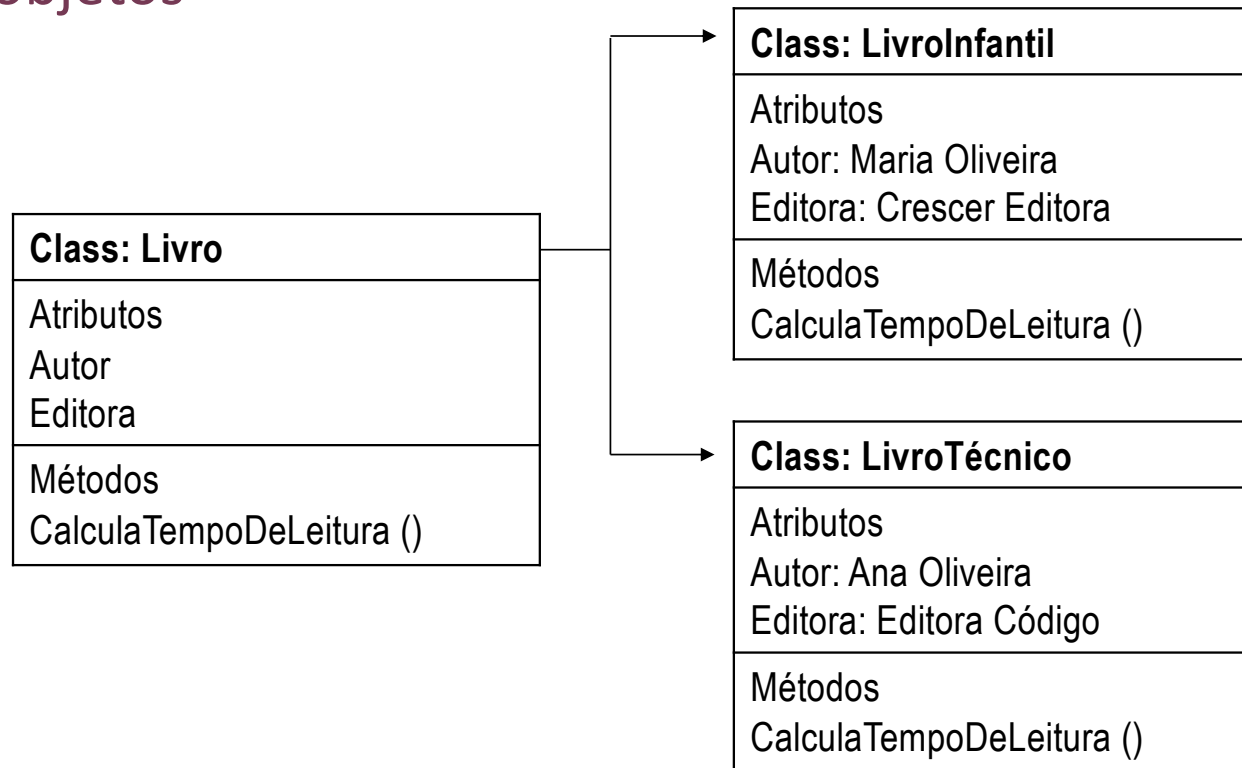


Classes

- A) Classes e objetos
- B) Criação do projeto
- C) Propriedades e métodos
- D) Instância de classe
- E) Lista de objetos
- F) Herança/Encapsulamento
- G) Polimorfismo
- H) Sobrecarga
- I) Sobrescrita

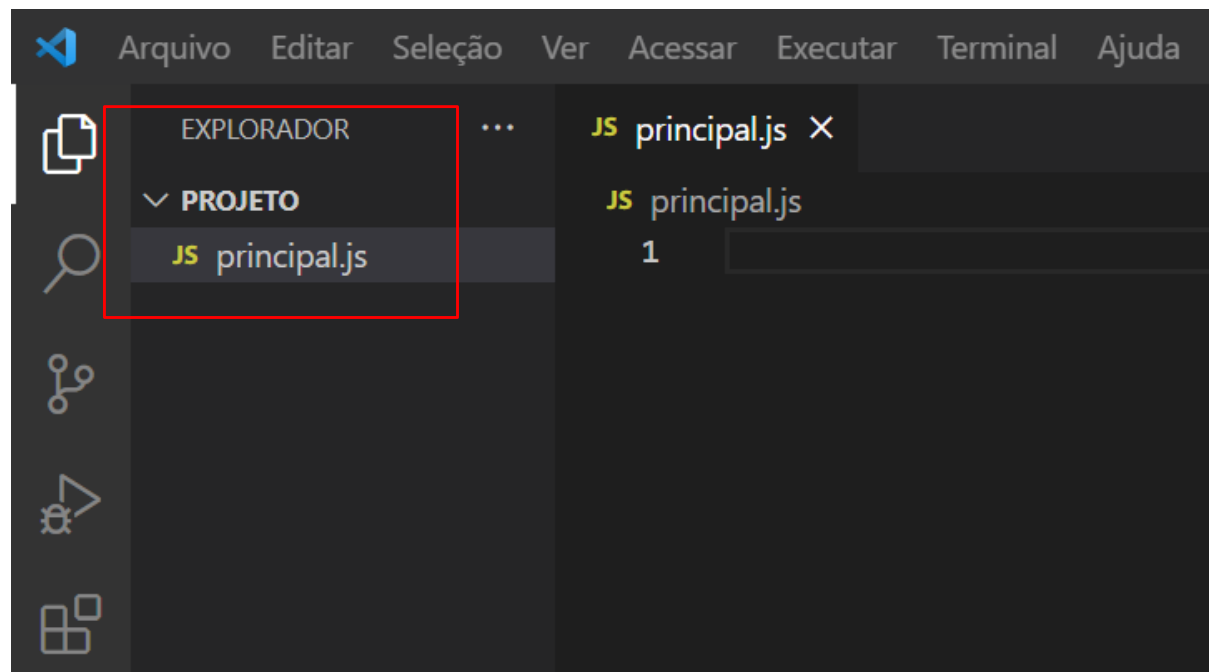
a) Classes e objetos

- Exemplos



b) Criação do projeto

- Classe e arquivo → mesmo nome



c) Propriedades e métodos

- Exemplo de classe, método construtor, atributos e um método implementado.

```
class Livro{  
    constructor (pNome, pPreco){  
        this.Nome = pNome; //atributos são declarados no construtor  
        this.Preco = pPreco; //atributos são declarados no construtor  
    }  
  
    calcularDesconto(){  
  
    }  
}
```

d) Instância de classe

- Instância = objeto de uma classe

```
class Livro{  
  constructor (pNome, pPreco){  
    this.Nome = pNome;  
    this.Preco = pPreco;  
  }  
  
  get Nome(){ return this.Nome;}  
  set Nome(pNome){ this.nome = pNome;}  
  get Preco(){ return this.Preco;}  
  set Preco(pPreco){ this.preco = pPreco;}  
}
```

```
var obj_livro = new Livro('Livro de POO', 100);//criando uma  
instancia da classe Livro  
console.log(obj_livro.nome)  
console.log(obj_livro.preco)
```

e) Lista de objetos

- Exemplo de criação de lista de objetos.

```
var lista = []  
var obj_livro1 = new Livro("POO com JavaScript",200);  
lista.push(obj_livro1);  
var obj_livro2 = new Livro("POO com C#",180);  
lista.push(obj_livro2);  
  
console.log(lista[0]);  
console.log(lista[1]);
```

f) Herança/Encapsulamento

- Exemplo

```
class Conta{  
    constructor(){  
        this.Saldo = 0;  
    }  
    get Saldo(){return this.saldo;}  
    set Saldo(pSaldo){this.saldo = pSaldo;}  
}
```

```
class Corrente extends Conta {  
    constructor(pLimite){  
        super();  
        this.Limite = pLimite;  
    }  
    get Limite(){return this.limite;}  
    set Limite(pLimite){this.limite = pLimite;}  
}
```

f) Herança/Encapsulamento

```
var objeto_corrente = new Corrente(300);  
objeto_corrente.saldo = 1000;  
console.log(objeto_corrente);
```


g) Polimorfismo

- Exemplo

```
class Conta{
    constructor(){
        this.Saldo = 0;
    }
    get Saldo(){return this.saldo;}
    set Saldo(pSaldo){this.saldo = pSaldo;}

    imprimir(){
        return "Saldo: " + this.saldo;
    }
}
```

```
class Corrente extends Conta {
    constructor(pLimite){
        super();
        this.Limite = pLimite;
    }
    get Limite(){return this.limite;}
    set Limite(pLimite){this.limite = pLimite;}

    imprimir(){
        return super.imprimir() + "\nLimite: " + this.limite;
    }
}
```

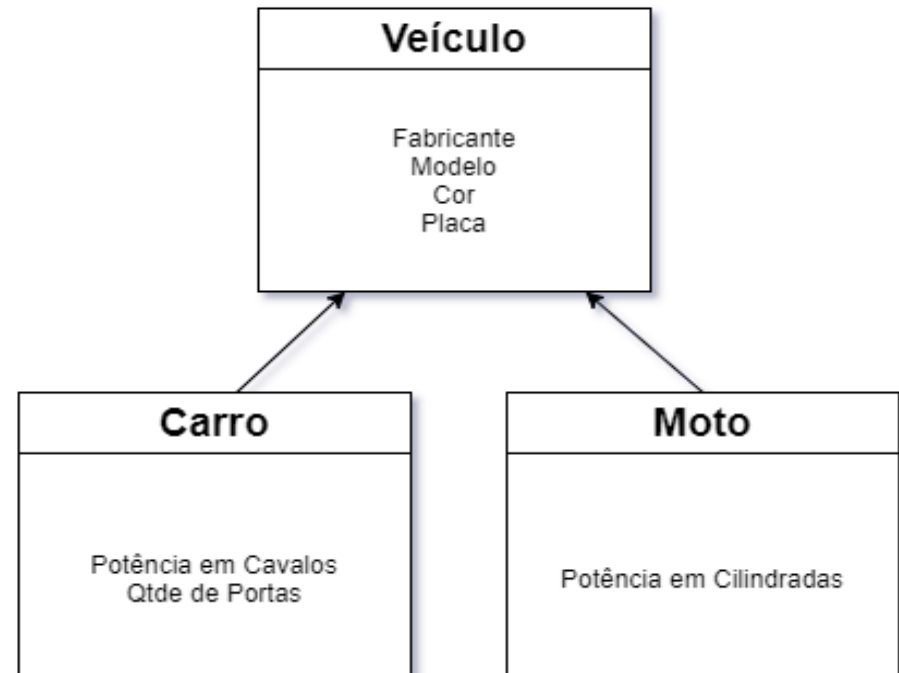
g) Polimorfismo

- Exemplo

```
var objeto_corrente = new Corrente(300);  
objeto_corrente.saldo = 1000;  
console.log(objeto_corrente.imprimir());
```

Herança

Quando trabalhamos com o paradigma orientado a objetos e começamos a criar nossas primeiras classes, logo entendemos que há uma grande necessidade de compartilhamento de atributos e métodos entre as classes.



h) Sobrecarga

```
class Operacoes{  
    constructor(){  
  
    }  
    Somar(valorA, valorB){  
        console.log(valorA + valorB)  
    }  
  
    Somar(valorA, valorB, valorC){  
        console.log(valorA + valorB + valorC)  
    }  
}
```

h) Sobrecarga

```
var objeto = new Operacoes();  
objeto.Somar(10,15)
```

```
var objeto = new Operacoes();  
objeto.Somar(10,15,30)
```

i) Sobrescrita

```
class Funcionario{
    constructor (pNome, pSalario){
        this.Nome = pNome;
        this.Salario = pSalario;
    }
    get Nome(){ return this.Nome;}
    set Nome(pNome){ this.nome = pNome;}
    get Salario(){ return this.Salario;}
    set Salario(pSalario){ this.salario = pSalario;}

    calcularSalario(){

    }
}
```

i) Sobrescrita

```
class Vendedor extends Funcionario{
    constructor (pNome, pSalario, pComissao){
        super(pNome, pSalario)
        this.Comissao = pComissao;
    }
    get Comissao(){ return this.Comissao;}
    set Comissao(pComissao){ this.comissao =
pComissao;}

@Override
    calcularSalario(){
        return this.salario + this.comissao;
    }
}
```

```
class OperadoraCaixa extends Funcionario{
    constructor (pNome, pSalario, pBonus){
        super(pNome, pSalario)
        this.Bonus = pBonus;
    }
    get Bonus(){ return this.Bonus;}
    set Bonus(pBonus){ this.bonus = pBonus;}

@Override
    calcularSalario(){
        return this.salario + this.bonus;
    }
}
```

i) Sobrescrita

```
var vendedor = new Vendedor("Luiz",1100, 150);  
console.log(vendedor.calcularSalario());  
  
var opcaixa = new OperadoraCaixa("José",1100, 100);  
console.log(opcaixa.calcularSalario());
```