



Quem se prepara, não para.

Arquitetura de Sistemas

7º período

Professora: Michelle Hanne

Características do Node.js

Node.js é uma plataforma construída sobre o motor **JavaScript** do Google Chrome, que proporciona construir aplicações de rede rápidas e escaláveis.

Node.js usa um modelo de I/O direcionada a **evento não bloqueante** que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados através de dispositivos distribuídos.

Que problema o Node pode resolver?

- Em linguagens como Java™ e PHP, cada conexão cria uma ***nova thread*** que potencialmente tem anexado **2 MB de memória com ela**. Em um sistema que tenha **8 GB de RAM**, isso põe o número máximo teórico de conexões concorrentes a cerca de **4.000 usuários**.
- Se você quer que sua aplicação web suporte mais usuários, você tem que adicionar mais e mais servidores.
- **Problemas:** velocidade de tráfego, velocidade do processador e capacidade da memória em lidar com o número de conexões concorrentes.

Que problema o Node pode resolver?

Node resolve esta questão trocando a maneira como a conexão é tratada no servidor.

Ao invés de criar uma nova **OS thread** a cada conexão (e alocar a memória anexa a ela), cada conexão dispara um evento executado dentro da **engine** de processos do Node.

Node afirma que nunca vai dar *deadlock*, já que não há bloqueios permitidos, e ele não bloqueia diretamente para chamadas de I/O.

Um servidor rodando Node pode suportar dezenas de milhares de conexões simultâneas.

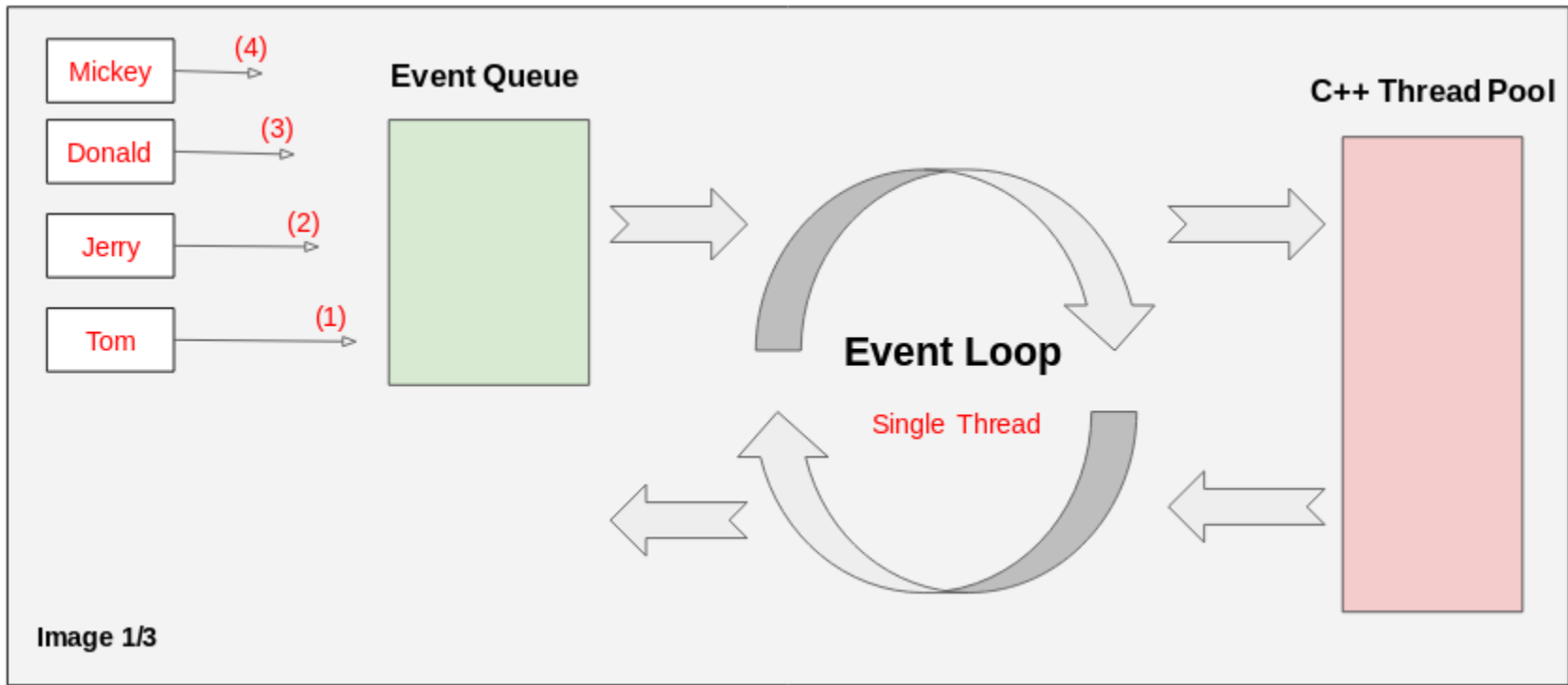
O que é o Node.JS

- Node é um servidor de programas. Entretanto o produto base do Node definitivamente não é como o *Apache* ou o *Tomcat*.
- Node tem o conceito de módulos que podem ser adicionados no núcleo do Node.
- O Node roda em uma JavaScript V8 VM (é o motor que a Google usa com seu navegador Chrome.)
- Sim, JavaScript do lado do servidor.

O que realmente acontece com o JavaScript no lado do cliente?

A *engine JavaScript* realmente interpreta o código e o executa. Com o V8 a Google criou um ultra-rápido interpretador escrito em C++, com um outro aspecto único: você pode baixar a *engine* e incorporá-la em qualquer aplicação desejada.

Isso não está restrito em rodar em um navegador. Então Node atualmente usa o motor JavaScript V8 escrito pela Google e propõe que seja usado no servidor.



- O NodeJS é um aplicativo Single Thread, que opera em uma plataforma criada por C ++. Esta plataforma usa multi-thread para executar tarefas ao mesmo tempo.

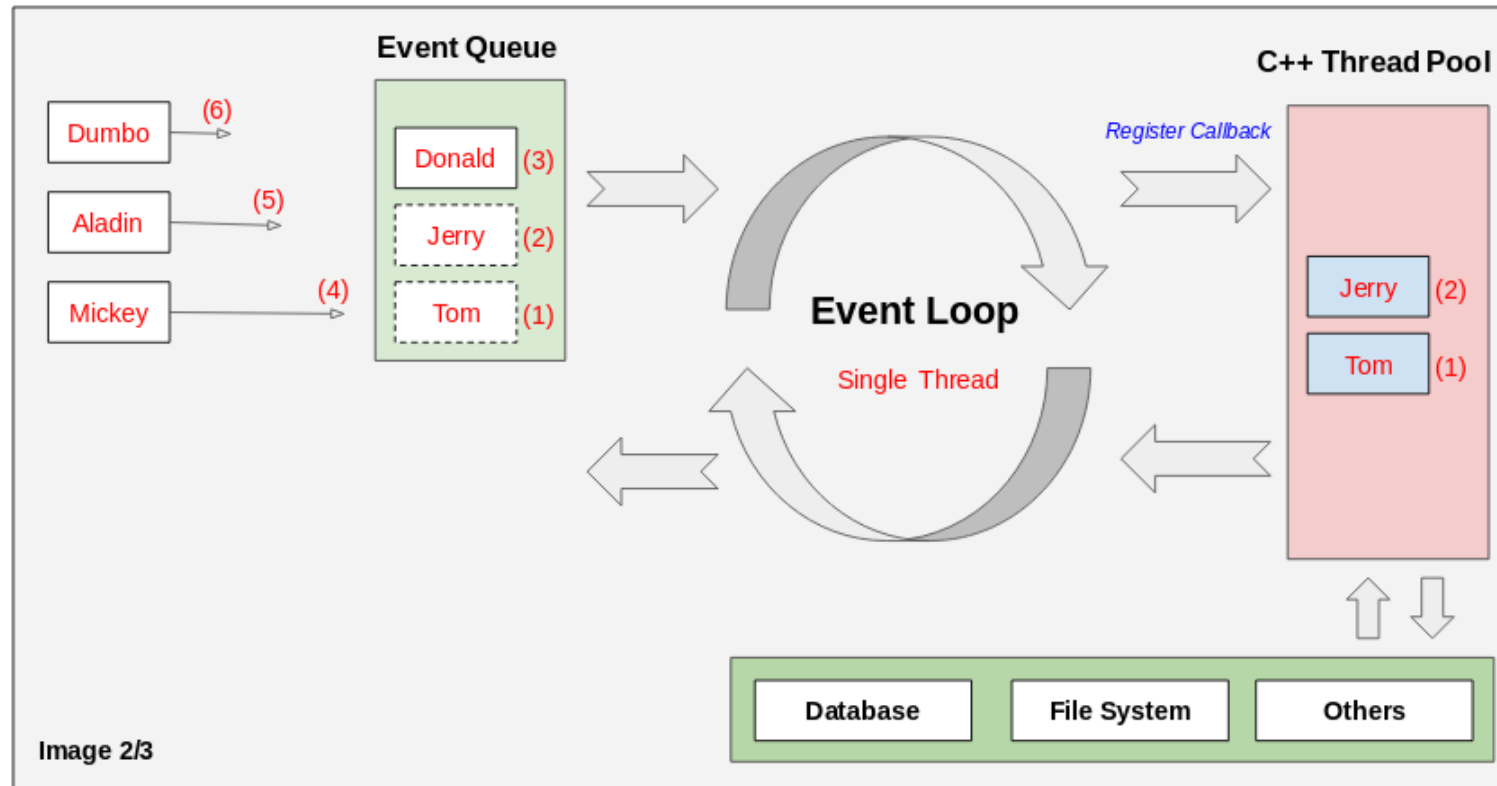
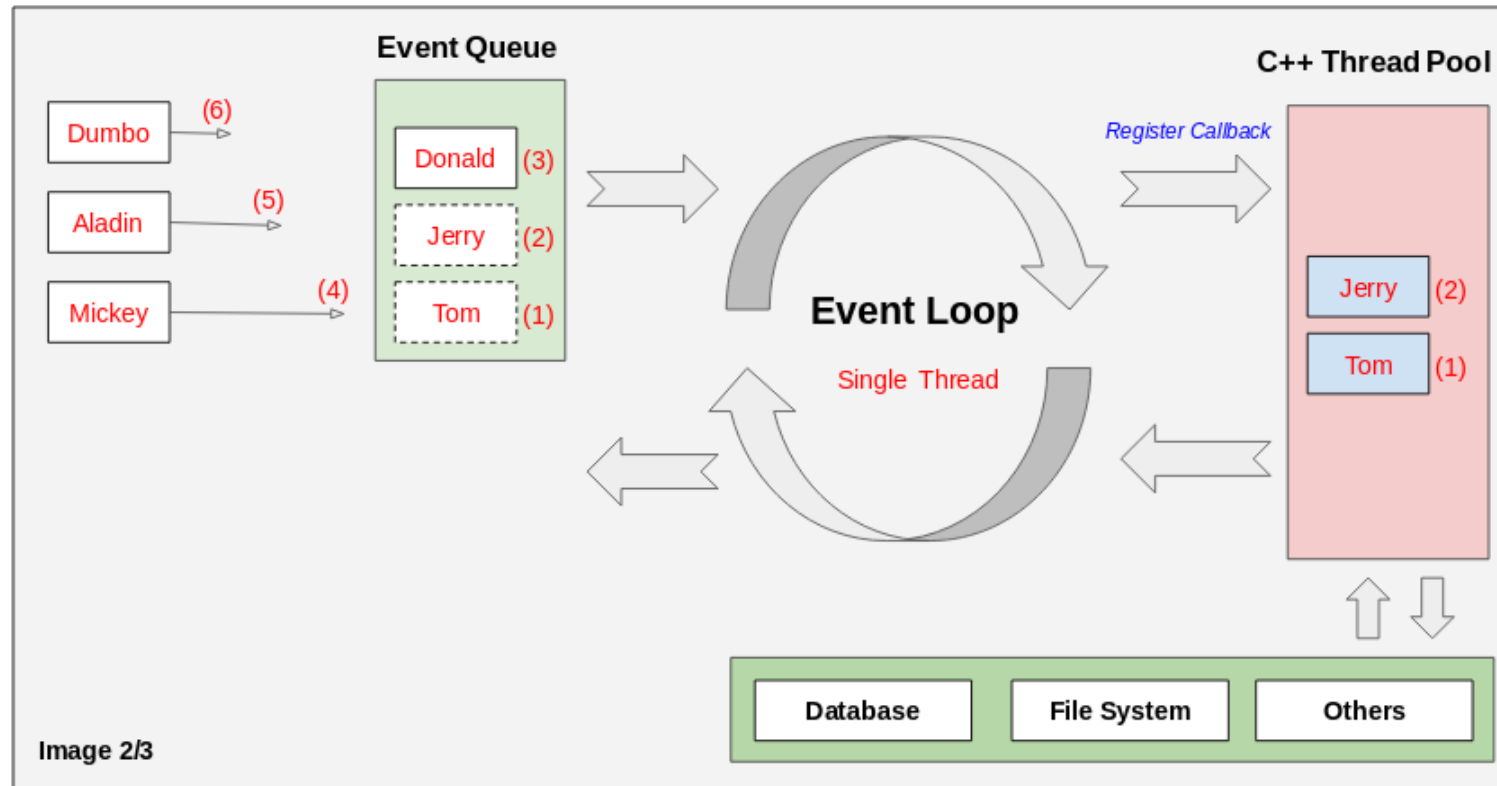
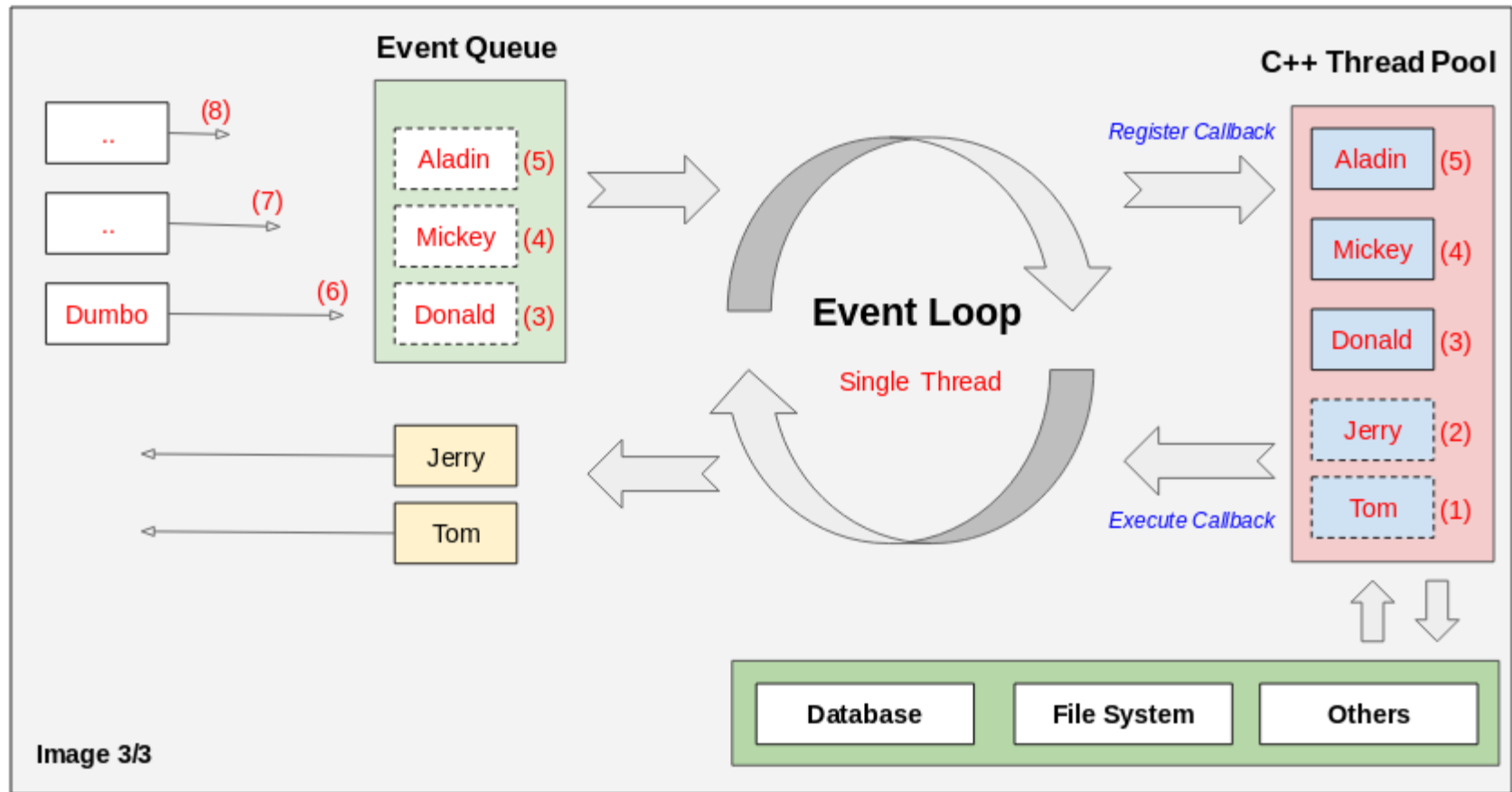


Image 2/3

- **Loop de Eventos:** Como um loop sem fim, ele passa as solicitações para o pool de threads e cada solicitação é registrada como uma função de retorno de chamada. Quando uma solicitação terminar de processar, a função de retorno de chamada correspondente será chamada para ser executada.



- Sendo um programa em linguagem C ++ , ele suporta vários threads. Portanto, neste documento, as solicitações serão tratadas em diferentes segmentos. O NodeJS também suporta Multi Processes. Isso significa que eles podem ser executados em núcleos diferentes.



- Quando uma solicitação termina o manuseio. O NodeJS chamará a função de retorno de chamada (registrada para esta solicitação) para executá-la.

Node orientado a Evento

- A principal filosofia por trás do Node é a programação orientada a eventos.
- Uma conexão é feita – evento! Dado é recebido através da conexão – evento! Dado parou de chegar através da conexão – evento!

Node orientado a Evento - Programação Assíncrona

- Na programação assíncrona, eventos assíncronos são executados independentemente do fluxo do programa principal.
- Ao invés de não fazer nada enquanto espera por um evento, o programa vai colocar o evento na fila do manipulador de eventos e continuar com o fluxo do programa principal.
- Quando o evento estiver pronto, o programa irá retornar para ele, via **callback**, executar o código e retornar para o fluxo principal do programa.
- Por isso, um programa assíncrono não vai executar de cima pra baixo como normalmente você vê em **códigos síncronos**.

Node orientado a Evento - Programação Assíncrona

- O Node usa *callbacks* assíncronos, também conhecidos como *callbacks não bloqueantes*, que permitem a continuidade do fluxo do programa em quanto são executadas operações como as de **I/O**. Quando a operação estiver concluída, ele vai emitir uma interrupção/*callback* para avisar ao programa que está pronto para executar e quando terminar, o programa volta para o que estava fazendo..

- APIs e Scripts
- Backend de aplicações Web dinâmicas
- Backend de Games
- Backend de Aplicações Escalonáveis
- IoT
- Mensageria

NPM e Packages

- Um dos benefícios do Node é o seu gerenciamento de pacotes, **npm**.
- Múltiplos pacotes podem ser reunidos e integrados para criar uma série de aplicações complexas. Alguns dos pacotes mais populares usados no Node:
 - ExpressJS: é atualmente um dos mais populares no site npm e nós ainda vamos falar bastante dele, é claro.
 - Mongoose: é um pacote que usamos para interagir com o MongoDB.
 - GruntJS: usado para automação de tarefas.
 - PassportJS: para autenticação em vários serviços de media social.
 - Socket.io: para construir aplicações websocket em tempo real.
 - Elasticsearch: que provê alta escalabilidade em operações de busca.

- Existem diversos **frameworks** para Node. O Express é um dos mais populares.

Classes

- A partir do ES6 é possível declararmos classes nativas com Javascript da seguinte forma:

```
class App { constructor() { }  
  customMethod() {  
    console.log('Hey'); } }
```

```
class App extends AnotherClass {}
```

Classes

- A partir do ES6 (versão 6 de 2015) é possível declararmos classes nativas com Javascript da

```
class App { constructor() { }  
  customMethod() {  
    console.log('Hey'); } }
```

```
class App extends AnotherClass {}
```

Criando classes em JavaScript e Node.JS

- Uma classe é uma especificação, um tipo novo de objeto da sua aplicação. Por exemplo, uma classe Pessoa (**inicie classes sempre com letra maiúscula e no singular**) irá definir propriedades e funções comuns a pessoas da sua aplicação.
- **A primeira recomendação é que você use uma classe por arquivo JS**, transformando-o **em um módulo JS** que deverá ser requerido/importado onde se desejar usar essa classe.

Exemplo – Criando Classes com Node

- **Passo 1:** Vamos criar um novo projeto no Node. Neste projeto criaremos um arquivo com o nome Cliente.js

```
//Cliente.js
3 module.exports = class Cliente {
4   //propriedades e funções da classe aqui
5   constructor(nome, idade, email) {
6     this.nome = nome;
7     this.idade = idade;
8     this.email = email;
9     this.dataCadastro = new Date();
10  }
11 }
```

criar o construtor dessa classe, uma função especial que inicializa um objeto deste tipo, usando argumentos e processamentos internos para definir as suas propriedades.

Exemplo – Criando Classes com Node

- O uso da palavra reservada **constructor** somente pode ser usada nessa função e ela é disparada automaticamente quando criamos um **novo objeto Cliente usando a keyword new**, como em outras linguagens orientadas a objeto (Java, C#, etc).

Exemplo – Criando Classes com Node

- **Passo 2:** Para usar essa classe que criamos, usei o **require** no módulo **Cliente.js** em uma constante, e essa constante representa a classe em si. Usando o **operador new**, instanciei um novo cliente com nome Luiz, idade 31 e e-mail mostrado acima.

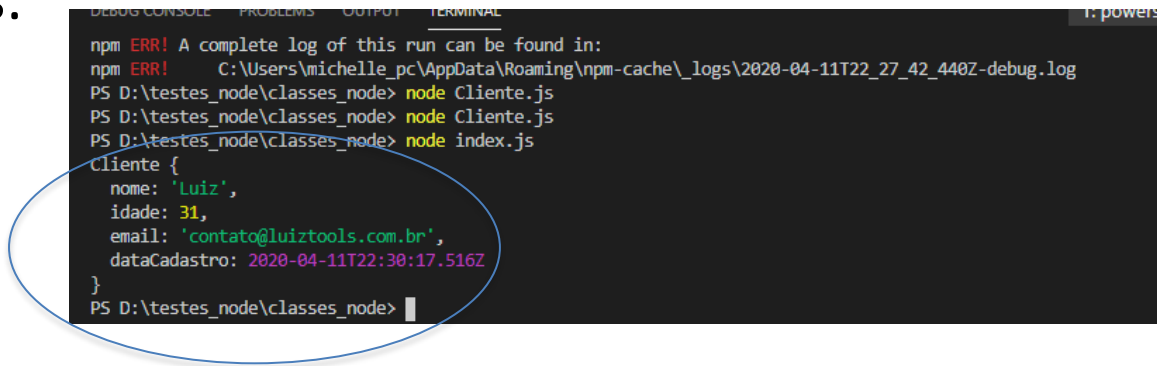
```
1 //index.js
2 const Cliente = require("./Cliente");
3 const cliente1 = new Cliente("Luiz", 31,
4 "contato@luiztools.com.br");
  console.log(cliente1);
```

Exemplo – Criando Classes com Node

- **Passo 3:** Executando dentro do Visual Studio Code, acesse: Terminal-> New Terminal.
- Execute os comandos:

node Cliente.js

node index.js



```
DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL
npm ERR! A complete log of this run can be found in:
npm ERR! C:\Users\michelle_pc\AppData\Roaming\npm-cache\_logs\2020-04-11T22_27_42_440Z-debug.log
PS D:\testes_node\classes_node> node Cliente.js
PS D:\testes_node\classes_node> node Cliente.js
PS D:\testes_node\classes_node> node index.js
Cliente {
  nome: 'Luiz',
  idade: 31,
  email: 'contato@luiztools.com.br',
  dataCadastro: 2020-04-11T22:30:17.516Z
}
PS D:\testes_node\classes_node>
```


Exemplo – Criando Classes com Node

- **Passo 4: Funções de classe em Javascript**
- *Toda classe é composta de propriedades e funções.* Essas funções, por uma questão de organização, devem ser **sempre relativas à responsabilidade da classe em si**, e geralmente manipulam ou utilizam as propriedades do objeto em questão.
- Não há necessidade da **palavra function** tradicionalmente usada, mas o restante segue a mesma lógica de **functions tradicionais**.

Exemplo – Criando Classes com Node

- Passo 4: Funções de classe em Javascript:
- Os objetos do tipo/classe Cliente possuem duas funções: **isAdult** que retorna **true/false** com base na idade do objeto e outra chamada **getFirstName** que baseada no nome do objeto/cliente, retorna a primeira parte do mesmo.

```
//Cliente.js
module.exports = class Cliente {
  //propriedades e funções da classe aqui
  constructor(nome, idade, email) {
    this.nome = nome;
    this.idade = idade;
    this.email = email;
    this.dataCadastro = new Date();
  }

  isAdult(){
    return this.idade >= 18;
  }

  getFirstName(){
    return this.nome.split(" ")[0];
  }
}
```

Exemplo – Criando Classes com Node

- **Passo 5:** Para chamar estas funções você primeiro deve instanciar objetos do tipo `Cliente` e suas execuções devem produzir retornos conforme propriedades de cada objeto em particular.

```
1 //index.js
2 const Cliente = require("./Cliente");
3 const cliente1 = new Cliente("Luiz", 31,
4 "contato@luiztools.com.br");
5 const cliente2 = new Cliente("Pedro", 5);
6 console.log(cliente1.nome + " é adulto? " +
  cliente1.isAdult());
  console.log(cliente2.nome + " é adulto? " +
    cliente2.isAdult());
```

Exemplo – Criando Classes com Node

- **Passo 6:** Execute novamente e veja os resultados

```
DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL
PS D:\testes_node\classes_node> node Cliente.js
PS D:\testes_node\classes_node> node index.js
Luiz é adulto? true
Pedro é adulto? false
PS D:\testes_node\classes_node> |
```

O segundo objeto está sem a instância de e-mail. Isso é totalmente permitido no JavaScript, embora possa causar alguma confusão. O ideal seria criar funções get e set para cada uma das propriedades internas das suas classes, para evitar que as mesmas sejam acessadas e manipuladas sem qualquer tipo de encapsulamento, uma vez que em JavaScript não temos modificadores de acesso como private, protected, etc

Exemplo – Criando Classes com Node

Passo 7: sobrescrita na orientação à objetos.

- Será criada uma nova Classe com o nome Endereco.js que contém a função toString, porém já existe esta função em todas classes JavaScript, mas estamos sobrescrevendo seu comportamento padrão através de declaração de outra função de mesmo nome
- Isso é uma das formas de Sobrecarga de método/função.

```
//Endereco.js
module.exports = class Endereco {
  constructor(rua, numero, bairro, cidade, uf) {
    this.rua = rua;
    this.numero = numero;
    this.bairro = bairro;
    this.cidade = cidade;
    this.uf = uf;
  }

  toString(){
    return this.rua + " " + this.numero + ", B. " + this.bairro
    + ", " + this.cidade + "/" + this.uf;
  }
}
```

Exemplo – Criando Classes com Node

Passo 8: inserindo o objeto do tipo endereço na classe Cliente

```
//Cliente.js
module.exports = class Cliente {
  //propriedades e funções da classe aqui
  constructor(nome, idade, endereco, email) {
    this.nome = nome;
    this.idade = idade;
    this.email = email;
    this.endereco = endereco;
    this.dataCadastro = new Date();
  }

  isAdult(){
    return this.idade >= 18;
  }

  getFirstName(){
    return this.nome.split(" ")[0];
  }
}
```

Exemplo – Criando Classes com Node

Passo 9: Para testar o uso de objetos **Endereco** como propriedade do **Cliente**, no arquivo **index.js**

```
//index.js
const Cliente = require("./Cliente");
const Endereco = require("./Endereco");

const enderecoLuiz = new Endereco("Tupis", 1
25, "São Vicente", "Gravataí", "RS");
const clienteLuiz = new Cliente("Luiz", 31,
enderecoLuiz, "contato@luiztools.com.br");
console.log(clienteLuiz.nome + " mora em " +
clienteLuiz.endereco);

const clientePedro = new Cliente("Pedro", 5)
;
clientePedro.endereco = new Endereco("Pedro
Vargas", 55, "Salgado Filho", "Gravataí", "R
S");
console.log(clientePedro.nome + " mora em "
+ clientePedro.endereco);
```

Exemplo – Criando Classes com Node

Passo 10: Testando

```
DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL
PS D:\testes_node\classes_node> node Cliente.js
PS D:\testes_node\classes_node> node Endereco.js
PS D:\testes_node\classes_node> node index.js
Luiz mora em Tupis 125, B. São Vicente, Gravatai/RS
Pedro mora em Pedro Vargas 55, B. Salgado Filho, Gravatai/RS
PS D:\testes_node\classes_node> 
```


Passo 11: Funções estáticas em JavaScript

Um componente estático (seja ele uma função ou propriedade) é compartilhado **entre todos objetos da mesma classe** e não necessita que a mesma seja instanciada para que o mesmo exista e possa ser usado/manipulado. Diferente das constantes, você pode mudar uma característica estática, mas se fizer isso, vai mudar para TODOS objetos que a possuem.

Exemplo – Criando Classes com Node

Passo 11: Para declarar uma propriedade estática, basta declará-la com a palavra-reservada **static**.

- Essa variável é da CLASSE e não do OBJETO, para usá-la você deve chamar “`classe.propriedade`”.
- É possível mudar o valor desta propriedade em tempo de execução ao contrário de **const**
- E o último ponto é que se mudá-la, muda para a CLASSE, independente dos OBJETOS, como no exemplo abaixo.

```
//Cliente.js
module.exports = class Cliente {
  //propriedades e funções da classe aqui
  constructor(nome, idade, endereco, email) {
    this.nome = nome;
    this.idade = idade;
    this.email = email;
    this.endereco = endereco;
    this.dataCadastro = new Date();
  }
  static idadeAdulto = 18;
  isAdult(){
    return this.idade >= 18;
  }

  getFirstName(){
    return this.nome.split(" ")[0];
  }
}
```

Exemplo – Criando Classes com Node

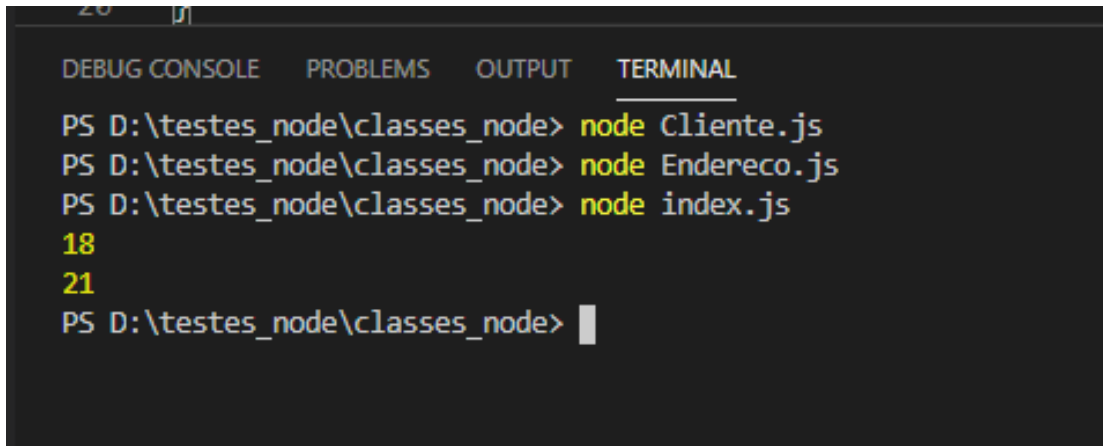
Passo 12: Note que independe de existir ou não algum objeto, a propriedade **idadeAdulto** existe para a CLASSE e, diferente de **const**, PODE ser modificada.

Ao usar **static**, você torna esta propriedade global para a classe.

```
//index.js
3 const Cliente = require("./Cliente");
4 const Endereco = require("./Endereco");
5
6 const enderecoLuiz = new Endereco("Tupis", 125, "São
7 Vicente", "Gravatá", "RS");
8 const clienteLuiz = new Cliente("Luiz", 31, enderecoLuiz,
9 "contato@luiztools.com.br");
1
0 console.log(Cliente.idadeAdulto);
  Cliente.idadeAdulto = 21;
  console.log(Cliente.idadeAdulto);
```

Exemplo – Criando Classes com Node

Passo 13: Teste a execução



```
DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL
PS D:\testes_node\classes_node> node Cliente.js
PS D:\testes_node\classes_node> node Endereco.js
PS D:\testes_node\classes_node> node index.js
18
21
PS D:\testes_node\classes_node> 
```

Dicas:

https://www.youtube.com/watch?time_continue=170&v=nTJc4Mjjqv4&feature=emb_logo

