09 - Utilizando .then() e .catch() em Promises

- Um cenário muito comum é que precisemos executar um código assim que a promise for resolvida, e para isso usamos o método **.then()**. Ele nos permite definir justamente a função "resolve" da promise, ou seja, a função que recebe o resultado como parâmetro e é chamada quando a promise é bem sucedida:

```
function execute() { return new Promise((resolve, reject) => {
console.log('A promise está sendo executada.') setTimeout(() => {
console.log('Resolvendo a promise...') resolve('Resultado') }, 3 *
1000) }) } const p = execute() p.then((result) => { console.log(`A
promise foi resolvida. O resultado foi: ${result}`) })
```

- O mesmo vale para quando uma promise é rejeitada. Para evitarmos o término da aplicação usamos o método **.catch()** para lidar com a rejeição da promise. Assim como no .then(), definimos o .catch() passando uma função como parâmetro que será justamente a função reject que será chamada em caso de rejeição, ou seja, ela própria tem como parâmetro o motivo da rejeição:

  Obs.: repare que encadeamos o .catch() no .then(), pois se não fizermos assim o código ainda resultará em erro. Ao encadearmos o .then() e o .catch() a rejeição da promise acontece da forma correta.

```
function execute() { return new Promise((resolve, reject) => {
console.log('A promise está sendo executada.') setTimeout(() => { if
(1 + 1 === 2) { reject('1 + 1 não é igual a 2') } else {
console.log('Resolvendo a promise...') resolve('Resultado') } }, 3 *
1000) }) } const p = execute() p.then((result) => { console.log(`A
promise foi resolvida. O resultado foi: ${result}`) }).catch((err) =>
{ console.log(`A promise foi rejeitada! Motivo: ${err}`) })
```

- Um detalhe que vale a pena mencionar é que a forma mais comum de definirmos os métodos .then() e .catch() em uma promise é encadeá-los diretamente na chamada da função:

```
// ... execute().then((result) => { console.log(`A promise foi
resolvida. O resultado foi: ${result}`) }).catch((err) => {
console.log(`A promise foi rejeitada! Motivo: ${err}`) })
```

- Por fim, assim como temos o try, catch e finally para tratamento de erros, podemos definir um método .finally() que segue o mesmo princípio, sendo executado quando a promise é finalizada, independente de se foi resolvida ou rejeitada:

  Obs.: repare que nesse caso a função de callback não possui nenhum parâmetro.

```
// ... execute().then((result) => { console.log(`A promise foi
resolvida. O resultado foi: ${result}`) }).catch((err) => {
console.log(`A promise foi rejeitada! Motivo: ${err}`) }).finally(()
=> { console.log('A promsie foi finalizada.') })
```