Gestion des erreurs en JavaScript
`try … catch`

Dernière mise à jour : 15/02/2024 09:16

Gestion des erreurs

- scripts contiennent parfois des erreurs
  - erreur de programmation (ex: appel d'une fonction non déclarée)
  - entrée incorrecte de l'utilisateur (ex: 12# au lieu de 123 alors qu'un entier est attendu)
  - réponse erronée d'un serveur
  - coupure réseau empêchant un fetch
  - ….
- Par défaut script s'arrête en affichant erreur dans la console

Syntaxe try … catch

- Possibilité de traiter les erreurs au sein d'un script

Pas d'erreur dans le script

code1 est exécuté

code2 est exécuté

le catch est ignoré

code3 est exécuté

```
//code1...

try {

    // code2...

} catch (err) {

    // Gestion des erreurs

}

// code3...
```

Une erreur dans code2

code1 est exécuté

lorsqu'une erreur intervient dans le bloc **`try`** l'exécution de code 2 est interrompue et le contrôle se place au début du bloc **`catch`**

exécution du code gestion des erreurs dans le bloc **`catch`**

code3 est exécuté

Une erreur dans un bloc try {...} ne tue pas le script – le bloc catch qui suis offre la possibilité de la gérer

Syntaxe try … catch

- Exemple

```
import { keyInYNStrict, question } from "readline-sync";

do {
    let jsonString = question("Entrez une chaîne JSON :");
    let obj = JSON.parse(jsonString);
    console.log("objet JavaScript correspondant : ");
    console.log(obj);
} while (keyInYNStrict("Encore "));

console.log("Au revoir...");
```

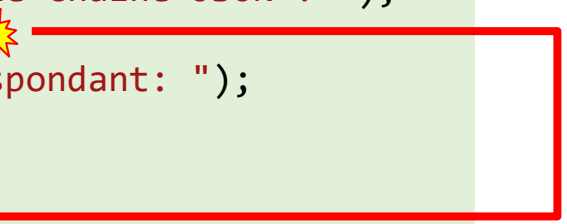
```
> node .\02ex2.mjs
Entrez une chaîne JSON : { "nom":"DUPOND" , "age":27}
objet JavaScript correspondant :
{ nom: 'DUPOND', age: 27 }
Encore [y/n]: y
Entrez une chaîne JSON : { nom: "DUPOND", age: 27 }
undefined:1
{ nom: "DUPOND", age: 27 }
  ^

SyntaxError: Unexpected token n in JSON at position 2
    at JSON.parse (<anonymous>)
    at file:///P:/ENSEIGNEMENT/M2CCI/exemples/02ex2.mjs:4:17
    at ModuleJob.run (node:internal/modules/esm/module_job:194:25)

Node.js v18.14.2
```

l'erreur arrête l'exécution

Syntaxe try … catch

- Exemple

```
import { keyInYNStrict, question } from "readline-sync";

do {
    try {
    let jsonString = question("Entrez une chaîne JSON : ");
    let obj = JSON.parse(jsonString);
    console.log("objet JavaScript correspondant: ");
    console.log(obj);
    }
    catch (err) {
        console.log("la chaîne JSON est incorrecte")
    }
} while (keyInYNStrict("Encore "));

console.log("Au revoir...");
```



l'erreur est attrapée par le `catch`

```
> node .\03ex3.mjs
Entrez une chaîne JSON : { "nom":"DUPOND" , "age":27}
objet JavaScript correspondant :
{ nom: 'DUPOND', age: 27 }
Encore [y/n]: y
Entrez une chaîne JSON : { nom: "DUPOND", age: 27 }
la chaîne JSON est incorrecte
Encore [y/n]: n
Au revoir...
```

try … catch fonctionne de manière synchrone

- que fait ce code ?

```
try {
    setTimeout(function () {
        JSON.parse('{name: "DUPONT", age: 12 }');
    }, 200);
} catch (err) {
    console.log("la chaîne JSON est incorrecte");
}

console.log("Au revoir...");
```

```
> node .\03ex3bis.mjs
Au revoir...
undefined:1
{name: "DUPONT", age: 12 }
 ^

SyntaxError: Unexpected token n in JSON at position 1
    at JSON.parse (<anonymous>)
    at Timeout._onTimeout (file:///P:/M2CCI/03ex3bis.mjs:3:8)
    at listOnTimeout (node:internal/timers:569:17)
    at process.processTimers (node:internal/timers:512:7)

Node.js v18.14.2
```

la fonction est exécutée ultérieurement, lorsque le moteur JavaScript a terminé l'exécution du code synchrone et déjà quitté la structure `try...catch`.

- Pour capturer une erreur produite dans une fonction planifiée (exécutée de manière asynchrone), le `try...catch` correspondant doit être à l'intérieur de cette fonction.

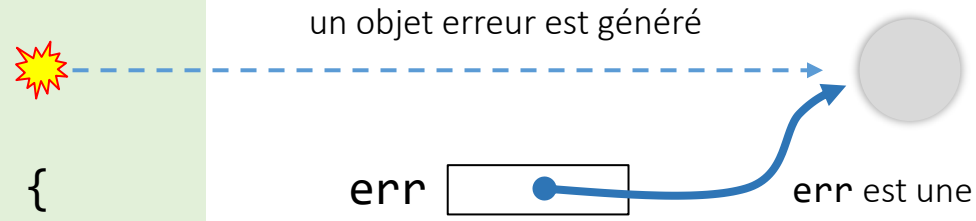
```
setTimeout(function () {
    try {
        JSON.parse('{name: "DUPONT", age: 12 }');
    } catch (err) {
        console.log("la chaîne JSON est incorrecte");
    }
}, 200);

console.log("Au revoir...");
```

```
> node .\03ex3ter.mjs
Au revoir...
la chaîne JSON est incorrecte
>
```

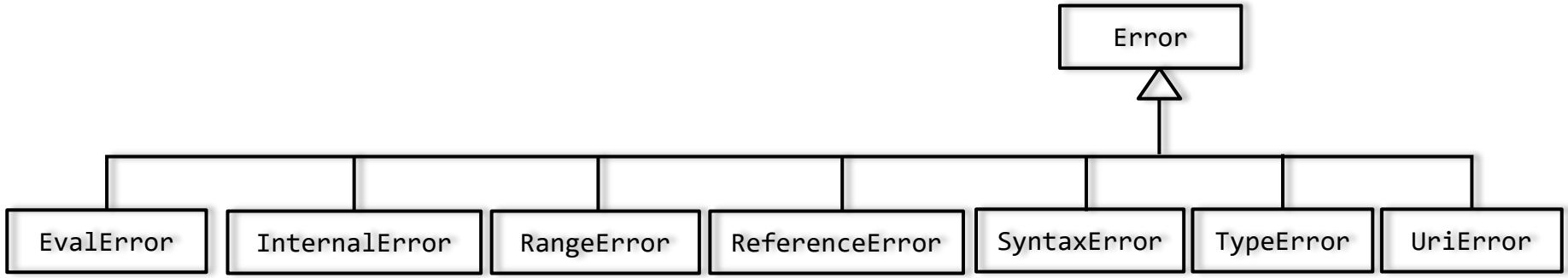
Objet d'erreur

- Quand une erreur intervient JavaScript crée un objet contenant les détails à son sujet.
- L'objet est ensuite passé en argument à `catch`
- Le nom utilisé en argument de catch est une référence vers l'objet erreur que vous pouvez ensuite utiliser dans le bloc `catch` pour analyser ce-dernier. Ce nom est laissé à la liberté du programmeur

```
try {

    //...

} catch (err) {

    // …

}
```



Les classes d'erreur standard (*built-in errors*) prédéfinies en JavaScript



Objet d'erreur

- Pour les erreurs prédéfinies (*built-in errors*) l'objet d'erreur possède deux propriétés standards :
  - **`name`**
    Nom de l'erreur. Par exemple, pour une variable non définie, il s'agit de "SyntaxError"
  - **`message`**
    Message textuel sur les détails de l'erreur

- Il existe d'autres propriétés non standard disponibles selon les environnements. L'une des plus largement utilisée et supportée est :
  - **`stack`**
    Pile d'exécution : chaîne contenant des informations sur la séquence d'appels imbriqués ayant entraîné l'erreur. Utilisé à des fins de débogage.

Objet d'erreur

- exemple

```
import { keyInYNStrict, question } from "readline-sync";
do {
  try {
    let jsonString = question("Entrez une chaîne JSON : ");
    let obj = JSON.parse(jsonString);
    console.log("objet JavaScript correspondant : ");
    console.log(obj);
  }
  catch (err) {
    console.log("la chaîne JSON est incorrecte");

    console.log(err.name);
    console.log(err.message);
    console.log(err.stack);

    console.log(err);
  }
} while (keyInYNStrict("Encore "));
console.log("Au revoir...");
```

```
> node .\04ex4.mjs
Entrez une chaîne JSON : { nom: "DUPOND", age: 24 }
la chaîne JSON est incorrecte
```

```
SyntaxError
```

```
Unexpected token n in JSON at position 2
```

```
SyntaxError: Unexpected token n in JSON at position 2
    at JSON.parse (<anonymous>)
    at file:///P:/ENSEIGNEMENT/M2CCI/04ex4.mjs:6:17
    at ModuleJob.run
(node:internal/modules/esm/module_job:194:25)
```

```
SyntaxError: Unexpected token n in JSON at position 2
    at JSON.parse (<anonymous>)
    at file:///P:/ENSEIGNEMENT/M2CCI/04ex4.mjs:6:17
    at ModuleJob.run
(node:internal/modules/esm/module_job:194:25)
```

Dans le navigateur utiliser plutôt `console.error()` que `console.log()`

Lancer ses propres erreurs

- Le programmeur a la possibilité de générer ses propres erreurs en utilisant l'opérateur `throw`

`throw <objet erreur>`

- n'importe quoi (valeur d'un type primitif, un chaîne de caractères, objet) peut être passé en argument de `throw`
- généralement il est préférable d'utiliser une référence vers un objet d'erreur héritant de la classe standard `Error`.
  - constructeurs prennent en paramètre un chaîne définissant le message
  - l'attribut `name` est le nom de la classe de l'erreur

```
let err;

err = new Error("une erreur standard");
console.log(err.message);
console.log(err.name)
err = new SyntaxError("une erreur de syntaxe");
console.log(err.message);
console.log(err.name)
err = new TypeError("une erreur de type");
console.log(err.message);
console.log(err.name);
…
```

```
une erreur standard
Error
```

```
une erreur de syntaxe
SyntaxError
```

```
une erreur de type
TypeError
```

Lancer ses propres erreurs

- exemple

```
import { keyInYNStrict, question } from "readline-sync";

function readUser() {
    console.log("Entrez un utilisateur");
    let jsonString = question("  chaîne JSON  : ");
    let user = JSON.parse(jsonString);
    return user;
}

do {
    try {
        let user  = readUser();
        console.log("nom de l'utilisateur " + user.name);
    } catch (err) {
        console.log('Erreur JSON ' + err.name + ' : ' +
                        err.message);
    }
} while (keyInYNStrict("Encore "));
console.log("Au revoir...");
```

```
> node .\06ex6.mjs
Entrez un utilisateur
  chaîne JSON  : { "name": "DUPOND", "age" : 27}
nom de l'utilisateur DUPOND
Encore [y/n]: y
Entrez un utilisateur
  chaîne JSON  : {"lastname":"DUPOND", "age" : 27}
nom de l'utilisateur undefined
Encore [y/n]: n
Au revoir...
```

L'absence de l'attribut **name** peut poser problème

Lancer ses propres erreurs

- exemple

```
import { keyInYNStrict, question } from "readline-sync";

function readUser() {
    console.log("Entrez un utilisateur");
    let jsonString = question("  chaîne JSON  : ");
    let user = JSON.parse(jsonString);
    if (! user.name) {
      throw new SyntaxError("Données incomplètes –
                                        pas de name");
    }
    return user;
}

do {
    try {
        let user  = readUser();          instruction non exécutée
        console.log("nom de l'utilisateur " + user.name);
    } catch (err) {
        console.log('Erreur JSON ' + err.name + ' : ' +
                        err.message);
    }
} while (keyInYNStrict("Encore "));
console.log("Au revoir...");
```

```
> node .\06ex6.mjs
Entrez un utilisateur
  chaîne JSON  : { "name": "DUPOND", "age" : 27}
nom de l'utilisateur DUPOND
Encore [y/n]: y
Entrez un utilisateur
  chaîne JSON  : {"lastname":"DUPOND", "age" : 27}
Erreur JSON SyntaxError : Données incomplètes - pas
de name
Encore [y/n]: n
Au revoir...
```

L'absence de l'attribut **name** provoque une erreur attrapée par le **catch**

Propagation (remontée) des erreurs

- Les erreurs remontent de bloc en bloc jusqu'à ce qu'un éventuel bloc `catch` l'attrape.
- Si aucun bloc `catch` n'est rencontré, le programme s'arrête

```
...
Encore [y/n]: y
Entrez un utilisateur
  chaîne JSON  :
{"lastname":"DUPOND", "age" : 27}
Erreur JSON SyntaxError : Données
incomplètes - pas de name
Encore [y/n]: y
...
```

```
import { keyInYNStrict, question } from "readline-sync";

function readUser() {
    console.log("Entrez un utilisateur");
    let jsonString = question("  chaîne JSON  : ");
    let user = JSON.parse(jsonString);
    if (! user.name) {
        throw new SyntaxError("Données incomplètes - pas de name");
    }
    return user;
}

function readUsers(n) {
    let res = [];
    for (let i = 0; i < n; i++) {
        res.push(readUser());
    }
    return res;
}

do {
    try {
        console.log("rentrez 3 users");
        let user  = readUsers(3);
    } catch (err) {
        console.log('Erreur JSON ' + err.name + ' : ' + err.message);
    }
} while (keyInYNStrict("Encore "));

console.log("Au revoir...");
```

Relancer un erreur non traitée

- Attention : un bloc **`catch`** attrape toutes les erreurs

```
> node .\09ex9.mjs
Entrez un utilisateur
  chaîne JSON  : {"name": "DUPOND",
"age": 27}
Erreur JSON ReferenceError : user
is not defined
Encore [y/n]: n
Au revoir...
```

```
import { keyInYNStrict, question } from "readline-sync";

function readUser() {
    console.log("Entrez un utilisateur");
    let jsonString = question("  chaîne JSON  : ");
    user = JSON.parse(jsonString);
    return user;
}

do {
    try {
        let user  = readUser();
        console.log("nom de l'utilisateur " + user.name);
    } catch (err) {
        console.log('Erreur JSON ' + err.name + ' : ' +
                err.message);
    }
} while (keyInYNStrict("Encore "));
console.log("Au revoir...");
```

`let` oublié avant **`user`**

Ici l'erreur n'est pas sur les données JSON mais concerne une erreur de programmation. Le bloc **`catch`** ne devrait par la traiter.

Relancer un erreur non traitée

- `catch` ne doit traiter que les erreurs qu'il connaît et "relancer" toutes les autres.

1. toutes les erreurs sont attrapées par **`catch`**.
2. Dans le bloc **`catch (err) {...}`** l'objet d'erreur **`err`** est analysé
3. Si on ne sait pas comment le gérer faire **`throw err`**.

```
> node .\10ex10.mjs
Entrez un utilisateur
  chaîne JSON  : {"name": "DUPOND", "age": 27}
file:///P:/M2CCI/10ex10.mjs:6
        user = JSON.parse(jsonString);
             ^

ReferenceError: user is not defined
    at readUser
(file:///P:/ENSEIGNEMENT/M2CCI/10ex10.mjs:6:7)
    at file:///P:/M2CCI/10ex10.mjs:12:15
    at ModuleJob.run
(node:internal/modules/esm/module_job:194:25)

Node.js v18.14.2
```

```
import { keyInYNStrict, question } from "readline-sync";

function readUser() {
    console.log("Entrez un utilisateur");
    let jsonString = question("  chaîne JSON  : ");
    user = JSON.parse(jsonString);
    return user;
}

do {
    try {
        let user  = readUser();
        console.log("nom de l'utilisateur " + user.name);
    } catch (err) {
        if (err instanceof SyntaxError) {
            console.log('Erreur JSON ' + err.name + ' : ' +
                        err.message);
        }
        else {
            throw err;
        }
    }
} while (keyInYNStrict("Encore "));
console.log("Au revoir...");
```

`let` oublié avant **`user`**

L'erreur est relancée, on sort du bloc **`catch`** et elle peut être soit capturée par une structure **`try...catch`** externe (si elle existe), soit elle arrête le script.

Bloc `finally`

- La structure `try {... } catch { … }` peut être suivie par un bloc de code supplémentaire : `finally { … }`
- Ce bloc, si il existe, s'exécute dans tous les cas :
  - après le bloc `try`, s'il n'y a pas eu d'erreur,
  - après le bloc `catch`, s'il y a eu des erreurs

```
import { keyInYNStrict, question } from "readline-sync";

do {
    try {
        console.log("début try");
        if (keyInYNStrict("on provoque une erreur ? ")) {
            foo();
        }
        console.log("fin try");
    } catch (err) {
        console.log("dans catch");
        console.log(err.name + " : " + err.message);
    } finally {
        console.log("dans finally");
    }
} while (keyInYNStrict("Encore "));
console.log("Au revoir...");
```

```
> node .\11ex11.mjs
début try
on provoque une erreur ? [y/n]: n
fin try
dans finally
Encore [y/n]: y
début try
on provoque une erreur ? [y/n]: y
dans catch
ReferenceError : foo is not defined
dans finally
Encore [y/n]: n
Au revoir...
```

Bloc `finally`

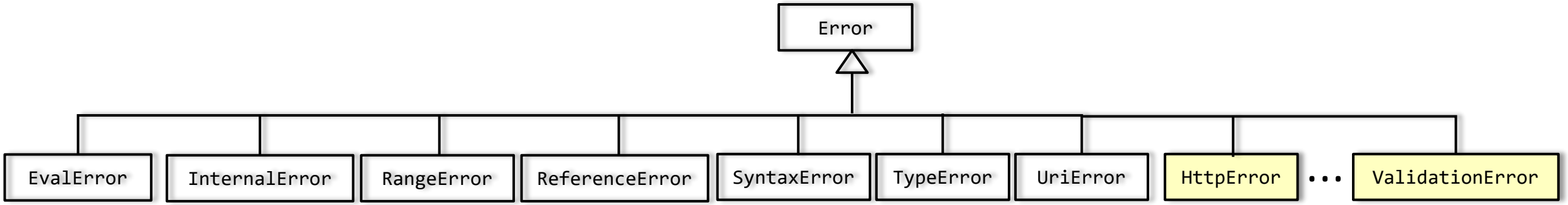
- le bloc `finally` est exécuté pour toute sortie de `try...catch`, y compris un `return` explicite.

```
import { keyInYNStrict, question } from "readline-sync";
function testFinally() {
    try {
        console.log("début try");
        if (keyInYNStrict("on provoque une erreur ? ")) {
            foo();
        }
        console.log("fin try");
        return new Date().toLocaleString();
    } catch (err) {
        console.log("dans catch");
        console.log(err.name + " : " + err.message);
    } finally {
        console.log("dans finally");
    }
}

do {
    console.log(testFinally());
} while (keyInYNStrict("Encore "));
console.log("Au revoir...");
```

```
> node .\12ex12.mjs
début try
on provoque une erreur ? [y/n]: n
fin try
dans finally
14/02/2024 00:42:43
Encore [y/n]: y
début try
on provoque une erreur ? [y/n]: y
dans catch
ReferenceError : foo is not defined
dans finally
undefined
Encore [y/n]: n
Au revoir...
```

Définir ses propres types d'erreur

- `throw` accepte n'importe quel type d'argument ➔possibilité pour le programmeur de définir ses propres types d'erreur pour signaler des problèmes spécifiques à son application :
  - HttpError : pour problème réseau
  - ValidationError : pour erreur de validation des données saisies
  - DataBaseError : pour erreur base de données
  - …
- Pour définir ces types d'erreur une bonne pratique est d'étendre (dériver) le type `Error`
  - permet d'hériter des propriétés standards `name`, `message` , `stack`
  - permet d'ajouter des propriétés spécifiques



Classes d'erreur standard (*built-in errors*) prédéfinies en JavaScript

Classes d'erreurs personnalisées

Définir ses propres types d'erreur

```
import { keyInYNStrict, question } from "readline-sync";

function readUser() {
    console.log("Entrez un utilisateur");
    let jsonString = question("  chaîne JSON  : ");
    let user = JSON.parse(jsonString);
    if (! user.name) {
         throw new SyntaxError("Données incomplètes –
                                    pas de name");
     }
    return user;
}

do {
    try {
        let user  = readUser();
        console.log("nom de l'utilisateur " + user.name);
    }
    catch (err) {
        if (err instanceof SyntaxError) {
           console.log('Erreur JSON ' + err.name + ' : ' +
                    err.message);
        }
        else {
          throw err;
        }
    }
} while (keyInYNStrict("Encore "));
console.log("Au revoir...");
```

instruction non exécutée

- exemple

Deux sortes d'erreur différentes

```
> node .\07ex7.mjs
Entrez un utilisateur
  chaîne JSON  : {"name": "DUPONT3" ; "age": 27 }
Erreur JSON SyntaxError : Unexpected token ; in JSON at 1
position 19
Encore [y/n]: yEncore [y/n]: y
Entrez un utilisateur
  chaîne JSON  : {"lastname":"DUPOND", "age" : 27}
Erreur JSON SyntaxError : Données incomplètes - pas de name 2
Encore [y/n]: n
Au revoir...
```

Un traitement unique

Dans le traitement des erreurs on aimerait pouvoir les distinguer

Définir ses propres types d'erreur

```
import { keyInYNStrict, question } from "readline-sync";

function readUser() {
    console.log("Entrez un utilisateur");
    let jsonString = question("  chaîne JSON  : ");
    let user = JSON.parse(jsonString);
    if (! user.name) {
         throw new ValidationError("Données incomplètes –
                                    pas de name");
     }
    return user;
}

do {
    try {
        let user  = readUser();
        console.log("nom de l'utilisateur " + user.name);
    }
    catch (err) {
        if (err instanceof ValidationError) {
           console.log('Erreur Validation ' + err.name + ' : ' +
                    err.message);
        }
        else if (err instanceof SyntaxError) {
           console.log('Erreur JSON ' + err.name + ' : ' +
                    err.message);
        }
        else {
          throw err;
        }
    }
} while (keyInYNStrict("Encore "));
console.log("Au revoir...");
```

1

2

- exemple

Deux sortes d'erreur différentes

```
> node .\07ex7.mjs
Entrez un utilisateur
  chaîne JSON  : {"name": "DUPONT3" ; "age": 27 }
Erreur JSON SyntaxError : Unexpected token ; in JSON at position 19
Encore [y/n]: yEncore [y/n]: y
Entrez un utilisateur
  chaîne JSON  : {"lastname":"DUPOND", "age" : 27}
Erreur Validation ValiationError : Données incomplètes - pas de name
Encore [y/n]: n
Au revoir...
```

1

2

Pour pouvoir les distinguer création d'une nouvelle classe d'erreur

```
class ValidationError extends Error {
    constructor(message) {
        super(message); // appel constructeur
                        // de la classe parente
        this.name = "ValidationError"; // pour affecter
                        // à la propriété name la bonne
                        // valeur (sinon cela serait
                        // Error)
    }
}
```