Two types of projects: Executables (with main()) and class libraries. Argument for more projects: **Single Responsibility Principle**. Separating business logic in its own project has some benefits:

* Single project will quickly become unmaintainable if it gets large. Makes maintenance easier to split it.
* If support another type of device later, we could replace Web API layer with a new layer, keeping the business logic intact.

**Entity classes => MAP TO DATABASE**

Why not just return them?

* By **exposing** entity classes, we are EXPOSING OUR DB DESIGN. And the DESIGN OF THE DB MAY NOT MATCH THE MODEL WE WOULD LIKE TO USE IN OUR API.
* Entity classes may contain **circular refs** to other classes, WILL CAUSE TROUBLE WHEN SERIALIZED TO JSON.

I.e., when returning **list of courses**.. client MAY ONLY BE INTERESTED IN A FEW PROPERTIES.

May also NOT HAVE ALL THE FIELDS.. **f.ex. CourseTemplate**.. which didn‘t have all course info. Nor did Course itself.. (CourseTemplate with name.. vs course instances.. no duplication)

**ID OF OBJECT IS GENERATED BY DB.. for instance.. some are calculated like created date..** s

**DTO (Data Transfer Object):** Or...

**DTO(ut).. DT OUT..**

**ViewModel => IN (VALIDATION LOGIC ADDED HERE)**

**Project setup:**

* Web API project – takes care of the **HTTP communication – NO LOGIC**
* Models project – contains our **DTO** and **ViewModel** classes, which are exposed by the service.
* Service project – contains the **business logic**.
* Tests project – contains unit tests. Could be split into more... one for testing services, another for models..
* Entities project – contains our data model – ENTITY CLASSES