Repetition Questions (dt. Wiederholungsfragen) Lesson 6

Topics and Concepts (Link to Lecture, via Fact Sheets)

The lesson of the lecture today covered the following concepts (see fact sheets in script folder):

1. Tactic DDD
2. Architecturally Evident Coding Style (AECS)

In the corresponding exercise[1], we worked with these concepts. In the self-study assignment handed out this week, you have the opportunity to practice the concepts one more time by yourself.

Questions

Topic/Concept: Tactic DDD

1. How does DDD relate to OOAD and PoEAA?
2. Which four layers does the sample application introduce/deal with?
3. What is the difference between an Entity and a Value Object in DDD?
4. What is the purpose of an Aggregate?
5. Should a Repository return entities that are not root entities in any Aggregate?
6. Name at least two more DDD patterns not mentioned in questions 3, 4, 5.

Topic/Concept: Architecturally Evident Coding Style (AECS)

1. What is the motivation for the AECS concept (for instance, in terms of software quality attributes)?
2. Do all DDD patterns promote an AECS? If yes, explain why/how. If no, provide a counter example. Name at least one DDD pattern whose name strongly suggests that it does pronote an AECS.
3. Which class name is better from an AECS point of view: `ClientManager` vs. `RetailBankCustomerAggregate`?
4. Which grammar constructs and naming conventions would you recommend for a) components classes, b) operations of a DDD entity and c) services and their operations?

Answers

Topic/Concept: Tactic DDD

1. *The Domain-Driven Design (DDD) patterns refine the Domain Model pattern from PoEAA (the DDD pattern Service can also be seen as a DDD version of Transaction Script in PoEAA). A design- and implementation-level domain model that populates the Business Logic Layer (BLL) of a layered application architecture is one of the major outputs and artifacts of OOAD (which is a family of methods, or analysis and design techniques).*
2. *Interface (serving human users over the Web and other applications via events), Application, Domain, Utility (which includes data access).*
3. *Entities have a unique identifier and can change their state during their lifetime; value objects do not have an identity and have to be copied to obtain new values (they are immutable). Both can carry data and behavior (methods). The methods of value objects should be free of side effects.*

[1] ../3-exercises-solutions/ZIO-AppArch-ExerciseWeek6.pdf

4. *Aggregates cluster or group related entities and value objects, and have a single root entity as entry point. They are responsible for checking and preserving invariants (such as certain business rules) that cannot be assigned to a single entity without breaking the conceptual integrity of that entity (in terms of cohesion). System transaction boundaries are typically set at the aggregate boundary and can help preserving conceptual integrity and enforce invariants.*
5. *A local repository only used within the same aggregate might do that. But usually repositories are used by components in the interface layer and the application layer (or entities in other aggregates), and these should only work with the root entities of aggregates. The original DDD book by E. Evans explicitly says that there should only be one repository per aggregate (which usually works with the root entity).*
6. *See lecture slides 11, 12 and 26: Ubiquitous Language, Intention Revealing Interface, Module, Core Domain, Generic Subdomain*

Topic/Concept: Architecturally Evident Coding Styles (AECS)

1. *See FAQ in exercise instructions (and sample solution): When the system and its architecture have to be extended or modified otherwise (during maintenance and evolution), the architecture might have to be changed. External documentation might not be up-to-date; and even if it is up-to-date, an extra step is required to look it up. Hence, the code can help reviewers and maintainers to orient themselves and avoid mistakes when changing the code and, indirectly, the architecture.*
2. *Yes, if applied in the spirit of the pattern and if the implementation artifacts are named properly: The role and responsibilities of entities, value objects, etc. in the architecture are defined in the pattern solutions. Intention Revealing Interface is one pattern that talks about naming conventions explicitly.*
3. `RetailBankCustomerAggregate` *as it unveils domain-specific information as well as pattern/architectural role.*
4. *a) components: nouns indicating role and responsibilities, b) operations: verbs describing the implemented logic and its outcome, c) noun derived from a verb for class (xxxService, with xxx being an activity), simple "invoke" or "execute" for its single method.*