Blade Components Deep — Dobara Istemal Hone Wale Hisse

Build a piece once, use it everywhere: class and anonymous components, props, slots and named slots, $attributes merging and the rule of three.

Introduction

Your Students app now repeats itself: the same student card on the list page, the search results and the teacher page; the same green banner markup in five files. LV.3's `@include` helped — but includes are dumb copies: no inputs, no defaults, no manners. **Blade components** are includes that grew up: named pieces with their own **props** (inputs), **slots** (content you pour in) and the ability to merge the caller's own classes and attributes.

The working rule is the **rule of three**: the first time you build a piece, build it; the second time, copy it; the third time you reach for the same markup — stop, and make it a component. From then on it exists exactly once, and fifty pages share it. This lesson builds both kinds — class components and anonymous ones — and the layout component that can retire `@extends` entirely.

(Roman Urdu: Jo tukra teesri baar likhne lago, use component bana lo — aik baar banao, har jagah lagao, aik jagah badlo, har jagah badal jaye.)

Key points

- **Two kinds of components: class components** (`php artisan make:component StudentCard`) pair a PHP class with a view — use them when the piece needs logic or computed data. **Anonymous components** are view-only files in `resources/views/components/` — use them for pure markup with props.
- **Usage looks like HTML:** `<x-student-card :student="$student" />` — the `x-` prefix, your data passed as attributes. The colon `:` means "this is PHP data", without it the value is a plain string.
- **Props are declared inputs:** class components receive them through the constructor; anonymous ones declare `@props(['type' => 'success'])` — with defaults, so callers only pass what differs.
- **Slots are pour-in content:** whatever sits between `<x-alert> ... </x-alert>` arrives as `$slot`. **Named slots** (`<x-slot name="title">`) give a component several pour-in holes — a card with a title hole and a body hole.

  (Roman Urdu: Prop darwazay se di gayi cheez hai; slot andar bhari gayi cheez — component khali dabba hai, aap usay sajate hain.)
- **`$attributes` merges the caller's wishes:** attributes you did not declare as props (a class, an id, a data-*) collect in `$attributes`. `$attributes->merge(['class' => 'btn'])` prints your defaults PLUS the caller's extras — the caller's `class="mt-4"` joins, never replaces.

- **`<x-layout>` vs `@extends`:** a layout can itself be a component; pages then wrap content in `<x-layout> ... </x-layout>` instead of juggling `@section` names. Both are honest — components shine when the layout also takes props (a page title, a wide/narrow flag).
- **Components are not for everything:** a piece used once, in one page, is just markup. The rule of three protects you from component-ing the whole app into confetti.

Code examples

Example 1 — A class component: the student card, built once

```
php artisan make:component StudentCard
```

```
// app/View/Components/StudentCard.php
class StudentCard extends Component
{
    public function __construct(public Student $student) {}

    public function render()
    {
        return view('components.student-card');
    }
}
```

```
{{-- resources/views/components/student-card.blade.php --}}
<div class="card">
    <h3>{{ $student->name }}</h3>
    <p>{{ $student->city }} — {{ $student->marks }} marks</p>
    @if ($student->marks >= 90)
        <span class="badge">Topper</span>
    @endif
</div>
```

```
{{-- anywhere, as often as needed --}}
<x-student-card :student="$student" />
```

What it does: Artisan creates the pair — class + view. The constructor receives the student (typed, so mistakes error loudly). The view renders the card, Topper logic included — once, inside the component, instead of in every page that shows a student.

What you will see: the list page, the teacher page and the search results all stamp the identical card. Change the card's design in its one view file and every page updates together — the LV.3 dream, completed.

Example 2 — Anonymous component with props and slots: the alert box

```
{{-- resources/views/components/alert.blade.php --}}
@props(['type' => 'success'])

<div class="alert alert-{{ $type }}">
    @if (isset($title))
        <strong>{{ $title }}</strong>
    @endif
    <p>{{ $slot }}</p>
</div>
```

```
{{-- in any page --}}
<x-alert type="success">
    <x-slot name="title">Saved!</x-slot>
    Student added to the class list.
</x-alert>

<x-alert type="danger">
    Marks must be between 0 and 100.
</x-alert>
```

What it does: No class at all — `@props` declares one input with a default of `success`. The body poured between the tags arrives as `$slot`; the named slot `title` is checked with `isset` and printed only when given. The second call passes no title and a different type — same component, different mood.

What you will see: a green box titled **Saved!** with its message, and below it a red box with the warning and no title. One file produced both — the LV.6 flash banner just retired into this component.

Example 3 — $attributes merging: a button that accepts your classes

```
{{-- resources/views/components/primary-button.blade.php --}}
@props(['href'])

<a href="{{ $href }}" {{ $attributes->merge(['class' => 'btn btn-primary']) }}>
    {{ $slot }}
</a>
```

```
<x-primary-button href="{{ route('students.create') }}" class="mt-4" id="addBtn">
    Add Student
</x-primary-button>
```

What it does: `href` is a declared prop; `class` and `id` are not — they land in `$attributes`. The merge prints the component's own classes AND the caller's: the caller's `mt-4` is *added* to `btn btn-primary`, not

swapped in its place, and the `id` rides along untouched.

What you will see: the page source shows one honest anchor:

```
<a href="http://127.0.0.1:8000/students/create" class="btn btn-primary mt-4" id="addBtn">Add Student</a>
```

Component defaults and caller wishes, married — no class ever lost in the merge.

Quick recap

- Components = reusable Blade pieces: class components (logic + view) and anonymous ones (view + `@props`).
- Props are inputs; `$slot` is poured-in content; named slots are extra holes.
- `$attributes->merge()` combines the component's defaults with the caller's own attributes.
- Rule of three: build, copy, then component. Layouts can be components too (`<x-layout>`).

Common mistakes

1. **Component-ing everything on day one** — forty tiny components for one-use markup makes the app unreadable confetti. Fix: the rule of three is the gate; earn each component with real repetition.
2. **Forgetting the colon on data props** — `<x-student-card student="$student" />` passes the literal TEXT "$student", not the object. Fix: `:student="$student"` whenever the value is PHP, no colon for plain strings.
3. **Replacing instead of merging attributes** — printing `class="btn"` and dropping the caller's `class="mt-4"` breaks the caller's page silently. Fix: always go through `$attributes->merge()` in shared components.
4. **Logic-heavy anonymous components** — loops, queries and calculations crammed into a view-only file. Fix: when a piece starts thinking, promote it to a class component and let the class do the thinking.

Practice

1. Retire the LV.6 flash banner into the Example 2 alert component and use it on the students index for both success and danger messages. Then restyle the alert once and watch every page agree.
2. Build an `<x-student-row>` component for the index table (props: student; the row includes Edit link and Delete form). Count the lines your index view loses — that number is the component's rent paid.

Practice quiz

1. What is the difference between class components and anonymous components?

- There is none

- Class components pair a PHP class with a view, for pieces needing logic or computed data; anonymous components are view-only files in resources/views/components/, for pure markup with props
- Anonymous components cannot take props
- Class components cannot be reused

Answer: Class components pair a PHP class with a view, for pieces needing logic or computed data; anonymous components are view-only files in resources/views/components/, for pure markup with props

2. What is the difference between <x-student-card student="$student" /> and <x-student-card :student="$student" />?

- There is none
- Without the colon the value is the plain string "$student"; the colon means this is PHP data, the object itself
- The colon makes it a string
- The first passes data, the second passes nothing

Answer: Without the colon the value is the plain string "$student"; the colon means this is PHP data, the object itself

3. What is $slot, and what are named slots?

- $slot is a prop like any other
- Whatever sits between <x-alert> ... </x-alert> arrives as $slot, poured-in content; named slots (<x-slot name="title">) give a component several pour-in holes
- $slot is the component's class name
- Named slots are only for layouts

Answer: Whatever sits between <x-alert> ... </x-alert> arrives as $slot, poured-in content; named slots (<x-slot name="title">) give a component several pour-in holes

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/laravel-blade-components · More free tools and notes at https://scopetreasure.com