EXCEPTIONS

Pour commencer, on va créer 2 classes, User et Login

```php
namespace Class;

class User
{
    public function __construct(public string $username, public string $password)
    {
    }

    public function isVerified(): bool
    {
        return true;
    }
}
```

```php
namespace Class;

class Login
{
    public function __construct(protected User $user )
    {
    }

    public function login()
    {
        return true;
    }
}
```

Dans index.php

```php
use Class\Login;
use Class\User;

require('../vendor/autoload.php');

$user = new User( username: 'toto', password: 'password');
$login = new Login($user);
var_dump($login->login());
```

Pas de soucis, on retourne bien « true »

```
/app/public/index.php:10:boolean true
```

On va volontairement ne pas rentrer dans une condition pour retourner une erreur

Dans Login.php

```php
public function login(): bool|string
{
    if(!$this->user->isVerified()){
        echo "Erreur: utilisateur non vérifié";
    }
    return true;
}
```

Dans User.php

```php
public function isVerified(): bool
{
    return false;
}
```

Erreur: utilisateur non vérifié

On va maintenant utiliser la classe native de php => Exception, et remplacer le echo dans la fonction login()

```php
public function login(): bool|string
{
    if(!$this->user->isVerified()){
        throw new \Exception( message: 'Utilisateur non vérifié');
    }
    return true;
}
```

Fatal error: Uncaught Exception: Utilisateur non vérifié in /app/Class/Login.php on line *18*

Exception: Utilisateur non vérifié in /app/Class/Login.php on line *18*

Stack

Par défaut, PHP nous propose plusieurs classes pour gérer les erreurs, https://www.php.net/manual/fr/class.exception.php , mais on peut aussi créer nos propres classes de gestion d'erreurs.

On va créer un nouveau dossier dans Class => Exceptions et faire un fichier UserNotVerifiedException.php

```php
namespace Class\Exceptions;

class UserNotVerifiedException extends \Exception
{
    //Si le message est toujours le même
    //on peut le définir ici
    protected $message = 'Utilisateur non vérifié';
}
```

Dans Login.php on peut maintenant appeler la classe

```php
if(!$this->user->isVerified()){
    throw new UserNotVerifiedException();
}
```

Dans le navigateur, rien ne change hormis le nom de la classe appelé

Fatal error: Uncaught Class\Exceptions\UserNotVerifiedException: Utilisateur non vérifié in /app/Class/Login.php on line *20*

Class\Exceptions\UserNotVerifiedException: Utilisateur non vérifié in /app/Class/Login.php on line *20*

Uncaught signifie que l'erreur n'est pas « attrapé » (catch) pour régler cela il va falloir passer par des try / catch

La ou on appelle la méthode login (index.php) on passe la méthode dans un try / catch

```php
try {
    // on essaye de se logger
    $login->login();
// si on y arrive pas, on catch l'erreur
} catch (UserNotVerifiedException $e) {
    // on retourne le message d'erreur
    echo $e->getMessage();
}
```

Utilisateur non vérifié

On peut aussi customiser le message d'erreur avec plusieurs propriétés fourni par Exception

```php
// si on y arrive pas, on catch l'erreur
} catch (UserNotVerifiedException $e) {
    // on retourne le message d'erreur
    // On peut aussi l'améliorer
    echo $e->getMessage() .' sur la ligne '. $e->getLine() .' dans le fichier '. $e->getFile();
}
```

Utilisateur non vérifié sur la ligne 21 dans le fichier /app/Class/Login.php

On a aussi la possibilité d'utiliser plusieurs « catch » pour gérer les erreurs.

On crée une nouvelle classe exception

```php
namespace Class\Exceptions;

class UserIsBanException extends \Exception
{
    // cas où l'utilisateur est bani
    protected $message = 'Utilisateur est bani';
}
```

Dans login.php

```php
public function login(): bool|string
{
    if(!$this->user->isVerified()){
        throw new UserNotVerifiedException();
    }
    if($this->user->isBan()){
        throw new UserIsBanException();
    }
    return true;
}
```

```php
try {
    // on essaye de se logger
    $login->login();
// si on y arrive pas, on catch l'erreur
} catch (UserNotVerifiedException $e) {
    // on retourne le message d'erreur
    // On peut aussi l'améliorer
    echo $e->getMessage() .' sur la ligne '. $e->getLine() .' dans le fichier '. $e->getFile();
}catch (UserIsBanException $e) {
    // on retourne le message d'erreur
    // On peut aussi l'améliorer
    echo $e->getMessage() .' sur la ligne '. $e->getLine() .' dans le fichier '. $e->getFile();
}
```

Utilisateur est bani sur la ligne 25 dans le fichier /app/Class/Login.php

Dans ce cas ou les 2 exceptions étendent de la classe Exception, on peut faire du polymorphisme

```php
try {
    // on essaye de se logger
    $login->login();
// si on y arrive pas, on catch l'erreur
} catch (\Exception $e) {
    // on peut typer sur Exception,
    // car nos classes Exception étendent Exception
    // PHP sait retrouver la bonne exception crée
    echo $e->getMessage() .' sur la ligne '. $e->getLine() .' dans le fichier '. $e->getFile();
}
```

Maintenant on pourrait utiliser un environnement static pour gérer nos erreurs (Comme dans la plupart des frameworks)

On crée une nouvelle classe UserException et on va y mettre des méthodes

```php
namespace Class\Exceptions;

class UserException extends \Exception
{
    public static function notVerified():static
    {
        return new static( message: 'Utilisateur non vérifié (en mode static)');
    }
}
```

Dans User.php

```php
public function isVerified(): bool
{
    return false;
}
```

Dans Login.php on commente isBan et on « throw » la méthode static

```php
        if(!$this->user->isVerified()){
//            throw new UserNotVerifiedException();
            throw UserException::notVerified();
        }
```

Utilisateur non vérifié (en mode static) sur la ligne 9 dans le fichier /app/Class/Exceptions/UserException.php

Dans index.php

```php
// On utilise notre classe UserException
} catch (UserException $e) {
    echo $e->getMessage() . ' sur la ligne ' . $e->getLine() . ' dans le fichier ' . $e->getFile();
}
```

CONCLUSION:
On utilisera de préférences les Exceptions pour récupérer des erreurs critique lié au code (ex: connexion en BDD) on ne les utilisera pas pour récupérer une erreur de saisi d’email ou de mot de passe lors d’une authentification.