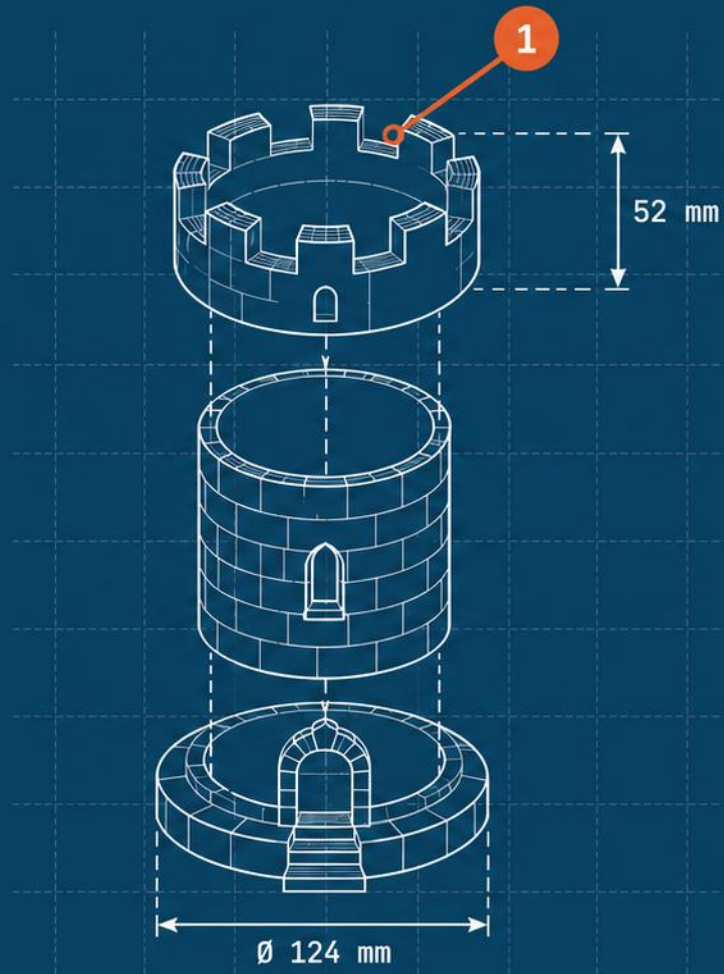


Bloc P00 · chapitre 0

Pourquoi la P00 avant Laravel

```
class Post extends Model { use HasFactory; }
```

Cette ligne, vous l'écrirez en semaine 4.
Aujourd'hui, on apprend à la lire.



Vingt lignes de Laravel, un : le contrôleur

1

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Models\Post;
6  use Illuminate\View\View;
7
8  class PostController extends Controller
9  {
10     public function index(): View
11     {
12         $posts = Post::latest()->get();
13
14         return view('posts.index', ['posts' => $posts]);
15     }
16 }
```

app/Http/Controllers/PostController.php, tel qu'il sort d'un projet neuf. Vous l'écrirez en semaine 4.

Vingt lignes de Laravel, deux : le modèle

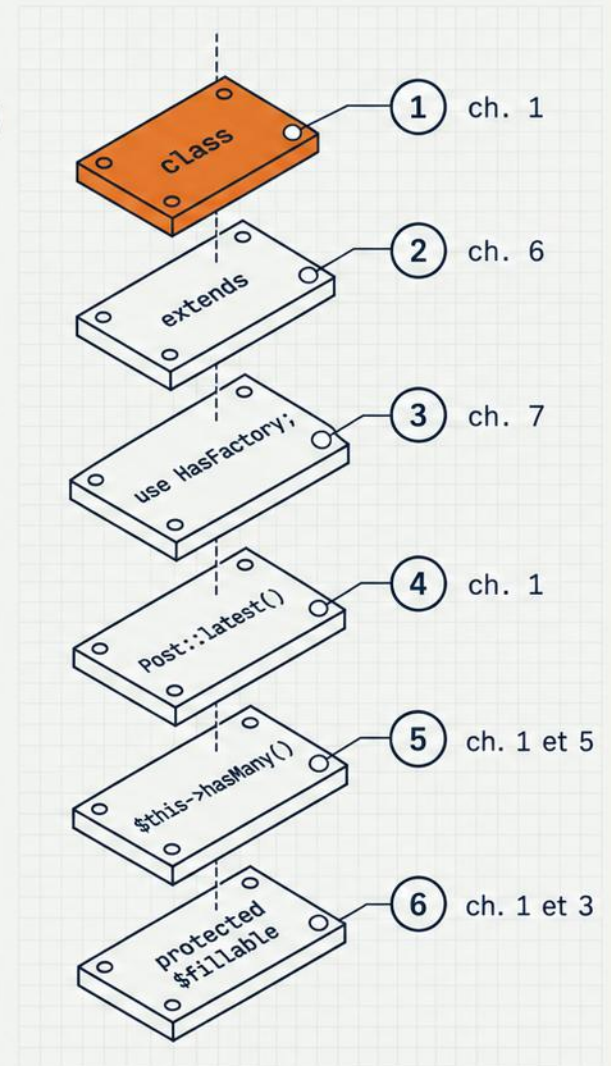
```
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7  use Illuminate\Database\Eloquent\Relations\HasMany;
8
9  class Post extends Model
10 {
11     use HasFactory;
12
13     protected $fillable = ['title', 'body'];
14
15     public function comments(): HasMany
16     {
17         return $this->hasMany(Comment::class);
18     }
19 }
```

Aucune boucle, aucun calcul écrit par vous. La requête SQL existe, mais c'est un objet de Laravel qui la construit. Ces lignes déclarent des objets et des liens entre eux.

Pour lire ces vingt lignes, il faut huit mots

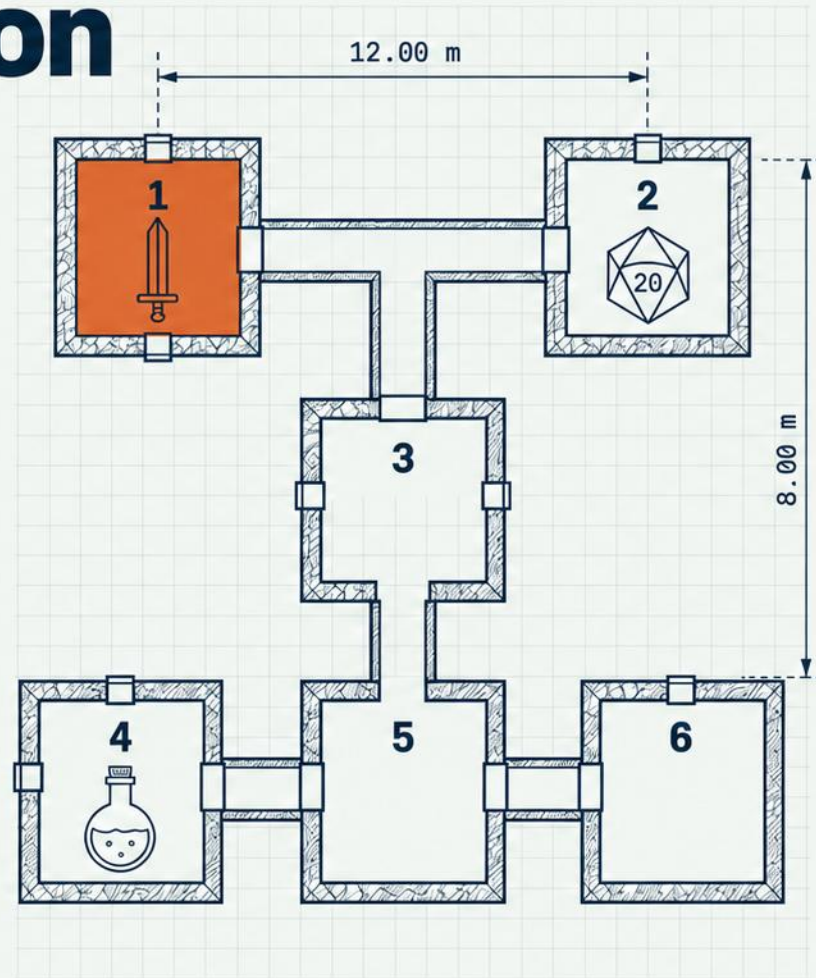
- `class` → une **classe**, le moule dont on tire des objets • ch. 1
- `extends` → l'**héritage** • ch. 6
- `use HasFactory;` → un **trait** • ch. 7
- `Post::latest()` → un appel **statique**, adressé à la classe • ch. 1
- `$this->hasMany()` → un appel sur une **instance** • ch. 1 et 5
- `protected $fillable` → l'**encapsulation** • ch. 1 et 3

Le but des deux prochaines semaines : rendre ces vingt lignes évidentes.
Rien de plus.



Un seul décor : le Donjon

1. Objets et classes, l'atelier (Composer, Pest, CI), kata niveau 1
2. Encapsulation, UML, relations entre objets, kata niveau 2
3. Héritage, polymorphisme, interfaces, traits, enums, katas 3 et 4
4. Atelier Formes : dessiner en SVG avec des objets
5. Atelier Collection, lire du code Laravel
6. Revue de code du Donjon complet, codé à la maison, et bilan du bloc



Un héros, un dé, un sac, des monstres. Un seul décor, pour que votre attention aille aux concepts.

Cinq périodes, quatre temps

1. Un code qui sent mauvais.

Cinq minutes, en binôme.

Ça marche, mais quelque chose cloche.

2. La théorie.

Une lecture courte,

des exemples à lancer chez vous.

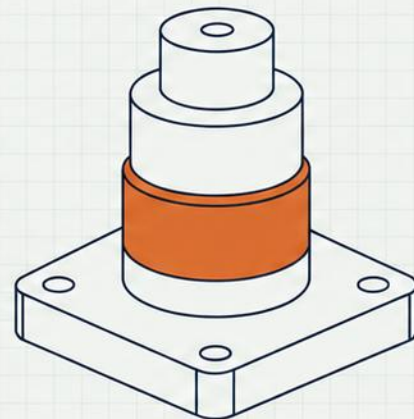
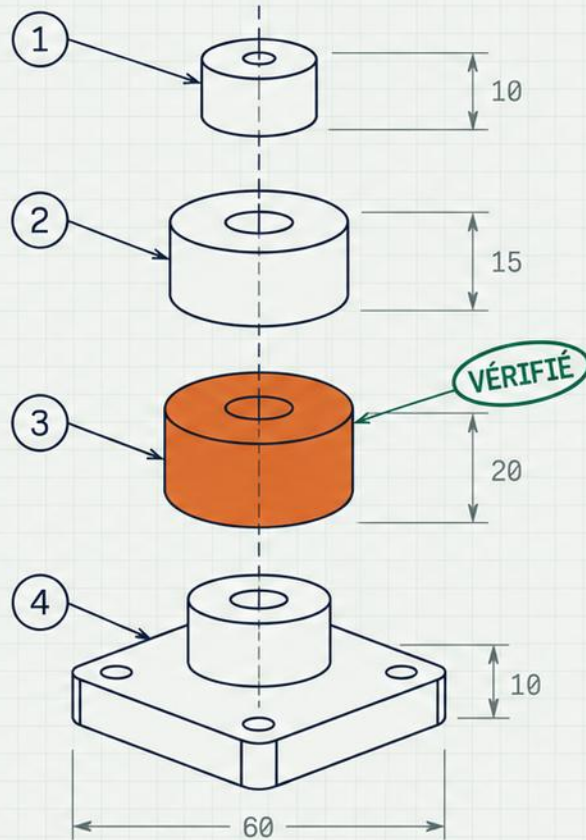
3. Le kata.

Un exercice court qu'on répète,
comme en arts martiaux : des tests
déjà écrits, des classes vides.

Vert, vous continuez. Rouge, le message
dit où regarder.

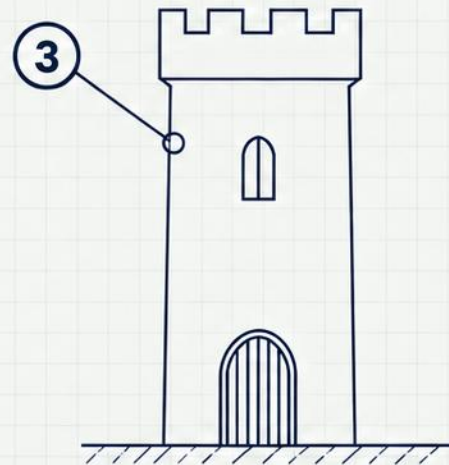
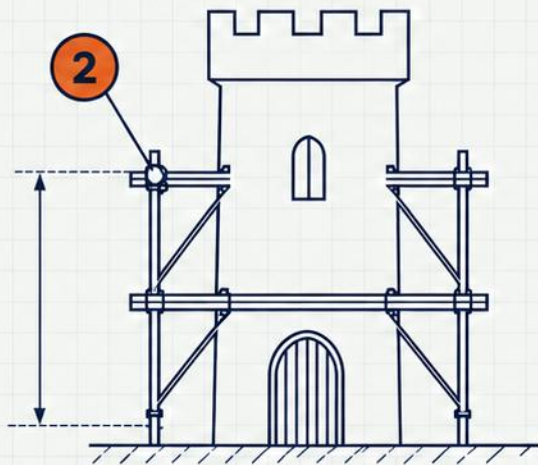
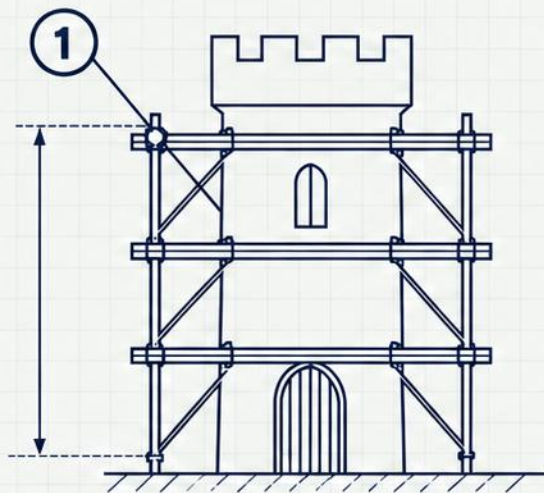
4. La revue.

On lit ensemble le code d'un binôme
volontaire.



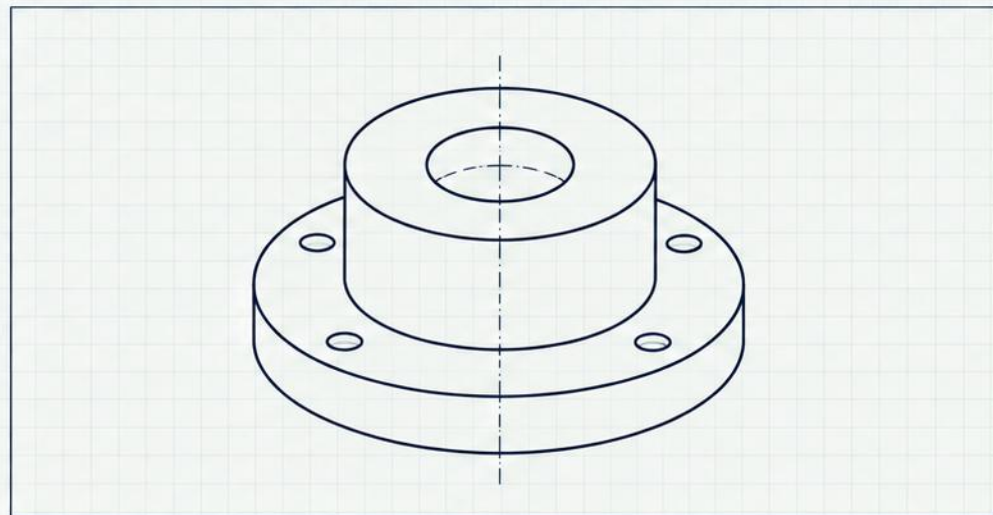
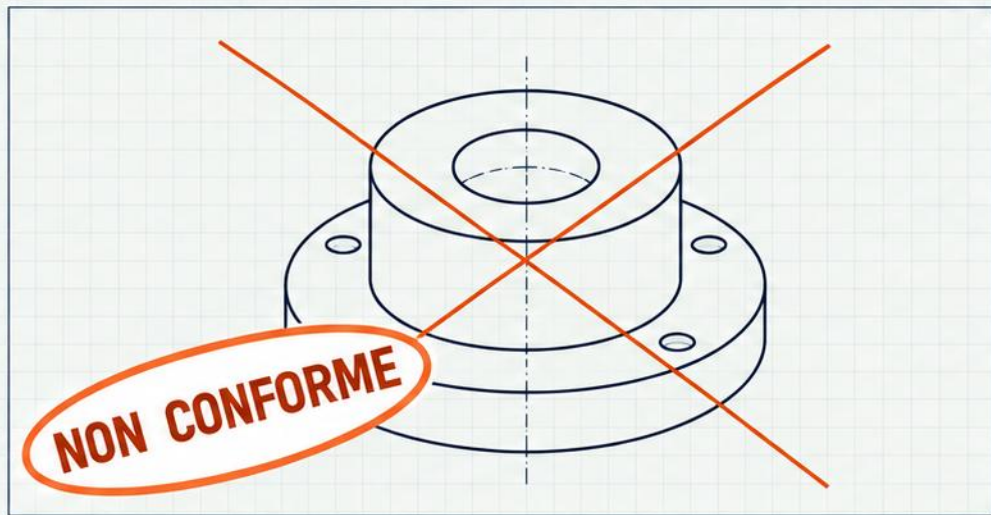
Une piste. Un vrai essai. Puis la solution expliquée.

- Vous bloquez → vous recevez **une piste** : une question, un nom de fonction
- Vous repartez avec ça et vous faites **un vrai essai**
- Toujours bloqué → **la solution, expliquée** : le code et le raisonnement
- Les premiers katas sont très guidés. Les suivants de moins en moins, jusqu'au chapitre 10, feuille blanche.



L'IA est permise. Une seule règle : pouvoir réécrire seul ce que vous rendez

- **Bonne façon** : lui demander d'expliquer un message d'erreur, ou de critiquer votre code, puis vérifier en lançant les tests
- **Mauvaise façon** : coller l'énoncé, recopier la réponse
- Au chapitre 10, que vous ferez seuls, vous noterez les questions posées et ce que vous en avez gardé



5XCOS · P00

PL. 0.08

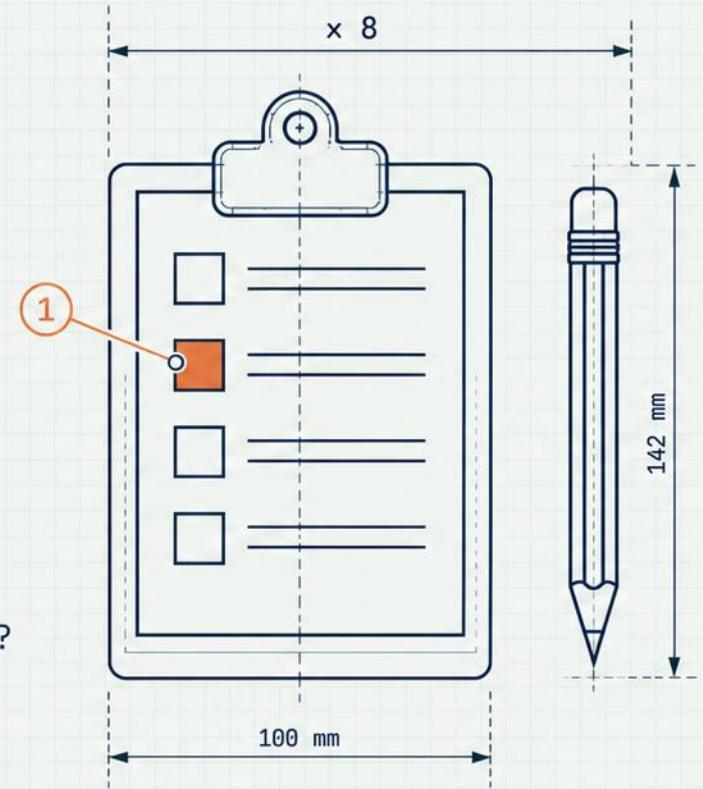
IA

éch. 1:1

Huit questions, cinq minutes, aucune note

- ☐ Que renvoie `strlen('donjon')`, et de quel type ?
- ☐ **return** ou **echo** : quelle différence dans une fonction ?
- ☐ Tableau indexé, tableau associatif : quelle différence ?
- ☐ Que vaut `'10' == 10` ? Et `'10' === 10` ?
- ☐ Que signifie `?int` dans fonction **findHero(string \$name): ?int** ?

Trois erreurs ou plus : relisez le chapitre Introduction avant la séance 2.



Trois choses avant la séance 2

1. **Faites le quiz** sans relire vos notes.
Comptez vos erreurs.
2. **Relisez les vingt lignes de Laravel** et écrivez en français ce que fait `Post::latest()->get()`.
Gardez votre réponse : on la ressortira au chapitre 10.
3. **Préparez votre machine** : un compte GitHub, un terminal où `php --version` et `composer --version` répondent.

Le chapitre 1 commence tout de suite : un héros, en tableau puis en objet.

