DATA: 09 DE AGOSTO DE 2024
PROFESSOR: EVERSON SOUSA

TEMA DA AULA: ASSOCIAÇÃO

Vamos lá! Uma associação é um relacionamento entre classes que permite que uma instância de uma classe faça a outra realizar uma ação.

É como dizer para a aplicação que um objeto de um tipo está conectado, ou depende de um objeto de outro tipo.

Exemplos:
- Objetos de uma classe **Person** possuem um atributo que é um objeto da classe **Address**.
- Objetos de uma classe **Product** possuem um atributo que é um array de objetos da classe **Category**.

Vamos aos códigos:

PASSO 1:

Utilizando os exemplos acima, iremos primeiro criar um arquivo JS com o nome de **Address.js** e com uma estrutura bem simples iremos criar uma classe chamada **Address** (Endereço):

```
class Address {
  constructor(street, number, neighborhood, city, state) {
    this.street = street
    this.number = number
    this.neighborhood = neighborhood
    this.city = city
    this.state = state
  }
```

Temos nessa estrutura as atribuições **street** (rua), **number** (número), **neighborhood** (bairro), **city** (cidade), **state** (estado).

Depois de criar nossa “receita de bolo”, ou melhor dizendo, nossa classe Address, iremos criar um método que terá uma mensagem de retorno quando for executado no Shell (ou console):

```
  fullAddress() {
    return `Rua ${this.street}, nº ${this.number}, ${this.neighborhood} -
${this.city}/${this.state}`
  }
```

OBS: não esquecer que os métodos ficam dentro da classe e abaixo do método constructor!

Então agora temos um método que retornará a seguinte mensagem (por exemplo): ***Rua Elisa Sampaio, nº 479, Centro – Paramoti/CE***.

O código completo ficará dessa forma:

```
1  class Address {
2    constructor(street, number, neighborhood, city, state) {
3      this.street = street
4      this.number = number
5      this.neighborhood = neighborhood
6      this.city = city
7      this.state = state
8    }
9
10   fullAddress() {
11     return `Rua ${this.street}, nº ${this.number}, ${this.neighborhood} -
   ${this.city}/${this.state}`
12   }
13 }
14
15 module.exports = Address
```

PONTO IMPORTANTE!

Podemos ver que na linha 15 tem uma linha de código "*module.exports = Address*". Esse código indica que iremos exportar as informações desse nosso arquivo Address.js para poder fazer a associação. Melhor dizendo, é algo similar o que fazemos no HTML quando linkamos o CSS dentro da tag <head>.

PASSO 2:

Depois de criarmos o script Address.js, iremos criar outro script chamado **Person.js**, obviamente, criando uma classe com nome Person (pessoa). Essa classe será a mais simples possível para não dificultar nosso aprendizado sobre associações:

```
1  class Person {
2    constructor(name, address) {
3      this.name = name
4      this.address = address
5    }
6  }
7
8  module.exports = Person
```

Nossa classe Person só irá possuir os atributos **name** (nome) e **address** (endereço).

Nesse momento, precisamos ter bastante atenção no atributo address, pois nosso intuito aqui é justamente associar o atributo address da classe Person com as informações do script Address.

PASSO 3:

Depois que terminarmos de criar os dois arquivos, **Address.js** e **Person.js**, iremos criar as devidas associações. Iremos nesse momento criar um arquivo chamado somente de **index.js**. Esse script será a peça principal da aula de hoje, pois é nesse script que conterá o exemplo da nossa associação.

De início, você lembra que no final de cada script tinha uma linha de código com "module.exports = Person"? Essa linha de código está EXPORTANDO informações. Então já que ela está enviando informações, precisando criar um código onde a gente possa receber as tais informações. **Nas primeiras linhas do nosso script index.js iremos escrever**:

```
1   const Address = require('./Address')
2   const Person = require('./Person')
```

Esses códigos indicam que estamos suscetíveis a receber informações dos arquivos Address.js e Person.js. Nesse caso, iremos criar uma variável de qualquer nome, depois iremos linkar os endereços dos tais scripts. Não é necessário colocar o .js nos endereços.

Agora nesse momento iremos criar nossas instâncias. Lembra o que é instância né? Instância é aquele código onde você irá colocar as informações que pedem nos parâmetros das classes. Melhor com exemplo:

```
1   const Address = require('./Address')
2   const Person = require('./Person')
3
4   const addr = new Address('Rua Elisa Sampaio', 479, 'Centro', 'Paramoti', 'CE')
```

Aqui estamos criando uma instância com o nome de addr. Podemos ver que a instância está recebendo a classe Address (new Address), ou seja, já começamos ver a associação aqui! Nesse momento não precisamos criar uma instância no próprio arquivo Address.js, essa instância está sendo criada no arquivo index.js, já que estamos importando informações devido a linha 1.

Mas não é só isso! Também podemos fazer o seguinte:

```
1   const Address = require('./Address')
2   const Person = require('./Person')
3
4   const addr = new Address('Rua Elisa Sampaio', 479, 'Centro', 'Paramoti', 'CE')
5   const john = new Person('John', addr)
```

Na linha 5 podemos ver outra instância sendo criada, mas tem uma informação diferente aqui. Lembra quais atribuições temos em nossa classe Person? Volte no arquivo Person.js e verá que só pedimos o name e address.

Então obviamente, nossa instância deve conter 2 valores, devido a ter 2 parâmetros. O primeiro sendo name, e o segundo sendo address. Só que aqui, não iremos escrever de forma fixa esses valores. O segundo valor, que seria address, irá puxas todas as

informações que contêm na instância new Address (lnha 4). Então name recebe “John”, e o address recebe todas as informações da variável da linha 5 addr. Legal né?

Nessa situação, economizamos bastante escrita, além de deixar nosso script mais dinâmico.

Agora iremos ver como está o index.js completo:

```
1  const Address = require('./Address')
2  const Person = require('./Person')
3
4  const addr = new Address('Rua Elisa Sampaio', 479, 'Centro', 'Paramoti', 'CE')
5  const john = new Person('John', addr)
6
7  console.log(john)
8  console.log(john.address.fullAddress())
```

Na linha 7 e 8 do script index.js temos 2 consoles. O primeiro console irá mostrar as informações da instância john. Já o segundo console, ao chamar a variável john (linha 5), podemos ir mais a frente e chamar o address. Esse address vem do script Person.js, linha 4 (this.address = address). E fullAddress() vem do método do script Address.js, de mesmo nome. Ou seja, nossa associação percorreu o arquivo index.js, depois passou para o script Person.js lá no atributo this.address = address, e esse atributo está recebendo informações do script Address.js, onde por via, está executando o método fullAddress(), que será executado nosso return.

Vamos ver como ficou o resultado!

No Shell, iremos acessar a pasta com cd e o caminho. Depois iremos executar “node index.js”. Iremos executar o index.js porque esse agora é o arquivo principal.

O resultado ficou assim:

```
~/.../09-ago/5-associacao$ node index.js
Person {
  name: 'John',
  address: Address {
    street: 'Rua Elisa Sampaio',
    number: 479,
    neighborhood: 'Centro',
    city: 'Paramoti',
    state: 'CE'
  }
}
Rua Rua Elisa Sampaio, nº 479, Centro - Paramoti/CE
~/.../09-ago/5-associacao$
```

Então podemos ver que no Shell apareceu o objeto Person, e em address, mostrou as informações do script Address.js.

Já abaixo, mostrou o return do nosso método fullAddress do script Address.js.

Caso tenha ficado confuso, volte, leia novamente e escreva os códigos um por um. Caso tenha entendido como funciona a associação, meus parabéns!

Chegando aqui, não podemos terminar sem um exercício né? Esse exercício será postado formatado no GitHub, mas por enquanto você o encontrará aqui. Lá vai!

Data de entrega: até dia 15 de agosto, às 23h59. (Melhor começar agora pois aqui as coisas começam a ficar "interessantes"...)

PROJETO 2 – SIMULANDO UM BLOG COM CLASSES

Crie uma aplicação javascript que simule um funcionamento básico de blog utilizando classes e associações.

Você deverá criar uma classe Post e uma classe Comment, onde uma instância de Post poderá receber vários comentários. As instâncias de Post também devem ter um método específico para adição de comentários nelas.

Você também deverá criar uma classe Author, e os objetos da classe Post também devem possuir um objeto da classe Author, que é o autor do post. Além disso, os objetos da classe Author também devem possuir um array de posts (objetos da classe Post) e um método específico para criação de posts, que deverá criar uma instância utilizando aquele próprio autor, incluir o post no array e então retornar o post criado.

Vamos lá e boas práticas!