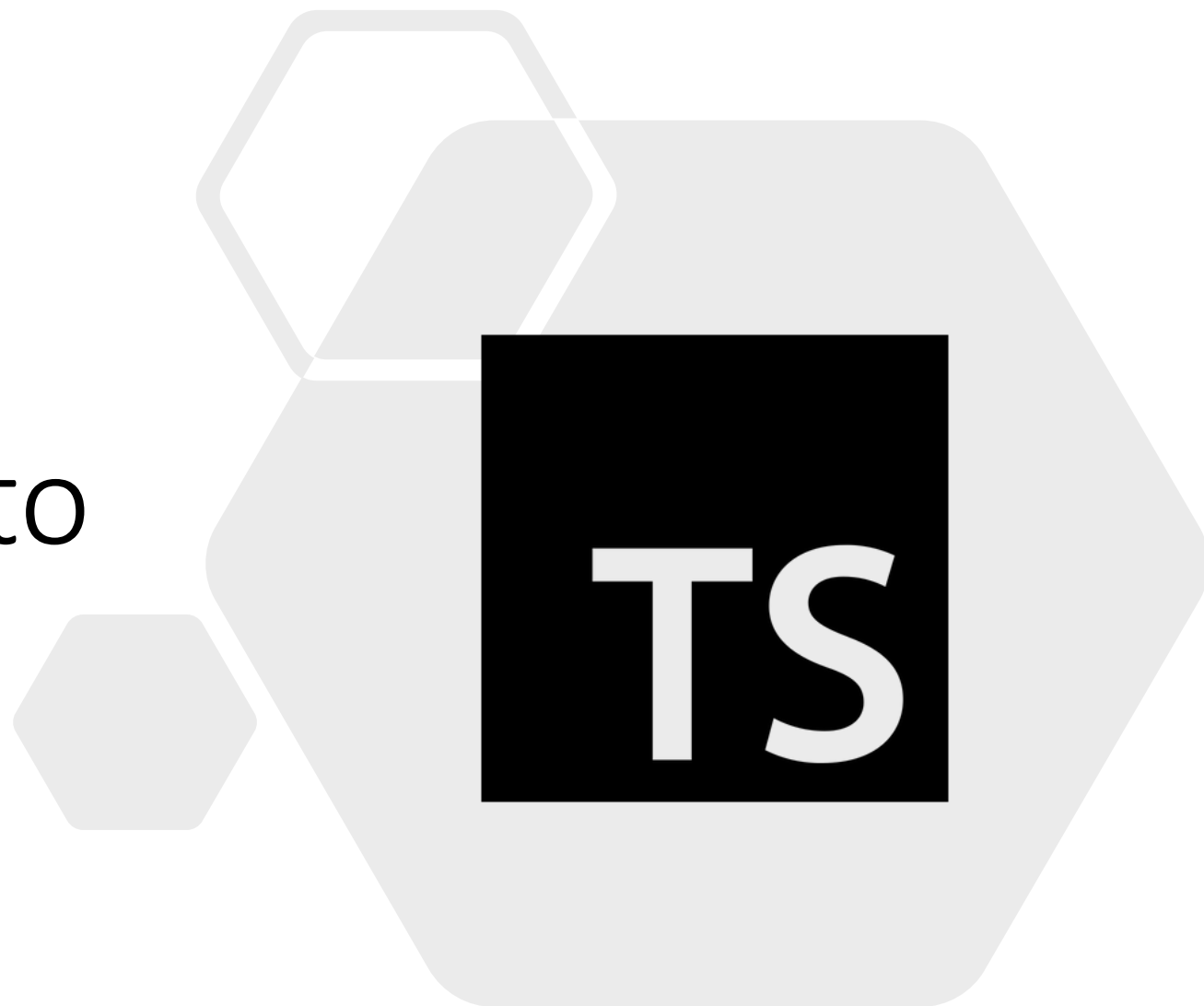
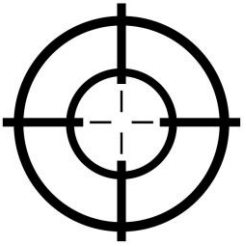


Encapsulamento



Encapsulamento

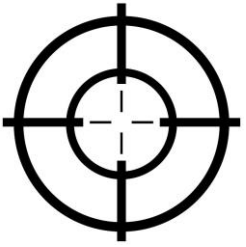


Encapsulamento se refere ao agrupamento de dados, junto com os métodos que operam esses dados. Muitas linguagens de programação usam encapsulamento frequentemente na forma de classes.

Encapsulamento refere-se à restrição do acesso direto a alguns dos componentes de um objeto.

Encapsular dados, portanto, significa qualificar a forma de armazenar dados em objetos.

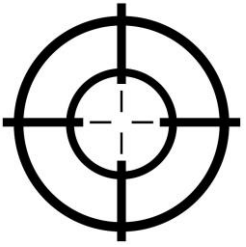
Qual a vantagem do encapsulamento?



O encapsulamento é um mecanismo de restrição do acesso direto a alguns componentes de um objeto, de forma que outros objetos não possam acessar os valores de todas as variáveis ou métodos de um objeto específico.

O encapsulamento pode ser usado para ocultar ou proteger membros e métodos associados a uma classe ou objeto instanciado, do acesso direto de códigos externos.

Como proteger dados?



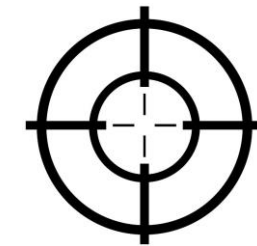
```
export default class Empresa {  
  public razaoSocial: string  
  public nomeFantasia: string  
  public cnpj: string  
  public endereco: Endereco  
  public telefones: Telefone[]  
  public funcionarios: Funcionario[]  
}
```

```
export default class Empresa {  
  private razaoSocial: string  
  public nomeFantasia: string  
  private cnpj: string  
  public endereco: Endereco  
  public telefones: Telefone[]  
  private funcionarios: Funcionario[]  
}
```

Para encapsular dados utiliza-se modificadores de acesso, palavras reservadas que modificam a forma de acesso aos dados.




Desta forma, a linguagem TypeScript pode controlar se determinados métodos ou propriedades são “visíveis” aos códigos fora da classe.

Como o acesso é bloqueado?



```
let empresa = new Empresa(funcionarios, endereco, 'ABC LTDA', 'Mercado online', '999-999-999-99', telefones)
```

empresa.

	endereco	(property) Empresa.endereco: Endereco
	nomeFantasia	
	telefones	

```
export default class Empresa {  
  private razaoSocial: string  
  public nomeFantasia: string  
  private cnpj: string  
  public endereco: Endereco  
  public telefones: Telefone[]  
  private funcionarios: Funcionario[]  
}
```

Os atributos privados deixam de ser visíveis para fora do objeto e portanto, não podem ser modificados diretamente.

Os demais atributos públicos, que continuam visíveis, são modificáveis diretamente.

Quais e quantos são os modificadores de acesso?

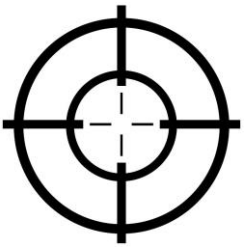
Para a linguagem TypeScript existem três modificadores de acesso, eles são: `private`, `protected` e `public`.

A visibilidade padrão dos membros de uma classe é pública, ou seja, modificada pela palavra-chave `public`. Um membro público pode ser acessado em qualquer lugar.

```
export default class Empresa {  
  private razaoSocial: string  
  { nomeFantasia: string  
    private cnpj: string  
    endereco: Endereco  
    public telefones: Telefone[] }  
  private funcionarios: Funcionario[]  
}
```

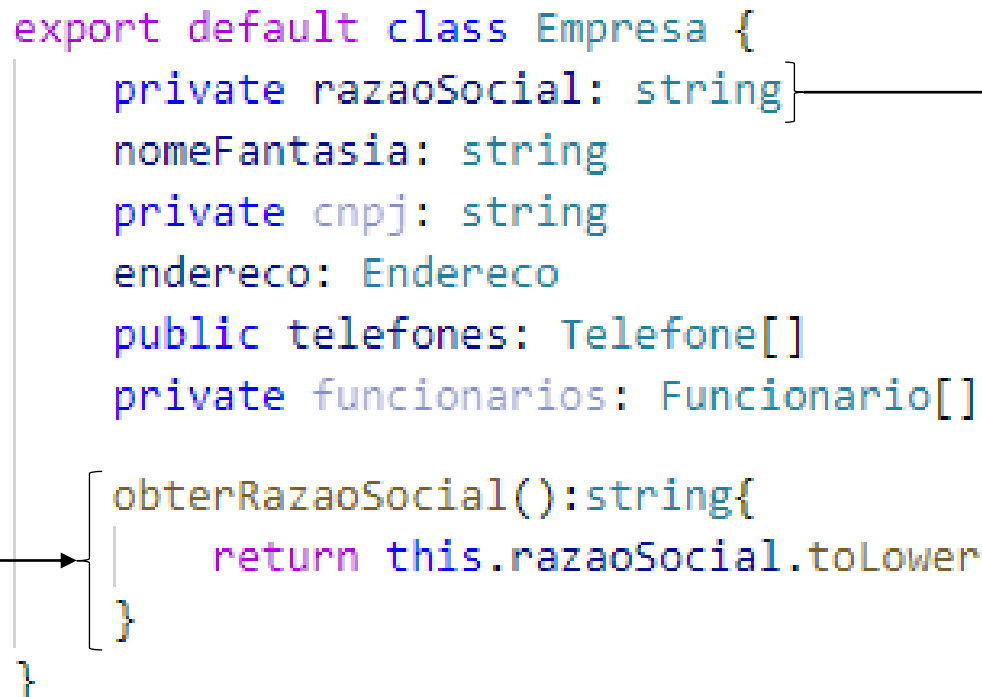
Quando não se declara o modificador o padrão aplicado é `public`.

Encapsular significa nunca mais acessar?



Não! Encapsular significa acessar com restrição. A forma de fazer isto é através de métodos.

```
export default class Empresa {  
    private razaoSocial: string  
    nomeFantasia: string  
    private cnpj: string  
    endereco: Endereco  
    public telefones: Telefone[]  
    private funcionarios: Funcionario[]  
  
    obterRazaoSocial(): string {  
        return this.razaoSocial.toLowerCase()  
    }  
}
```

A diagram with two arrows. One arrow starts from the text 'isto é através de métodos.' and points to the 'obterRazaoSocial()' method call in the code. The other arrow starts from the 'private razaoSocial: string' property and points to the 'this.razaoSocial' access inside the method's return statement.

Lembre-se que o objetivo do encapsulamento é modificar a forma de acesso aos dados.

O modificador private bloqueia, completamente, a visibilidade de um membro da classe a qualquer código fora do próprio corpo da classe.

Métodos get e set



```
export default class Empresa {  
  private razaoSocial: string  
  nomeFantasia: string  
  private cnpj: string  
  endereco: Endereco  
  public telefones: Telefone[]  
  private funcionarios: Funcionario[]  
  
  getRazaoSocial():string{  
    return this.razaoSocial.toLowerCase()  
  }  
  
  setRazaoSocial(razaoSocial:string){  
    this.razaoSocial = razaoSocial  
  }  
}
```

Existem várias convenções entre os desenvolvedores que utilizam o paradigma de programação orientada à objetos. Uma delas é o uso de métodos get e set. A convenção está na nomenclatura dos métodos.

Métodos get/set são usados para acessar propriedades privadas e também outros valores que devam ser calculados.

Palavras reservadas get e set



```
export default class Empresa {  
  private razaoSocial: string  
  nomeFantasia: string  
  private cnpj: string  
  endereco: Endereco  
  public telefones: Telefone[]  
  private funcionarios: funcionario[]  
}
```

```
export default class Empresa {  
  private razaoSocial: string  
  nomeFantasia: string  
  private cnpj: string  
  endereco: Endereco  
  public telefones: Telefone[]  
  private funcionarios: Funcionario[]  
}
```

```
  public get pegarRazaoSocial(): string {  
    return this.razaoSocial  
  }
```

```
  getRazaoSocial():string{  
    return this.razaoSocial.toLowerCase()  
  }
```

```
  public set alterarRazaoSocial(razaoSocial: string) {  
    this.razaoSocial = razaoSocial  
  }
```

```
  setRazaoSocial(razaoSocial:string){  
    this.razaoSocial = razaoSocial  
  }
```

Este tipo de implementação permite o acesso aos métodos como se fossem propriedades.

O que são campos readonly?

Campos readonly (somente leitura) são propriedades que podem ser acessadas, ou seja, ter seu valor lido, mas não podem ser modificadas, ou seja, ter seu valor alterado por códigos externos à classe.

A atribuição de valor para um campo do tipo readonly pode ser feita no momento de sua declaração ou no método construtor.

Comparando códigos...



```
export default class Empresa {  
  private razaoSocial: string  
  nomeFantasia: string  
  private cnpj: string  
  endereco: Endereco  
  public telefones: Telefone[]  
  private funcionarios: funcionario[]
```

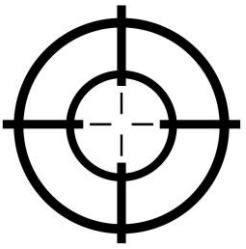
```
  public get pegarRazaoSocial(): string {  
    return this.razaoSocial  
  }
```

```
  public set alterarRazaoSocial(razaoSocial: string) {  
    this.razaoSocial = razaoSocial  
  }  
}
```

```
export default class Empresa {  
  readonly razaoSocial: string  
  nomeFantasia: string  
  private cnpj: string  
  endereco: Endereco  
  public telefones: Telefone[]  
  private funcionarios: Funcionario[]  
}
```

Usar o modificador de acesso readonly ajuda a diminuir o código da classe, pois não é preciso privar o campo para sua proteção e, também, declarar um método para obter seu valor.

Uma outra vantagem do encapsulamento...



O encapsulamento pode melhorar o tratamento e manutenção da qualidade dos dados.

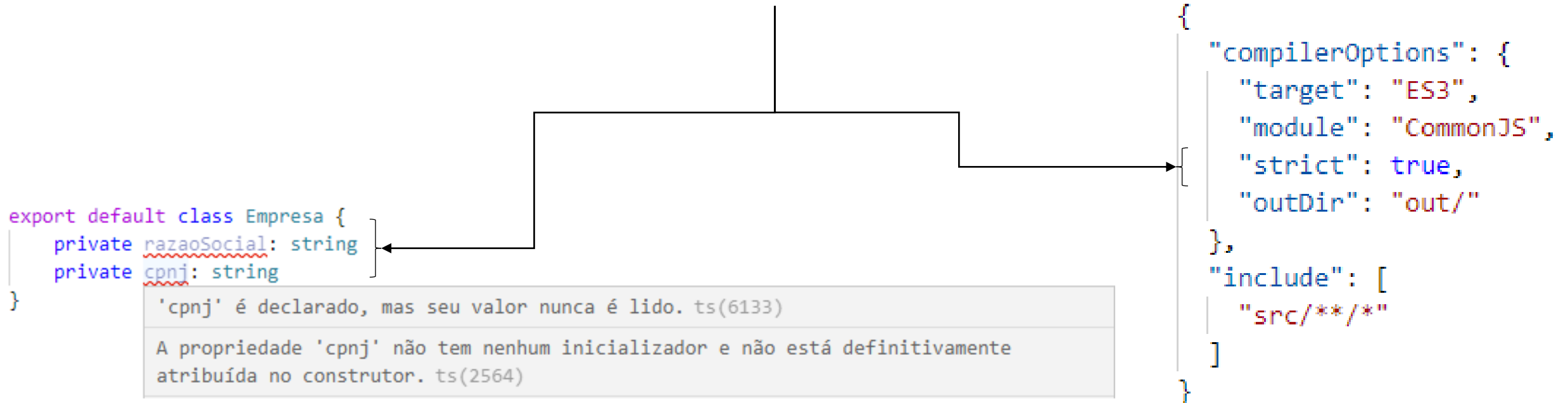
```
getRazaoSocial():string{  
    return this.razaoSocial.toUpperCase()  
}  
  
setRazaoSocial(razaoSocial:string){  
    this.razaoSocial = razaoSocial.toUpperCase();  
}
```

Nestes métodos, antes do atributo ser preenchido ou lido ocorre um “tratamento”.

Verificação strict do compilador...



O transcompilador TypeScript possui várias configurações, dentre elas existe a opção de verificação strict. Quando esta opção está habilitada, o transcompilador não permite que uma classe tenha campos declarados sem a certeza de sua inicialização na construção do objeto.



Asserção de atribuição definitiva



A asserção de atribuição definitiva (definite assignment assertion) é um recurso que permite forçar a declaração de um campo, em uma classe, mesmo que ele não tenha inicialização durante a construção do objeto.

Para isso, adiciona-se um sinal de !, na declaração do campo.

```
export default class Empresa {  
    private razaoSocial!: string  
    private cnpj!: string  
}
```

