-->

Milestone 1 (M1) - Clean Code, SOLID-Prinzipien und Aggregate-Konventionen

In this milestone, you will refactor your code from M0, to adhere to the DDD conventions, Clean Code rules and SOLID principles. Compliance with these rules will be automatically tested by additional tests (compared to M0).

There are no new features yet (that will come later). The task description remains the same. Your own packages and classes need to be located below `thkoeln.archilab.ecommerce.solution`, as in M0.

E1 - Aggregates

Aggregates are clusters of related domain objects that are treated as a single unit for data changes. The have one root entity (the aggregate root) controlling all access and consistency rules within the aggregate. Aggregates are the natural candidates for modules (top level packages) in large(r) software system. (See the videos explaining aggregates in the M1 description) on the ArchiLab ST2 page.

In this task, you need to **identify the aggregates** in our shop system, so that you can structure your code base into suitable packages. Below are all the entities that you must group into aggregates.

1. StockLevel
2. User
3. Order
4. ShoppingCart
5. OrderPart
6. FulfillmentCenter
7. ShoppingCartPart
8. Item

Please use the table in `main/resources/E1.md` to specify each aggregate root, and other entities contained in the aggregate (if any). The aggregate root goes into the first column, and the other entities into the second column. The row order, or the order of the entities in the second column, does not matter.

Hint: Your repo contains a hashed solution table, and your specifications are hashed as well and then compared to the solutions. This means that **you can run the `RestTableTests` locally**, and don't have to commit & push every time. However: **What counts in the end, is your report page!** (In case there are problems with this local feature.)

E2 - Migrate your codebase from M0 according to DDD Conventions, Clean Code & SOLID

In this milestone, you need to assess your code base from M0, and migrate it to the conventions followed in this course. Each of the following aspects will be checked by dedicated non-functional automated tests.

E2.1 - DDD building blocks (entities, value objects) implemented properly

There are a number of general do's and don'ts that you should follow when implementing entities and value objects. Amongst the most relevant ones (which we will check) are:

- Value Objects are immutable, and are considered equal if all their attributes are equal
- References between entities are NEVER bidirectional, ALWAYS unidirectional

More details can be found in the ArchiLab Anti-Patterns bei "DDD mit Spring JPA" infopage.

E2.2 - DDD Conventions and proper Aggregate Packages (as in E1)

- Your code follows the **naming conventions** usually used in DDD
- Your code follows the **package and layer conventions** we have defined in this course
- The relevant aggregates are implemented, and located in the correct packages (as specified in E1)

More information and a brief motivation you can find in the ArchiLab DDD-konforme Package-Konvention infopage.

E2.3 - References to other Aggregates only via ID Reference

Aggregates need to be "transaction boundaries" in order to be decoupled of each other. One essential rule is to reference other aggrates only by ID reference, not by object reference. This prevent the temptation to modify other aggregates in the same transaction. It also helps keeping queries small. See the ArchiLab infopage on the 4 main aggregate design rules for a more detailed motivation.

The IDs should be typed. You need to derive them from the abstract `thkoeln.archilab.ecommerce.GenericId` class. In addition, you need to implement a converter (which ensures the proper DB mapping), derived from `thkoeln.archilab.ecommerce.GenericIdConverer`.

The ArchiLab infopage on aggregate references via typed IDs explains in detail how you can do that.

E2.4 - Clean Code and SOLID Principles

When you refactor and enhance your code from M0, you need to apply the Clean Code rules and the SOLID principles. We will check the following aspects:

- Clean Code
    - Meaningful names for variables, classes, packages, ...
    - Methods should be small, not longer than max. 30 lines of code, and if possible should be much smaller.
    - Code lines should not extend 120 characters - add a linebreak otherwise, or try re-writing the code.
- SOLID
    - Single Responsibility Principle
        - Classes should serve one purpose only
        - No domain business logic in application services or controllers
    - Open-Closed Principle
        - No public access to member variables
        - States (see above for ShoppingCart) must be encapsulated - no possibility of directly changing the status from outside

This will help you:

- videos to Clean Code and SOLID (see M1 description in the ArchiLab ST2 page)
- Checkliste für Clean Code und SOLID-Prinzipien

E2.5 - Resolving cyclic dependencies using Dependency Inversion Principle

You code must not have any cyclic dependencies. You can avoid them by applying the Dependency Inversion Principle (DIP). See the

This will help you:

- the ArchiLab video on DIP
- Dedicated info page Zykel auflösen mittels Dependency Inversion Principle