Oops! I did it again

OOP for the Uninitiated

with Yorick Brown

TECHNATIVE DIGITAL
NEW TECH TALENT / FUTURE LEADERS

YB.
DIGITAL

Background

- A degree in International Politics doesn’t really help
- Learning on the job - using OOP in projects
- But do I really know it?
- And… it’s hard!

What, why, and how OOP

- Object-Oriented Programming
- What?
- Why?
- How?

Concepts shared across languages and OOP fundamentals

- **Class** – Blueprint for creating objects.
- **Object** – Instance of a class.
- **Property / Attribute** – Variables attached to an object.
- **Method** – Functions attached to a class.
- **Constructor** – A special method that runs when an object is instantiated.
- **Destructor** – Method called when an object is destroyed (common in PHP, C++, less explicit in Java, absent in JS).
- **Encapsulation** – Bundling data with methods, restricting direct access (private/public/protected).
- **Inheritance** – Extending classes (`extends` in PHP/JS/Java).
- **Polymorphism** – Objects can share methods but implement them differently.
- **Abstraction** – Abstract classes/methods force subclasses to implement behaviour.
- **Interface** – Contract defining methods a class must implement.
- **Static methods/properties** – Belong to the class, not an instance.
- **Namespaces / Modules** – Grouping related code under a logical scope (PHP uses `namespace`, JS uses `import/export`, Java uses `package`).

Things in common

Concept	PHP Syntax	JavaScript Syntax	Java Syntax	Shared?
Class	class MyClass { ... }	class MyClass { ... }	class MyClass { ... }	Yes
Constructor	function __construct()	constructor()	MyClass()	Yes, different names
Object	$obj = new MyClass();	let obj = new MyClass();	MyClass obj = new MyClass();	Yes
Inheritance	class Child extends Parent {}	class Child extends Parent {}	class Child extends Parent {}	Yes
Interface	interface MyInterface {}	Not built-in, use classes	interface MyInterface {}	Yes (PHP & Java)
Namespacing	namespace MyNamespace;	Modules/objects	package my.package;	Yes, different
Access modifiers	public,protected,private	public,private(ES6+)	public,protected,private	Yes
Dependency Injection	Not a language feature, done via constructor	Not a language feature, done via constructor	Framework support	Yes, via pattern
Singleton	Enforced in user code	Enforced in user code	Enforced in user code	Yes
Factory	Design pattern method	Function returning class	Factory class pattern	Yes
Helper	Class or function	Function or module	Static utility class/function	Yes

TECHNATIVE DIGITAL
NEW TECH TALENT / FUTURE LEADERS

YB. DIGITAL

Patterns

Design patterns

- **Dependency Injection**
- **Singleton**
- **Factory**
- **Adapter**
- **Decorator**
- **Observer**
- **Helper / Utility classes**