

DATA: 16 DE AGOSTO DE 2024
PROFESSOR: EVERSON SOUSA

TEMA DA AULA: HERANÇA

Herança é um recurso que permite criar classes que incluem, ou "herdam", todos os atributos e métodos de uma outra classe.

A classe que é herdada damos o nome de **superclass** (ou classe mãe), e a classe que herda, o nome de **subclass** (ou classe filha).

Uma classe filha possui todos os atributos e métodos da classe mãe, mas também pode possuir os seus próprios (que serão exclusivos dela).

Exemplos:

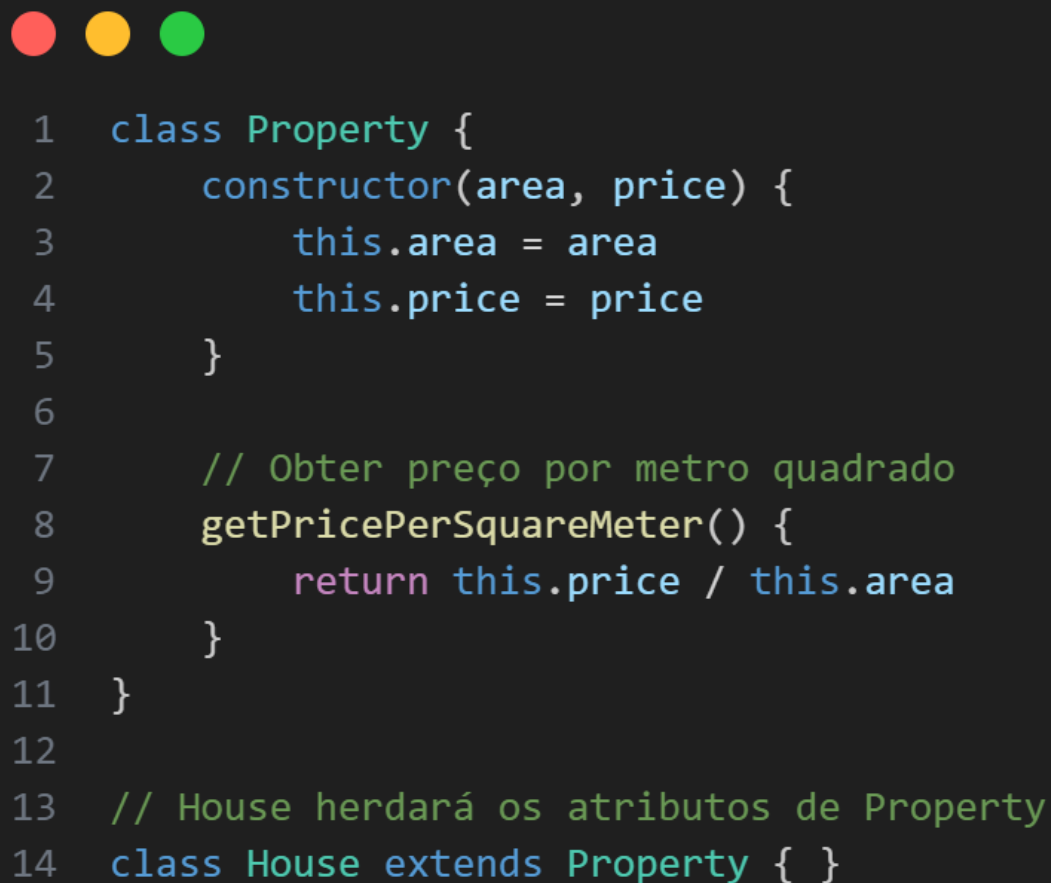
- Imagine uma classe **User** que possui seus atributos e métodos. Poderíamos ter classes **AdminUser**, **EditorUser** e **StandardUser** que são todas filhas de **User**, mas cada uma também possui seus próprios atributos e métodos.
- Ou ainda uma classe **Property** (como um imóvel) pode ser a classe mãe das classes filhas **House** e **Apartment**.

Vamos aos códigos:

Criamos uma classe chamada **Property** como no exemplo acima, além de adicionar um método chamado **getPricePerSquareMeter**:

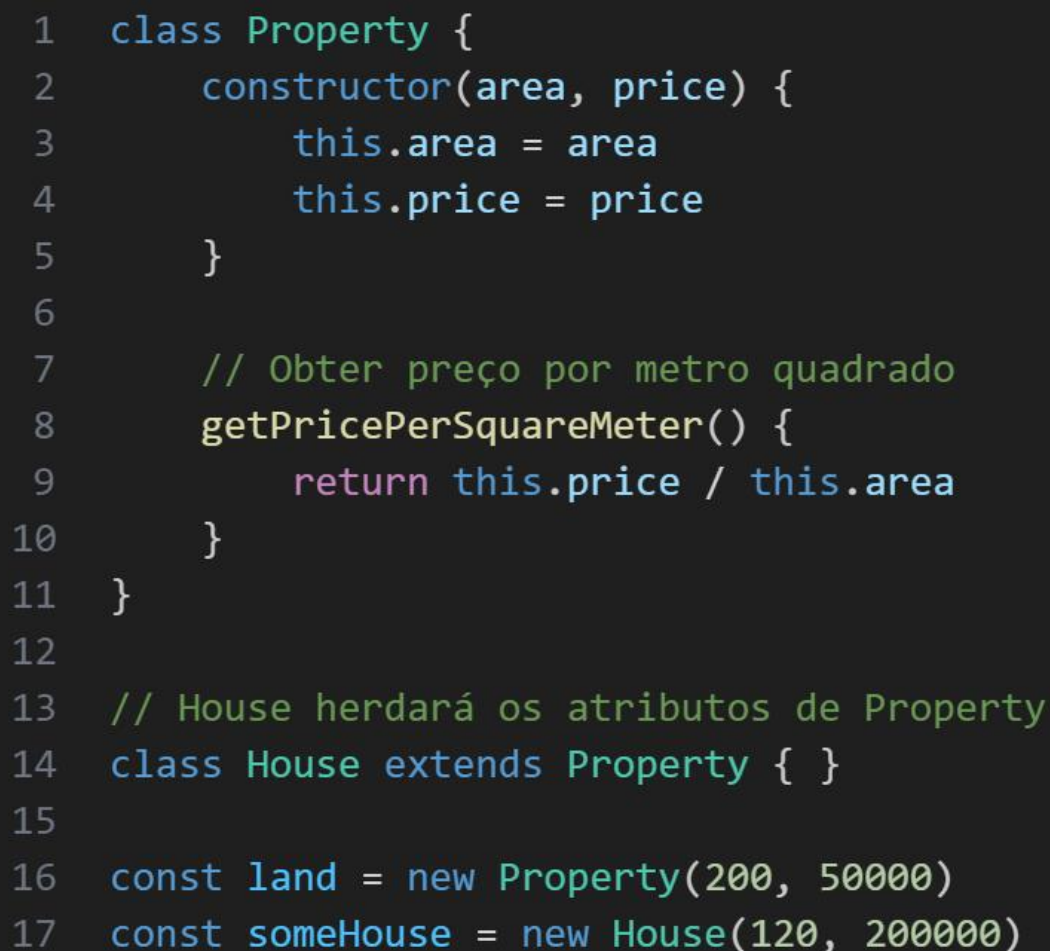
```
1  class Property {  
2      constructor(area, price) {  
3          this.area = area  
4          this.price = price  
5      }  
6  
7      // Obter preço por metro quadrado  
8      getPricePerSquareMeter() {  
9          return this.price / this.area  
10     }  
11 }
```

Agora, abaixo da classe **Property**, criamos uma classe chamada de **House**. Só que nessa situação, iremos adicionar um comando chamado “**extends**”, indicando que é uma extensão e que a nova classe **House** herdará todos as propriedades da classe **Property**. Confira o código até o momento com essa adição:



```
1  class Property {
2      constructor(area, price) {
3          this.area = area
4          this.price = price
5      }
6
7      // Obter preço por metro quadrado
8      getPricePerSquareMeter() {
9          return this.price / this.area
10     }
11 }
12
13 // House herdará os atributos de Property
14 class House extends Property { }
```

No exemplo, não precisamos criar o **constructor** na classe **House** porque essa nova classe já herda todas as propriedades da classe **Property**. Vamos conferir como ficará o código com as instâncias:



```
1  class Property {
2      constructor(area, price) {
3          this.area = area
4          this.price = price
5      }
6
7      // Obter preço por metro quadrado
8      getPricePerSquareMeter() {
9          return this.price / this.area
10     }
11 }
12
13 // House herdará os atributos de Property
14 class House extends Property { }
15
16 const land = new Property(200, 50000)
17 const someHouse = new House(120, 200000)
```

Podemos ver aqui que a variável **land** está instanciando **Property**, que pede em seus parâmetros **area** e **price**. Já a variável **someHouse** está instanciando os mesmo parâmetros de **Property**, ou seja, **area** e **price**, fazendo assim, não ter a necessidade de repetir as propriedades em na nova classe.

Iremos agora criar uma classe chamada **Apartment** que herdará as propriedades de **Property**. Só que nesse exemplo, queremos adicionar mais propriedades, ou seja, pegaremos as que já tem no **Property** e adicionaremos **number**. Note que no **constructor**, precisamos repetir as propriedades de **Property**, mas não necessitaremos de transcrever o **this.area** e **this.price**:

```
1 class Apartment extends Property {
2   constructor(number, area, price) {
3     super(area, price) // serve para não repetir o this.area, this.price
4     this.number = number
5   }
6
7   getFloor() {
8     return this.number.slice(0, -2)
9   }
10 }
```

Na linha 3 podemos perceber a adição de um novo método, chamado de **super**. O método **super** serve para que não precise repetir o **this.area** e o **this.price**, fazendo assim, só um meio de validação das propriedades que estão sendo herdadas. Após isso, a única coisa que precisamos fazer é adicionar o **this.number** pois é uma propriedade nova. Vamos instanciar essa classe para continuar o exemplo:

```
1 // Adicionando mais coisas em classes herdadas
2 class Apartment extends Property {
3   constructor(number, area, price) {
4     super(area, price) // serve para não repetir o this.area, this.price
5     this.number = number
6   }
7
8   getFloor() {
9     return this.number.slice(0, -2)
10  }
11 }
12
13 const apt = new Apartment('201', 100, 160000)
14 console.log(apt)
15 console.log(apt.getFloor())
16 console.log(apt.getPricePerSquareMeter())
```

Na variável **apt**, ao instanciar, podemos reparar que estamos informando os três parâmetros, que são **number**, **area** e **price**. No console da linha 14 mostraremos o resultado da instância, no console da linha 15 iremos executar o método **getFloor** que faz parte da classe **Apartment**, e no console da linha 16 iremos executar o método que faz parte da classe **Property**, já que está sendo herdado. Simples!

Confira então o script completo:

```
1  class Property {
2      constructor(area, price) {
3          this.area = area
4          this.price = price
5      }
6
7      // Obter preço por metro quadrado
8      getPricePerSquareMeter() {
9          return this.price / this.area
10     }
11 }
12
13 // House herdará os atributos de Property
14 class House extends Property { }
15
16 const land = new Property(200, 50000)
17 const someHouse = new House(120, 200000)
18
19 console.log(land)
20 console.log(someHouse)
21
22 // Adicionando mais coisas em classes herdadas
23 class Apartment extends Property {
24     constructor(number, area, price) {
25         super(area, price) // serve para não repetir o this.area, this.price
26         this.number = number
27     }
28
29     getFloor() {
30         return this.number.slice(0, -2)
31     }
32 }
33
34 const apt = new Apartment('201', 100, 160000)
35 console.log(apt)
36 console.log(apt.getFloor())
37 console.log(apt.getPricePerSquareMeter())
```

Depois disso, vamos ao exercício!

Data de entrega: até dia 29 de agosto, às 23h59.

PROJETO 3 – CLASSES PARA O DOM

Crie um conjunto de classes em JS para auxiliar na interação com o DOM.

Você deverá criar, no mínimo, 4 classes diferentes:

1. Uma classe para um elemento genérico do DOM (dica: utilize o nome Component, pois o nome Element poderá gerar conflitos com a implementação do navegador). Essa classe deverá possuir uma propriedade privada para armazenar a referência ao elemento do DOM e um método de acesso para ler o valor dessa propriedade. Ela também deve possuir um método build para criar o elemento que deve ser chamado uma vez no construtor, mas também deve ser possível chamá-lo novamente através da instância. Ela também deve possuir um método render para adicionar o elemento na página que poderá ser chamado pela instância a qualquer momento.
2. Essa classe deverá possuir uma propriedade privada para armazenar a referência ao elemento do DOM e um método de acesso para ler o valor dessa propriedade.
3. Ela também deve possuir um método build para criar o elemento que deve ser chamado uma vez no construtor, mas também deve ser possível chamá-lo novamente através da instância.
4. Ela também deve possuir um método render para adicionar o elemento na página que poderá ser chamado pela instância a qualquer momento.
5. Uma classe específica para elementos input, que deve ser uma subclasse da classe Component.
6. Uma classe específica para elementos label, que deve ser uma subclasse da classe Component e no seu construtor deve ser possível indicar como primeiro parâmetro qual será o seu conteúdo em texto.
7. Uma classe específica para elementos form, que deve ser uma subclasse da classe Component e possuir um método para adicionar elementos como filhos (através das instâncias das classes Component e suas subclasses).

Vamos lá e boas práticas!