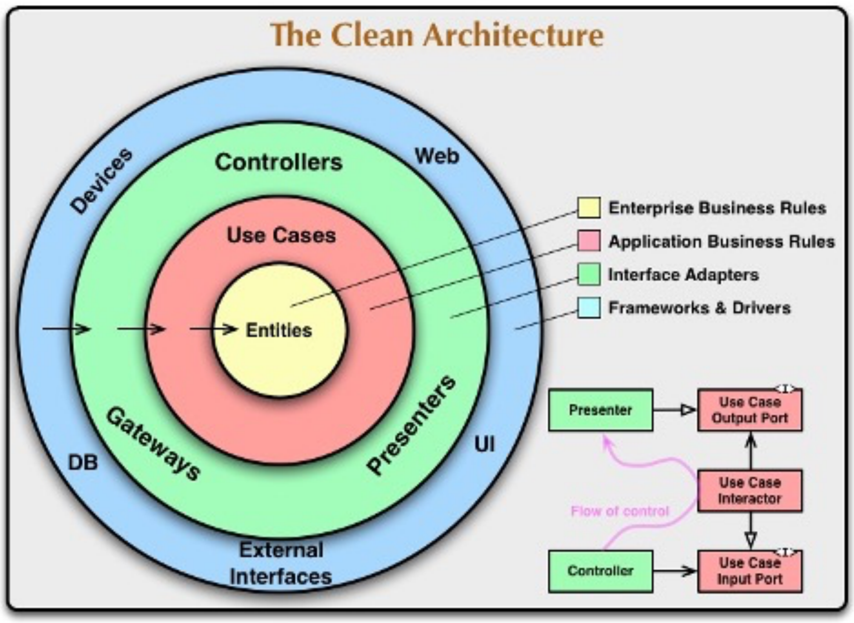
Exercise 4: Clean Architecture

Presentation of results in next lab /
Submission in AULIS until 13.11.25, 9:45am

Assignment 1: Clean Architecture for the Order System

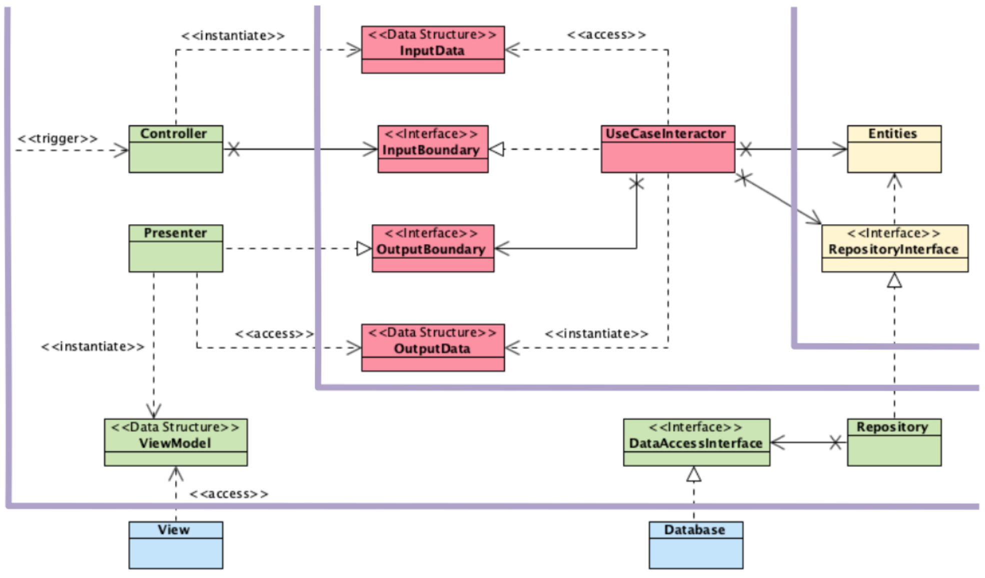
If you look closely at your order system, you will probably find one or the other violation of the *Dependency Inversion Principle*. Does your `OrderService` depend on the repository's implementation (a concrete detail)? Does your user `OrderService` maybe even access the user interface, e.g. in order to ask for parameters of newly created objects or for updating the emptied list of items after finishing an order? Clean architecture to the rescue!



Your task: Refactor your order system such that it conforms to the clean architecture.

Hints and steps:

- Simplification: We will no longer distinguish between *Products* and *Services* any longer. Instead, our *Orders* will only contain *LineItems* with properties for product name, price, and quantity.
- Create a package structure that matches the three inner rings of the clean architecture and move your code to the appropriate packages (and possibly sub-packages). Hints:
    - Your repository represents a gateway, your UIs are combined controllers / presenters.
    - The factories you implemented are part of the entities ring.
- Analyse your dependencies: are there any violations? If so, invert dependencies!
- Adapt your implementation to match the pattern of classes, dependencies, and interactions shown in the figure below (with adequate renaming, of course). Some hints and variations:
    - You don't have to define separate Controller and Presenter classes, i.e. each of your command line interface and your graphical user interface might be combined controller *and* presenter.
    - Define and implement interfaces which represent the input and output boundaries between UI and use cases.
    - Implement <<Data Structure>> classes for exchanging data between UI and use case.
    - Make your repository implement a repository interface defined in the domain ring.
    - To make things a little bit easier, we'll focus on the three inner rings only.
        - The interaction between the repository and the database (blue component with specific database technology) does not have to be regarded.
        - Forget about the ViewModel and View classes in the figure below! The View class would contain, e.g., Swing UI code. We'll leave that directly in the combined Controller / Presenter class.


<<instantiate>>
<<Data Structure>> InputData
<<access>>
<<trigger>>
Controller
<<Interface>> InputBoundary
UseCaseInteractor
Entities
Presenter
<<Interface>> OutputBoundary
<<Interface>> RepositoryInterface
<<access>>
<<Data Structure>> OutputData
<<instantiate>>
<<instantiate>>
<<Data Structure>> ViewModel
<<Interface>> DataAccessInterface
Repository
<<access>>
View
Database

Implement and test your solution! How do you assess the flexibility and overhead introduced by clean architecture?