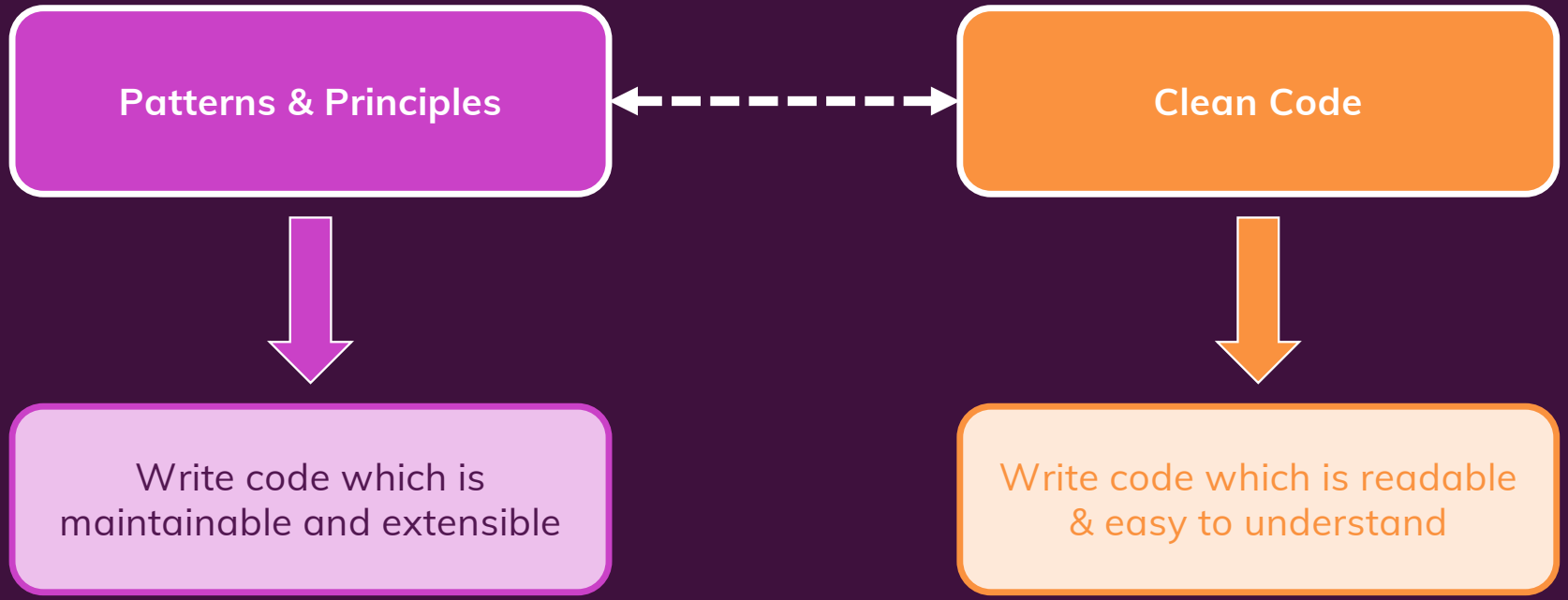
ACADEMIND

This Module Will Not Teach You OOP!

ACADEMIND

We Won't Dive Into All OOP Patterns & Practices

Clean Code and Patterns & Principles



ACADEMIND

The Difference Between Objects & Data Structures

Object

Private internals / properties, public API (methods)

Contain your business logic (in OOP)

Abstractions over concretions

Data Container / Data Structure

Public internals / properties, (almost) no API (methods)

Store and transport data

Concretions only

ACADEMIND

Polymorphism

The ability of an object to take on many forms.

Classes Should Be Small

You typically should prefer many small classes over a few large classes

Classes should have a single responsibility
Single-Responsibility Principle (SRP)

A Product class is responsible for product “issues” (e.g. change the product name)

Cohesion

How much are your class methods using the class properties?

Maximum Cohesion		No Cohesion
All methods each use all properties	Highly cohesive classes	All methods don't use any class properties
A highly cohesive object		Data structure / container with utility methods

Law Of Demeter

Code in a method may only access direct internals (properties and methods) of:

- the object it belongs to
- objects that are stored in properties of that object
- objects which are received as method parameters
- objects which are created in the method

ACADEMIND

Tell, Don’t Ask!

ACADEMIND

The SOLID Principles

S	Single Responsibility Principle
O	Open-Closed Principle
L	Liskov Substitution Principle
I	Interface Segregation Principle
D	Dependency Inversion Principle

ACADEMIND

The Single-Responsibility Principle

Classes should have a single responsibility – a class shouldn't change for more than one reason.

SRP & Clean Code

Restricting classes to one core responsibility leads to smaller classes

Smaller classes tend to be easier to read

ACADEMIND

The Open-Closed Principle (OCP)

A class should be open for extension but closed for modification.

OCP & Clean Code

Extensibility ensures small class (instead of growing classes) and can help prevent code duplication (DRY)

Smaller classes and DRY code increase readability and maintainability

ACADEMIND

The Liskov Substitution Principle (LP)

Objects should be replaceable with instances of their subclasses without altering the behavior.

ACADEMIND

The Interface Segregation Principle (ISP)

Many client-specific interfaces are better than one general purpose interface.

ACADEMIND

The Dependency Inversion Principle (DIP)

You should depend upon abstractions, not concretions.