calcular(a: any, b: any): number {
    if (typeof a === 'number' && typeof b === 'number') {
        return a + b;
```

Exemplo de saída:

```
PS D:\aula6> npm run um
> aula6@1.0.0 um
> ts-node ./src/exercicio1
Os parâmetros precisam ser números
Soma: 3
Fim do programa
```

```
    }
    throw new Error('Os parâmetros precisam ser números');
}

console.log("Soma:", calcular('oi', 2));
console.log("Soma:", calcular(1, 2));
console.log("Fim do programa");
```

Exercício 2: Tratar a exceção fazendo o código colocar todos os números aleatórios no array.

Requisito: a função aleatorio não poderá ser modificada.

```
function aleatorio():number{
    const nro = Math.floor(Math.random()*10);
    if( nro%2 === 0 ){
        return nro;
    }
    throw new Error("Número ímpar");
}

function arrayAleatorio(quantidade: number): number[] {
    const array:number[] = [];
    for( let i = 0; i < quantidade; i++ ){
        array.push(aleatorio());
    }
    return array;
}

const vet = arrayAleatorio(8);
console.log("Array:", vet);
console.log("Fim do programa");
```

Exemplo de saída:

```
PS D:\aula6> npm run dois
> aula6@1.0.0 dois
> ts-node ./src/exercicio2
Array: [
  2, 6, 0, 4,
  2, 8, 4, 2
]
Fim do programa
```

Exercício 3: Tratar a exceção fazendo os números ímpares serem multiplicados por 10 e colocados no array.

Requisito: a função aleatorio não poderá ser modificada.

```
class AleatorioError extends Error {
    constructor(message:string, public nro:number){
        super(message);
    }
}

function aleatorio():number{
    const nro = Math.floor(Math.random()*10);
    if( nro%2 === 0 ){
        return nro;
```

Exemplo de saída:

```
PS D:\aula6> npm run tres
> aula6@1.0.0 tres
> ts-node ./src/exercicio3
Array: [
  50, 6, 10, 8,
   6, 4, 50, 8
]
Fim do programa
```

```
    }
    throw new AleatorioError("Número ímpar", nro);
}

function arrayAleatorio(quantidade: number): number[] {
    const array:number[] = [];
    for( let i = 0; i < quantidade; i++ ){
        array.push(aleatorio());
    }
    return array;
}

const vet = arrayAleatorio(8);
console.log("Array:", vet);
console.log("Fim do programa");
```

Exercício 4: Tratar a exceção fazendo o programa gerar o resultado mostrado ao lado. Requisito: a classe Pilha não poderá ser modificada.

Exemplo de saída:

```
PS D:\aula6> npm run quatro
> aula6@1.0.0 quatro
> ts-node ./src/exercicio4
Maria
Luiz
Pedro
Ana
Pilha vazia
Fim do programa
```

```
class Pilha<T> {
    private items: T[] = [];

    push(item:T):void {
        this.items.push(item);
    }

    pop():T {
        const item = this.items.pop();
        if( item === undefined){
            throw Error("Pilha vazia");
        }
        return item;
    }
}

const nomes = ["Ana","Pedro","Luiz","Maria","Inês","José"];
const pilha = new Pilha<string>();
for(let i = 0; i < nomes.length; i++){
    pilha.push(nomes[i]);
}
let item = pilha.pop();
while( item ){
    console.log(item);
    item = pilha.pop();
}
console.log("Fim do programa");
```

Exercício 5: Refazer o Exercício 4 usando a classe Pilha a seguir. Observe que a pilha aceita no máximo 5 elementos.

Requisito: a classe Pilha não poderá ser modificada.

```
class Pilha<T> {
    private items: T[] = [];

    push(item: T): void {
        if (this.items.length == 5) {
            throw new Error("Pilha cheia");
        }
        this.items.push(item);
    }

    pop():T {
        const item = this.items.pop();
        if( item === undefined){
            throw Error("Pilha vazia");
        }
        return item;
    }
}
```

Exemplo de saída:

```
PS D:\aula6> npm run cinco
> aula6@1.0.0 cinco
> ts-node ./src/exercicio5
Pilha cheia
Inês
Maria
Luiz
Pedro
Ana
Pilha vazia
Fim do programa
```