Programação Orientada a Objetos

Exceções – Parte I

Ely – ely.miranda@ifpi.edu.br

Erros

- Os softwares devem estar preparados tratar e se recuperar de erros;
- Erros devem ser tratados sejam eles quaisquer;
- Tratamento de erros leva à robustez.

Robustez

- Capacidade de um software executar suas funções mesmo em condições anormais;
- Prever formas de execução não esperadas e ser capaz de resistir e se recuperar de falhas;
- Robustez leva à confiabilidade.

Alternativas a tratamento de erros

- Desconsiderar operação;
- Exibir mensagem de erro;
- Retornar um código de erro;
- Exceções.

Exemplo

```
class Conta {
    private _numero: String;
    private _saldo: number;

    constructor(numero: String, saldoInicial: number) {
        this._numero = numero;
        this._saldo = saldoInicial;
    }

    get saldo() {
        return this._saldo;
    }

   sacar(valor: number): void {
        this._saldo = this._saldo - valor;
    }
}
```

Exemplo

```
class Conta {

    //...como evitar que o saldo fique negativo?
    sacar(valor: number): void {
        this._saldo = this._saldo - valor;
    }

 }

 let conta: Conta = new Conta("1", 0.0);
 conta.sacar(200);
 console.log(conta.saldo); // -200
```

Desconsiderar a operação

```
class Conta {

    //...
    sacar(valor: number): void {
      if (this._saldo >= valor) {
        this._saldo = this._saldo - valor;
      }
    }

 }
```

Desconsiderar a operação

- Problema: ficamos na incerteza do sucesso ou não da operação.

```
let conta: Conta = new Conta();
conta.sacar(1000000000);
// A chamada ao método "sacar" funcionou?
```

Exibir mensagem de erro

```
class Conta {

    //...
    sacar(valor: number): void {
      if (this._saldo >= valor) {
        this._saldo = this._saldo - valor;
      } else {
        console.log("Saldo Insuficiente.");
      }
    }

}
```

Exibir mensagem de erro

- Ficamos atrelados à interface texto;
- E se estivéssemos usando interface gráfica?
  - A mensagem não seria notada, pois não se iria olhar o prompt/console.

Retornar um código de erro

```
class Conta {

    //...
    sacar(valor: number): boolean {
      if (this._saldo >= valor) {
        this._saldo = this._saldo - valor;
        return true;
      } else {
        return false;
      }
    }

}
```

Retornar um código de erro

```
class Conta {
   //...
   transferir(conta: Conta, valor: number): boolean {
        let sacou: boolean = conta.sacar(valor);
        if (sacou) {
            this.depositar(valor);
            return true;
        } else
            return false;
    }

}
```

Retornar um código de erro

```
class Banco {
    private _contas: Conta[] = [];

    //...
    sacar(numero: String, valor: number): number {
        let c!: Conta = this.consultar(numero);
        if (c != null) {
            let sacou: boolean = c.sacar(valor);
            if (sacou)
                return 1;
            else
                return 2;
        } else
            return 3;
    }
}
```

Retornar um código de erro

- Temos que testar o valor de retorno para saber o que houve;
- Quando o método já retorna valores, temos que reservar alguns para representar erros;
- Em C é muito comum usarmos “valores menores que 0 para erros”.

Retornar um código de erro

- Isto é considerada uma má prática:
  - Uso de flags ou números mágicos;
  - Pode levar à mudança do tipo de retorno do método:
    - Um método com retorno boolean, pode ter que retornar inteiros: 1 (true), 0 (false) e -1 (erro);

Retornar um código de erro

- Se o método já retorna um número calculado:
  - É difícil reservar códigos de erro que não se confundam um possível valor calculado;
  - O valor retornado não é intuitivo e carece de documentação para o entendimento.

Exceções

- Em linguagens modernas erros são sinalizados por Exceções:
  - Classes especiais que contém informações sobre o tipo de erro gerado;
  - Causam a interrupção da execução do código se não tratados.

Exceções

- A partir da exceção, nada mais é executado, a menos que o erro seja tratado;
- Com isso, um erro não causa implicações mais graves;
- Toda exceção deve ter uma mensagem associada que facilite a interpretação do erro.

Lançando exceções

- Podemos lançar exceções quando encontrarmos situações de erros;
- Lançar uma exceção é instanciar uma classe específica:
  - usa-se a palavra reservada throw ;
  - passa-se uma mensagem de erro.

A classe Error

- Podemos lançar exceções já existentes no Typescript ou criar nossas próprias exceções;
- A classe mais básica do TypeScritpt pra exceções é a classe Error.

```
throw new Error('Algo deu errado');
```

A classe Error

- O código após o lançamento de uma exceção

```
throw new Error('Algo deu errado');
console.log('Hello world');
//erro de compilação, o código não é alcançável
```

Lançando exceções

```
class Conta {

    //...
    sacar(valor: number): void {
        if (this._saldo < valor) {
            throw new Error('Saldo insuficiente.')
        }

        this._saldo = this._saldo - valor;
    }

}
```

Lançando exceções

```
let conta: Conta = new Conta('1', 50);

conta.sacar(100);

console.log('código não alcançável');
```

Consequências diretas

- O débito não é realizado caso caia na exceção;
- O código fica protegido para operações que dependam do sucesso do saque;
- O código de else não é necessário, pois a exceção interrompe o código a partir da sua instanciação.

Programação Orientada a Objetos

Exceções – Parte I

Ely – ely.miranda@ifpi.edu.br