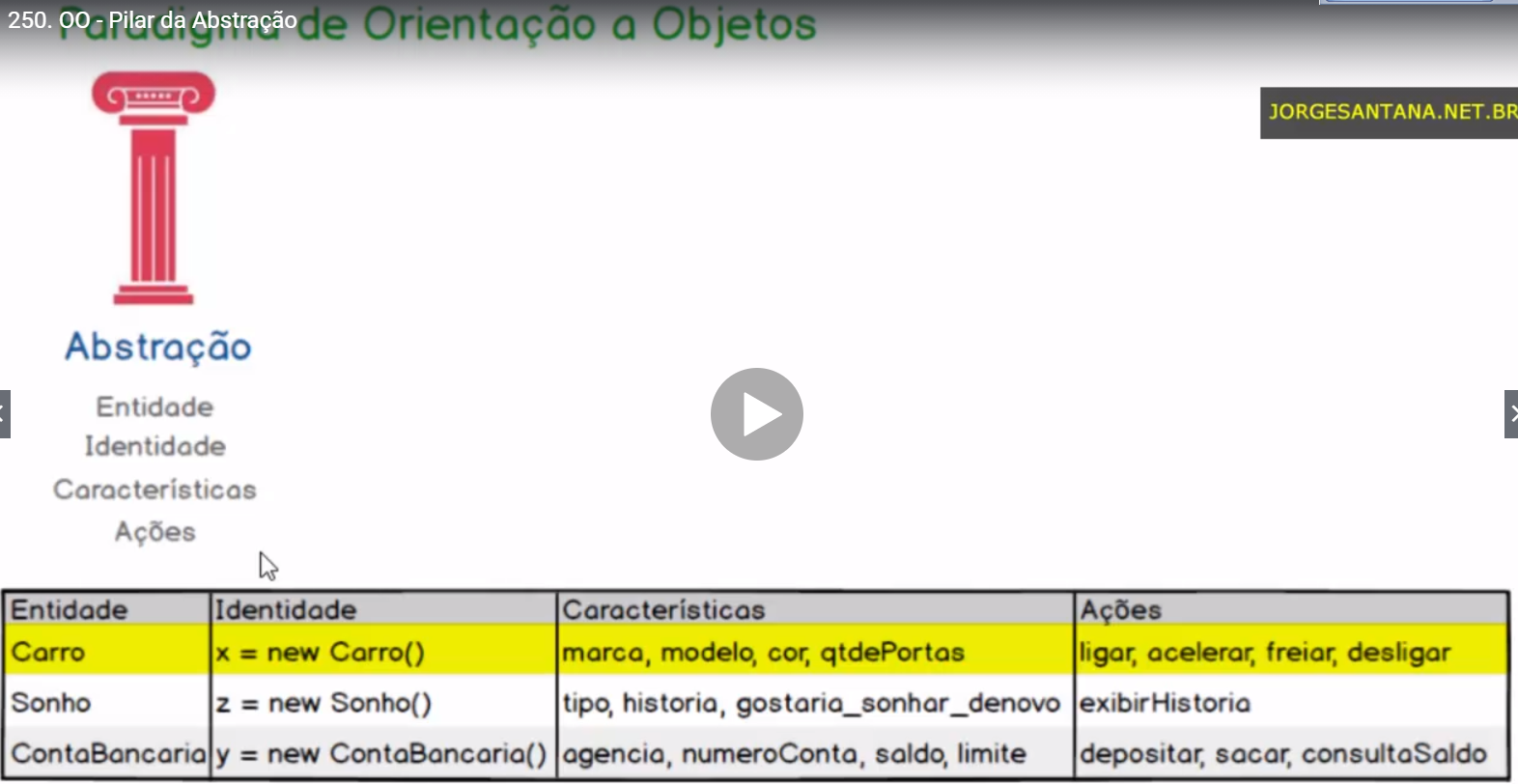
RESUMO ORIENTAÇÃO A OBJETOS



Class = representação da entidade ou objeto

Objeto = instancia do modelo;

Class = começar com letras maiculas

Dicas: A classe não é um objeto em si, e sim apenas um modelo e serve para geração de vários objetos se necessário instanciando.

Abstração x Encapsulamento tem seus dois lados: O lado conceitual e o lado técnico.

Encapsulamento significa enpasular o objeto, de tal modo tonar esse objeto seguro. Vai dizer o que dele está disponível para o sistema.

.Os **objetos** são constituídos de atributos e métodos

. O **encapsulamento** portanto é uma forma de da visibilidade para esses atributos e métodos de acordo com as nossas necessidades: E podemos falar que o determinado atributo é privado ou público assim como um método ele pode ser privado ou público.

Do ponto de vista de reutilização o enpsulamento permite a partir da visibilidade que métodos e atributos sejam herdados por outros objetos. Esse comportamento é conhecido como herança e herança por sua vez é o terceiro principio do paradigma.

Pode ser um dos pilares que merece uma atenção especial.

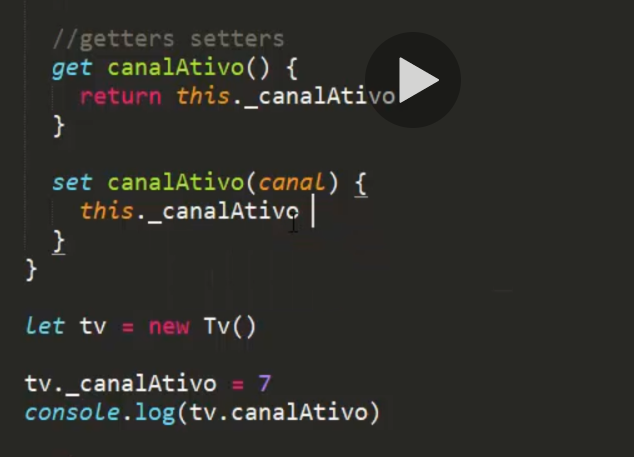
Métodos get e setters

Esses métodos são muito utilizados dentro de classes para lhes dar a possibilidade de recuperar a citar atributos privados. Embora esse tipo de conduta já seja bastante questionado os métodos get e setters que são muito comuns. E qual que é a proposta desses métodos simples atráves de métodos.

Exemplo: Vamos supor que eu quero recuperar um atributo privado.

**Get** canalAtivo( ) // eles funciona como uma função, ai sim essa função é pública. E a partir dessa função retorna o canalAtivo.

**Setters:** O set ele serve para acertar o valor de um atributo embora o javascript eu tenha a liberdade de recuperar a referencia para o objeto e acessar o atributo diretamente e a ele atribuir um valor



No set precisamos passar um parâmetro dentro da função.

indexOf : Ele procura dentro do array se existe uma referencia, e ele o retorna o índice do array aonde ele for encontrado ou se não for encontrado.

Introdução aos getters e setters do TypeScript

A seguir é apresentada uma simples Personclasse com três propriedades: age, firstName, e lastName:

class Person {

public age: number;

public firstName: string;

public lastName: string;

}

Linguagem de código: TypeScript ( typescript )

Para acessar qualquer propriedade da Personclasse, você pode simplesmente fazer assim:

let person = new Person();

person.age = 26;

Linguagem de código: TypeScript ( typescript )

Suponha que você atribua um valor proveniente da entrada do usuário à agepropriedade:

person.age = inputAge;

Linguagem de código: TypeScript ( typescript )

O inputAgepode ser qualquer número. Para garantir a validade da idade, você pode realizar uma verificação antes da atribuição da seguinte forma:

if( inputAge > 0 && inputAge < 200 ) {

person.age = inputAge;

}

Linguagem de código: TypeScript ( typescript )

Usar essa verificação em todos os lugares é redundante e tedioso.

Para evitar a repetição da verificação, você pode usar setters e getters. Os getters e setters permitem controlar o acesso às propriedades de uma classe.

Para cada propriedade:

* Um método getter retorna o valor do valor da propriedade. Um getter também é chamado de acessador.
* Um método setter atualiza o valor da propriedade. Um setter também é conhecido como um modificador.

Um método getter começa com a palavra-chave gete um método setter começa com a palavra-chave set.

class Person {

private \_age: number;

private \_firstName: string;

private \_lastName: string;

public get age() {

return this.\_age;

}

public set age(theAge: number) {

if (theAge <= 0 || theAge >= 200) {

throw new Error('The age is invalid');

}

this.\_age = theAge;

}

public getFullName(): string {

return `${this.\_firstName} ${this.\_lastName}`;

}

}

Linguagem de código: TypeScript ( typescript )

Como funciona.

* Em primeiro lugar, mudar os modificadores de acesso dos age, firstNamee lastNamepropriedades de publica private.
* Em segundo lugar, altere a propriedade agepara \_age.
* Terceiro, crie métodos getter e setter para a \_agepropriedade. No método setter, verifique a validade da idade de entrada antes de atribuí-la à \_agepropriedade.

Agora, você pode acessar o agemétodo setter da seguinte maneira:

let person = new Person();

person.age = 10;

Linguagem de código: TypeScript ( typescript )

Observe que a chamada para o setter não tem parênteses como um método regular. Quando você chama person.age, o agemétodo setter é invocado. Se você atribuir um agevalor inválido , o configurador lançará um erro:

person.age = 0;

Linguagem de código: TypeScript ( typescript )

Erro:

Error: The age is invalid

Linguagem de código: TypeScript ( typescript )

Quando você acessa o person.age, o agegetter é invocado.

console.log(person.age);

Linguagem de código: TypeScript ( typescript )

O seguinte adiciona os getters e setters às propriedades firstNamee lastName.

class Person {

private \_age: number;

private \_firstName: string;

private \_lastName: string;

public get age() {

return this.\_age;

}

public set age(theAge: number) {

if (theAge <= 0 || theAge >= 200) {

throw new Error('The age is invalid');

}

this.\_age = theAge;

}

public get firstName() {

return this.\_firstName;

}

public set firstName(theFirstName: string) {

if (!theFirstName) {

throw new Error('Invalid first name.');

}

this.\_firstName = theFirstName;

}

public get lastName() {

return this.\_lastName;

}

public set lastName(theLastName: string) {

if (!theLastName) {

throw new Error('Invalid last name.');

}

this.\_lastName = theLastName;

}

public getFullName(): string {

return `${this.firstName} ${this.lastName}`;

}

}

Linguagem de código: TypeScript ( typescript )

Mais exemplos de getters / setters de TypeScript

Como você pode ver no código, os setters são úteis quando você deseja validar os dados antes de atribuí-los às propriedades. Além disso, você pode executar lógicas complexas.

O seguinte mostra como criar o fullnamegetter e o setter.

class Person {

*// ... other code*

public get fullName() {

return `${this.firstName} ${this.lastName}`;

}

public set fullName(name: string) {

let parts = name.split(' ');

if (parts.length != 2) {

throw new Error('Invalid name format: first last');

}

this.firstName = parts[0];

this.lastName = parts[1];

}

}

Linguagem de código: TypeScript ( typescript )

Como funciona.

* O método getter retorna a concatenação do nome e do sobrenome.
* O método setter aceita uma string como o nome completo com o formato: first laste atribui a primeira parte à propriedade do primeiro nome e a segunda parte à propriedade do sobrenome.

Agora, você pode acessar o fullnamesetter e getter como uma propriedade de classe regular:

let person = new Person();

person.fullname = 'John Doe';

console.log(person.fullName);

Linguagem de código: TypeScript ( typescript )

Resumo

* Use getters / setters do TypeScript para controlar as propriedades de acesso de uma classe.
* Os getter / setters também são conhecidos como acessadores / mutadores.

**Método super ( )**

O operador super fornece acesso para o método do construtor da classe pai. Então o operador super é sempre incluído dentro do construtor de classe filhas

Polimorfismo

Polimorfismo nada mais é que a sobrescrito de métodos e isso acontece quando trabalhamos com a herança ou seja temos um objeto pai e um objeto filho sendo que o objeto filho herda os atributos e métodos no objeto pai, mas não necessariamente o objeto filho tem que se comportar da mesma forma que o objeto pai.