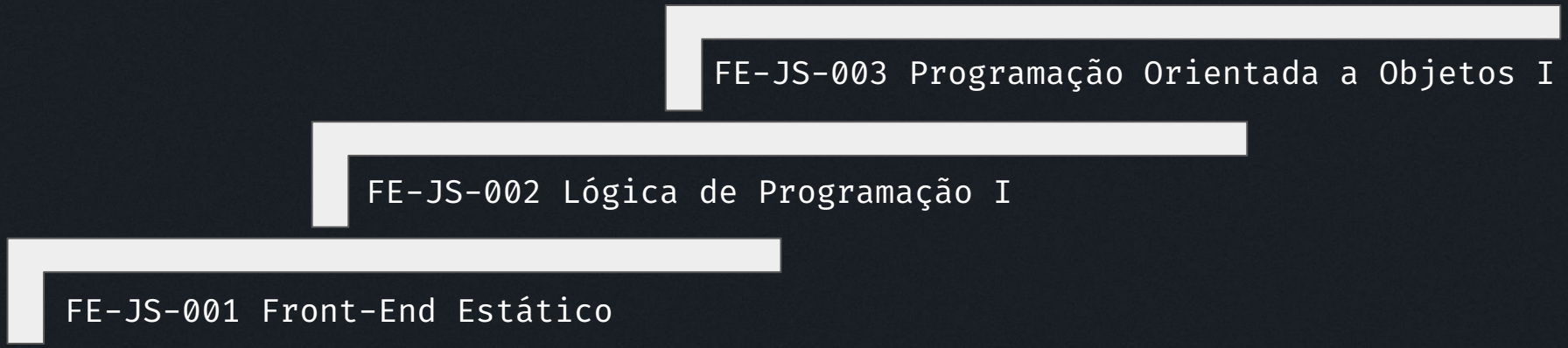
Aula 01 - Introdução à POO

Módulo Programação Orientada a Objetos I - Ada

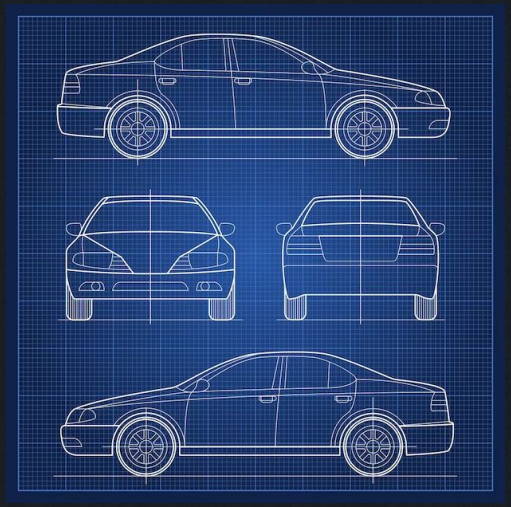
Recapitulando

Por onde já passamos?



Classes

Carro

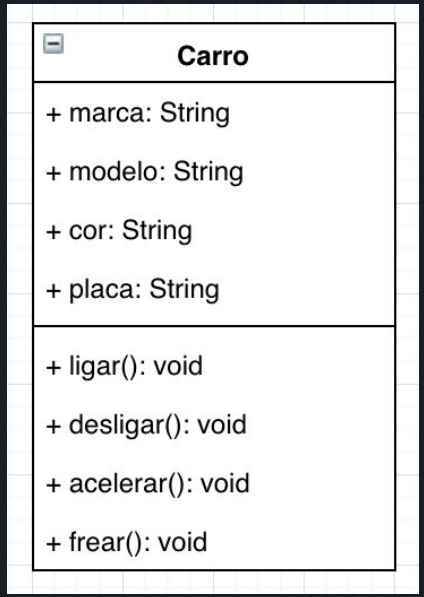


Nome →

Atributos →

Métodos →

UML Carro



Atributos, métodos e construtor

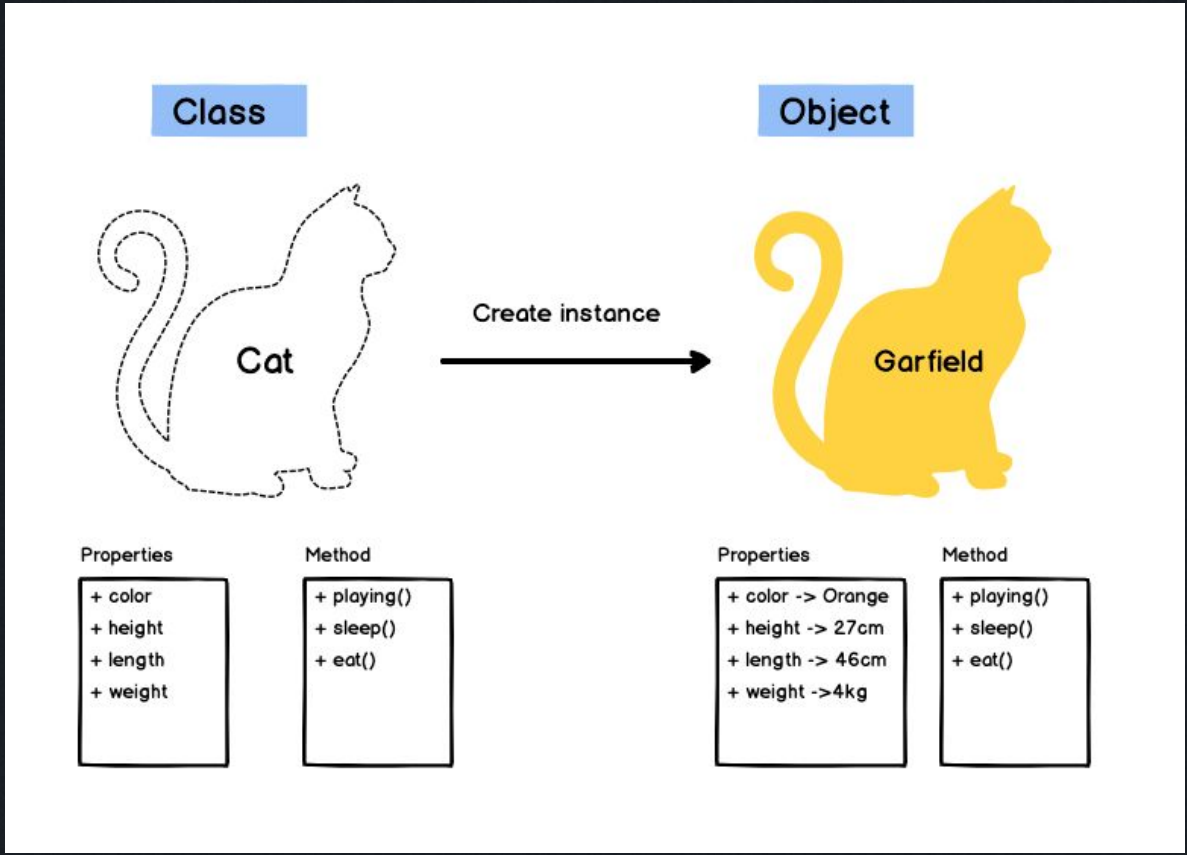
Sem classes

```
const agenda = {
  contatos: [
    {
        nome: 'contato1',
        telefone: 'telefone1',
        email: 'email1@teste.com'
    },
    {
      nome: 'contato2',
      telefone: 'telefone2',
      email: 'email2@teste.com'
    },
    {
      nome: 'contato3',
      telefone: 'telefone3',
      email: 'email3@teste.com'
    },
    {
      nome: 'contato4',
      telefone: 'telefone4',
      email: 'email4@teste.com'
    }
  ],
  adicionar: function(contato){
      this.contatos.push(contato)
  }
}
```

Com classes

```
class Agenda {
  #contatos = [];
  constructor(){}
  adicionarContato(contato){
    if (contato.nome.trim() === "")
        return;
    this.contatos = contato;
  }
  get contatos(){
    return this.#contatos;
  }
  set contatos(contato) {
    this.#contatos.push(contato);
  }
}
```

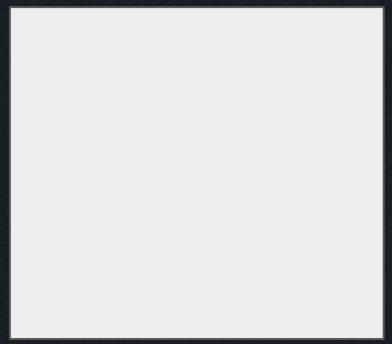
Classe X Objeto



Enquanto a classe é a base (Projeto) a instância é o próprio objeto com os estados e comportamentos

Atividade

Criar classes para figuras geométricas



Aula 02 - Static e encapsulamento

Módulo Programação Orientada a Objetos I - Ada

Recapitulando

Classes

Objetos

Atributos

Métodos

Construtor

Atributos e métodos estáticos

Para a criação de atributos ou métodos estáticos em JavaScript é utilizada a palavra reservada **static**, precedendo o nome do atributo ou método.

```
class Pessoa {
  static totalCadastrado = 0;
  constructor(nome) {
    this.nome = nome;
    ++Pessoa.totalCadastrado;
  }
}
```

#, private e accessors

Encapsulamento é o conceito de restringir o acesso aos atributos e métodos de uma classe diretamente, seja para leitura ou escrita.

A maioria das linguagens orientadas a objetos utilizam algum comando como **private**, por exemplo, para restringir esse acesso. No JavaScript, usamos o identificador **#** para este fim.

```
class Quadrado {
  #cor;
  constructor(base, altura) {
    this.#cor = "blue";
    this.base = base;
    this.altura = altura;
    this.area = this.#obterArea();
  }

  #obterArea() {
    return this.base * this.altura;
  }
}
```

Aula 03 - Herança e polimorfismo

Módulo Programação Orientada a Objetos I - Ada

Herança

A sintaxe para fazermos herança em JS é a mesma utilizada em JAVA, ou seja, a palavra **extends** denota a herança e a palavra **super** refere-se a superclasse que foi herdada.

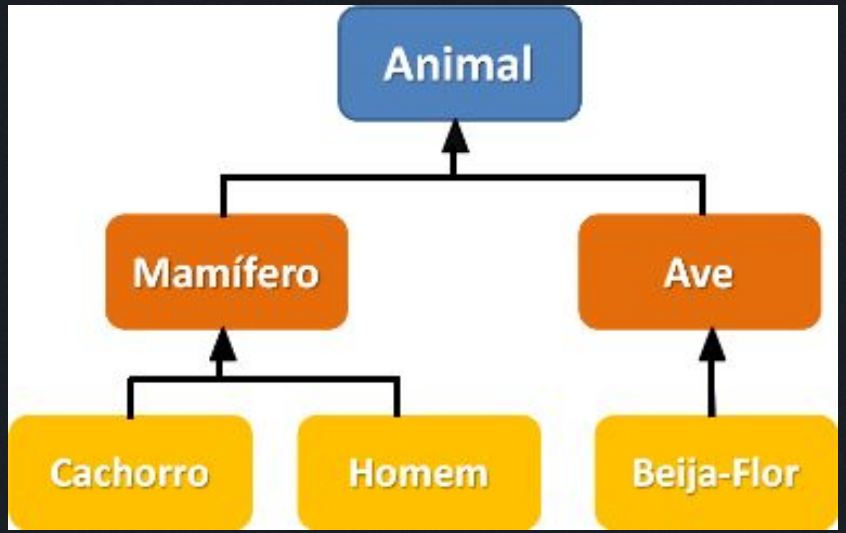
```
class Pessoa {
  constructor(nome, idade) {
    this.nome = nome;
    this.idade = idade;
  }
}

class Cidadao extends Pessoa {
  constructor(nome, idade, cpf, rg) {
    super(nome, idade);
    this.cpf = cpf;
    this.rg = rg;
  }
}
```

Herança

Quando não puder dizer que a subclasse **É** a superclasse, ela não será candidata para herança.

Quanto mais alta a classe, menos específica e mais genérica.



Polimorfismo

Com ele temos a possibilidade de sobrescrever um comportamento, gerando uma nova forma de execução do método pela subclasse.

```
class Pessoa {
  cumprimento() {
    return "Olá";
  }
}

class Professor extends Pessoa {
  cumprimento() {
    return "Olá turma!"
  }
}

class Funcionario extends Pessoa {
  cumprimento() {
    return "Bom dia, chefe!"
  }
}
```

Aula 04 - Prototype

Módulo Programação Orientada a Objetos I - Ada

Prototypes

Exemplo de prototype no tipo String

```
> String.prototype
< String {'', constructor: ƒ, anchor: ƒ, at: ƒ, big: ƒ, …} i
    anchor: ƒ anchor()
    at: ƒ at()
    big: ƒ big()
    blink: ƒ blink()
    bold: ƒ bold()
    charAt: ƒ charAt()
    charCodeAt: ƒ charCodeAt()
    codePointAt: ƒ codePointAt()
    concat: ƒ concat()
    constructor: ƒ String()
    endsWith: ƒ endsWith()
    fixed: ƒ fixed()
    fontcolor: ƒ fontcolor()
    fontsize: ƒ fontsize()
    includes: ƒ includes()
    indexOf: ƒ indexOf()
    isWellFormed: ƒ isWellFormed()
    italics: ƒ italics()
    lastIndexOf: ƒ lastIndexOf()
    length: 0
```

Exemplo de prototype **Italics** do tipo String

```
String.prototype.italics()
'<i></i>'
```

```
'bruno'.italics()
'<i>bruno</i>'
```

Exemplo de criação de função no prototype do tipo String

```
> a.__proto__.sum = function sum(param1, param2) {
      return param1 + param2;
  }
< ƒ sum(param1, param2) {
      return param1 + param2;
  }
```

```
> String.prototype.sum(2, 7)
< 9
```

Aula 05 - Projetos

Módulo Programação Orientada a Objetos I - Ada

Atividade

Projeto Individual

- Pensem em um catálogo/cardápio digital em HTML;
- Modelem as entidades necessárias imaginando possibilidade de cadastro, exibição, edição e deleção de itens (CRUD);
- Utilizar todos os conceitos vistos em sala de aula até o momento, de forma que faça sentido para o contexto do projeto;
- Criar pelo menos uma função utilizando prototype;
- Diagrama de classes UML simplificado.

- **EXTRA**
- Criar um HTML para exibir os itens;
- Formulário para cadastro dos itens.

Atividade

Projeto em Grupo

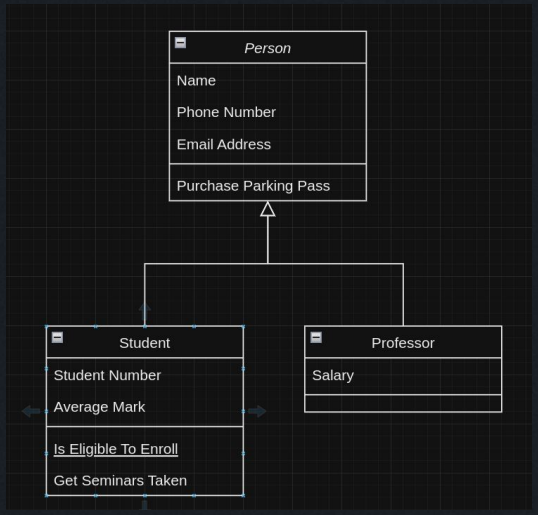
- Criar um jogo usando OO; (Forca, por exemplo)
- Pelo menos três classes para três entidades presentes no sistema do jogo (GameController, Player e Match);
- O usuário deve poder chutar a palavra de uma vez;
- O usuário deve poder jogar/tentar novamente (reiniciar);
- Deve haver uma dica ou tema visualmente indicados;
- O jogo deve possuir pontuação.

- **EXTRA**
- Buscar as palavras em uma API externa;
- Criar um teclado virtual (manipulação do DOM).

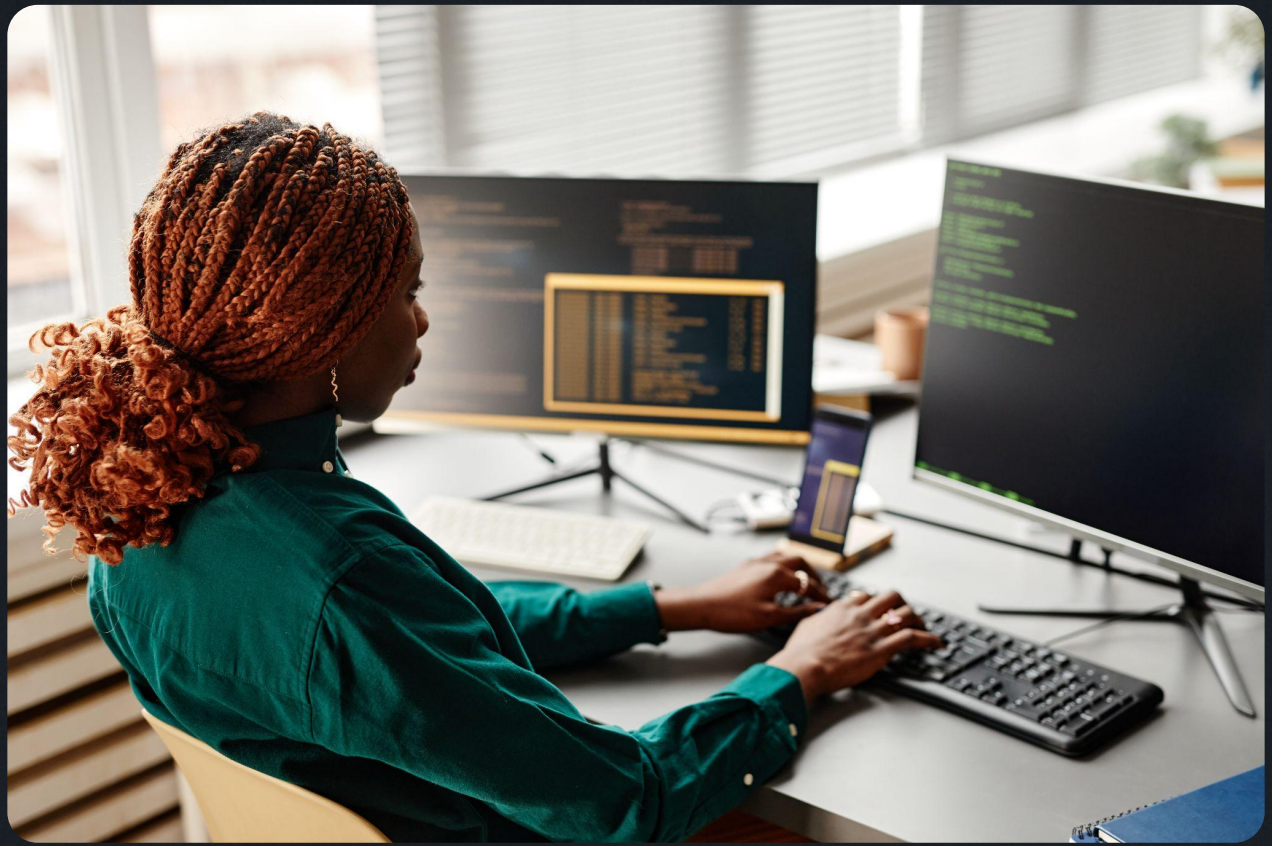
Atividade

Material de exemplo que podem inspirar
na criação dos projetos

- CRUD = https://codepen.io/brunoleomont/full/qVJQLd
- JOGO = https://github.com/eduardoworrel/universe-explorer
- Teclado = https://github.com/shhdharmen/keyboard-css
- Teclado 2 = https://developer.mozilla.org/pt-BR/docs/Web/API/Web_Audio_API/Simple_synth
- UML simplificado



Vamos praticar?!



Obrig.ada