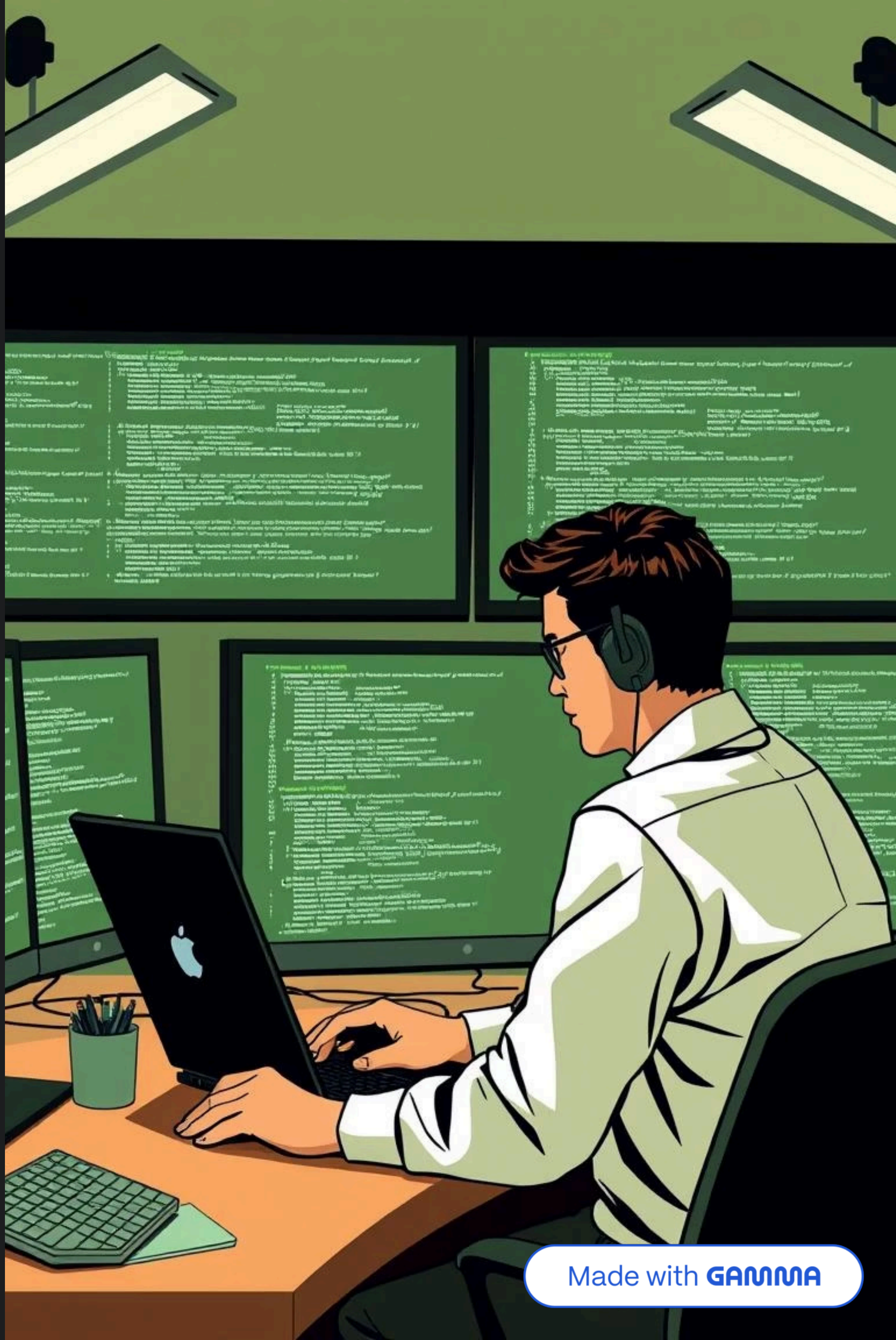
Principles of Object-Oriented Programming

Master the foundations of modern software design



What is Object-Oriented Programming?

Object-Oriented Programming (OOP) is a programming paradigm that structures code around objects and classes rather than functions and logic. It mirrors real-world entities, making code more intuitive, modular, and scalable.

Why OOP Matters

- Improved code organisation and reusability
- Easier maintenance and collaboration
- Better suited for large-scale applications

Core OOP Principles

Class

Blueprint defining structure and behaviour

Object

Instance created from a class

Encapsulation

Bundling data with methods

Abstraction

Hiding implementation complexity

Inheritance

Deriving new classes from existing ones

Polymorphism

Objects behaving in multiple forms

Classes and Objects Explained

Class: The Blueprint

A class is a template defining what attributes and methods an object should have. Think of it as architectural plans for a building—it specifies the structure but isn't the building itself.

Object: The Instance

An object is a concrete realisation of a class. It's the actual building constructed from those plans, with specific values and state.

Simple Analogy: Class = Cookie cutter shape; Object = individual cookies baked using that cutter.

Encapsulation: Protecting Your Data

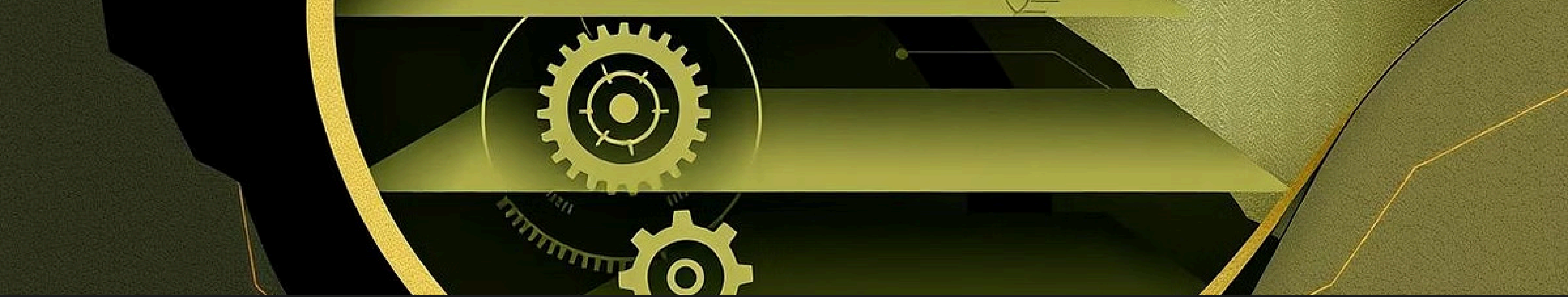
What is Encapsulation?

Encapsulation bundles related data and methods together, hiding internal details from the outside world. It controls access using private and public modifiers.

Real-World Analogy

A medicine capsule hides its ingredients inside. You take the capsule without knowing the exact composition—only the exterior matters to you.

Code Example: Private variables with public getter and setter methods control how data is accessed and modified, protecting data integrity.



Abstraction: Simplifying Complexity

Abstraction hides complex internal logic, exposing only essential features. It reduces complexity and allows users to interact with objects through simple, well-defined interfaces without understanding implementation details.

→ **Hide Complexity**

Internal mechanisms remain invisible to users

→ **Show Essentials**

Only necessary functionality is exposed

→ **Improve Usability**

Simpler interfaces make code easier to use and maintain

Inheritance: Building on Foundation

Inheritance allows a new class to derive properties and methods from an existing class, promoting code reuse and creating logical hierarchies.

1

Single Inheritance

Child class inherits from one parent class

2

Multilevel Inheritance

Grandparent → Parent → Child class chain

3

Real Example

`SportsCar` inherits speed and drive() method from parent `Car`, adding extra features like turbo()

Made with GAMMA

Polymorphism: Many Forms

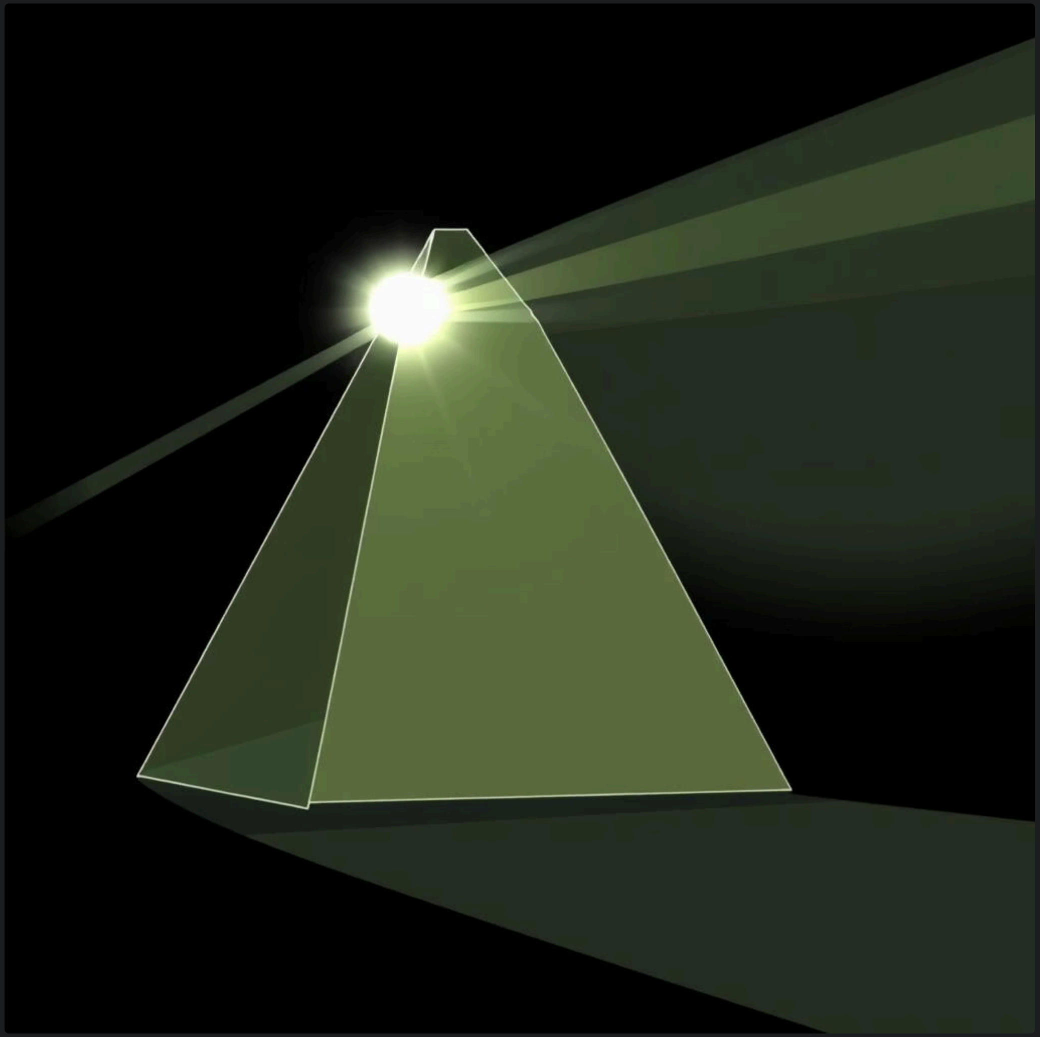
Polymorphism means "many forms"—same interface, different behaviour

Compile-Time (Static)

Method Overloading: Multiple methods with same name but different parameters. Java determines which to use during compilation.

Runtime (Dynamic)

Method Overriding: Subclass provides its own implementation of a parent method. Decision happens during execution based on object type.



Key Insight: Polymorphism lets you write flexible, generic code that works with multiple object types seamlessly.

Real-World OOP Applications

Object-Oriented Programming powers countless modern applications across industries:

Game Development

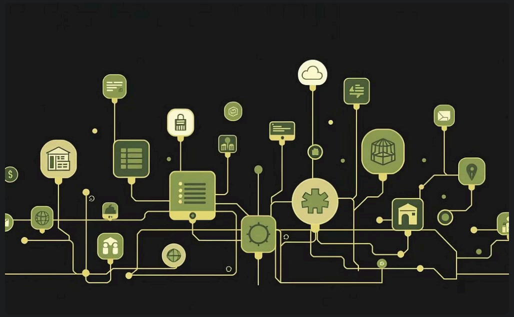
Characters, enemies, and items are objects with properties and behaviours

Mobile Apps

UI elements, data models, and services are all organised as objects

Web Development

Backend systems, databases, and frontend frameworks use OOP principles



Enterprise Software

Large systems managing business logic, users, and transactions as objects