Latihan UTS

Pattern Software Design

Nama : Steven Chowina

NIM : 2702295373

Kelas : LE01

Essay

Explain what these Tactical Patterns are and provide some examples for each. You may provide illustrations to help you explain the answers:

a. Domain Model Pattern -> This tactical pattern focuses on representing the core concepts of a system within objects (Object-Oriented), where these objects encapsulates both data and the logic related to that data. The primary idea is that the domain model is a conceptual map of the problem space and business logic.There are four component in the Domain Model Pattern :

1. **Entities**: Objects with a unique identity that continues through different states and lifecycles. So in here we can take example in e-commerce there are entity Order that have an unique ID.
2. **Value Objects**: Objects that don't have identity. They are defined only by their values/attributes. So in the case of e-commerce too we can say that Address in Entity Order are VO because address don't have id and always be with the entities.
3. **Domain Services**: Function (feature in app) that runs the business logic. In this case we can say that there are function to calculate a Shipping Cost in E-Commerce app. The Function Shipping Cost Calculator is a domain services because its need entities Product and the value object Weight to calculate the cost.
4. **Modules**: Logical groupings of related functionalities/features. They help to organize the code and make the system easier to understand. For example in the e-commerce system, there are Customer Module that includes Customer, Address and the Payment Method.

b. Lifecycle Pattern -> This tactical pattern is used for managing various stages or states of an object or component during its existence. There are three components in Lifecycle Pattern:

1. **Aggregate**: A group of related entites and value objects that are treated as a single unit. It enforcs consistency rules within its boundaru and defines the root entity that is the only point of interaction from the outside world. We can group it using Invariants (Business Rules), for example in E-Commerce there are "Discount for customer in Jabodetabek" so object Customer, Postcode and Address can be grouped into one Aggregate for that Business Rule.

2. **Factory**: Factory is used to create complex aggregrates or objects. Factory also hides the complexity of object creation and ensures that the objects created are valid respecting all necessart constraints. For example if we want to create a new Product object so ProductFactory would be responsible to initialize it.

3. **Repository:** Repository acts as a collection of aggregates (like a database table). It abstracts the persistence layer (like a database) and handles persistence and retrieval of aggregates. For example there are OrderRepository which handle saving and fetching Order aggregates. Instead of

interacting with the databae directly, we can just interact with the repository which handles all the persistence logic.

c. **Emerging Pattern** -> This tactical pattern is about evolving practices or techniques that help handle complex domain logic in a more efficient, scalable and maintainable way. There are two key concepts of emerging pattern:

1. **Domain Events**: represents a significant in the state of an entity or aggregate that the system needs to track. So domain events give a signal/flag for every event that has occurred in the domain that the other parts of the system may be interested in. For example in e-commerce, when a customer places an order that's a domain event. This place order event could be published to notify other part of system like shipping service.

2. **Event Sourcing**: Event sourcing store all changes (signal,flag,snapshot) to an entity or aggregate as a sequence of events rather than just storing the current state of entity. Event sourcing are very useful when we want to track or debugging error, rebuild the state of an object from its history. So for example if we want to fix the error in our system we can just see our audit log that already have store all the audit in our system.

Case Study

Application Theme: Coffee Shop Cashier System

You are asked to create a simple design of an application system to meet the following requirements. Read the requirements outlined below carefully and answer the questions beneath them.

The application facilitates individual accounts for each employee. Employees are categorized into distinct roles including Cashier, Cashier Manager, and Warehouse Manager, each with specific permissions and responsibilities. The application systematically records transaction details for each purchase, including crucial information. such as Transaction ID, date, purchased products (coffee, tea, and snacks), the assigned cashier, and the total transaction amount. Moreover, the application manages the inventory of raw materials stored in the warehouse, ensuring automatic updates with each transaction to maintain accurate stock levels. Additionally, the application offers comprehensive reporting capabilities, enabling users to generate detailed transaction reports categorized by daily, weekly, or monthly periods, facilitating informed decision-making and analysis.

Design the system by answering the following questions!

1. Determine Core Domains, Supporting Subdomains, and Generic Subdomains for this

system. Explain why you chose these domains.

➔ **1. Core Domains ->** represent the main focus of the application, which is the most critical functionality that differentiates the system and are the central of the business. For the case of Coffee Shop Cashier System, the core domains are **Transaction Management:** This domain manages the purchase transactions that are the core of the business. It also involves keeping track of transaction details too.

2. Supporting Subdomains -> provide additional functionality but not hold the same core value as the core domains, they only support the core domains but are not central to the business. For the case of Coffee Shop Cashier System, the supporting subdomains are **Inventory Management:** This domain deals with the management of raw materials in the warehouse to ensures accurate stock levels and automatically updates it after transaction that's also the core of the business.

3. Generic Subdomains -> represent functionality that is generic but is needed in many systems. For the case of Coffee Shop Cashier System, the generic subdomains are **Employee Management:** This subdomain manages employee, roles and the permissions and **Report Management:** This domain focuses on generating detailed transaction reports which help in decision making and analysis which is also the requirement of the system.

2. Find and write down the Bounded Contexts you've created based on the requirements

above! Provide a brief essay after listing all bounded contexts to explain why you divided

this system into your chosen contexts.

➔ **Bounded Contexts:**

1. **Transaction Context**
2. **Inventory Context**
3. **Employee Context**
4. **Report Context**

Explanations:

We separate these contexts to ensure that each area of the system is modular, focused, and easier to maintain. Here are the brief explanations for each context:

1. **Transaction:** Focuses solely on handling transaction data and the details for each purchase. It is separated because the logic of processing purchases, calculate it and even updating the transaction records is the core of this business.
2. **Inventory:** Manages all aspects related to inventory tracking, stock level management, and updating after every purchase. It is important to keep this separate because this is a distinct business area from transactions.
3. **Employee:** Handles user accounts and roles. This context is responsible for handling access and permissions for employees, which should not be mixed with the logic of transactions or inventory.
4. **Report:** Is specialized in generating and presenting reports based on data from the transaction context. This is a distinct context because focuses on data aggregation and presentation, which is separate from the transactional or inventory domain logic.

3. Determine the bounded context integration for this case.

-> To determine the bounded context integration for this case we need to consider the interactions between the different bounded contexts and how they communicate with each other. Since there are multiple domains in the system, we can use various integration techniques based on the relationships between the bounded contexts, such as Event-Driven and Remote Procedure Call.

a. Transaction Context -> Inventory Context and Transaction Context -> Report Context

For these contexts relationship we can use Event-Driven via Message Bus. Whenever the transaction done, the transaction context will send a Transaction Completed Event. Then Inventory Context will catch the event and will update the stock on the inventory. It also the same for the report context, it will save the transaction data for report.

b. Employee Context -> Transaction Context and Employee Context -> Inventory Context

For this contexts relationship we can use Remote Procedure Call. Because for example if a customer do a transaction then the Employee context will validate that the role is only cashier that can service the customer transaction. It's the same for the Inventory when the Inventory Manager want to update the inventory manually then the employee context will validate the role too.

4. Create a model for value objects used in this case. Explain your reasons!

-> **Value Objects** are immutable and are defined only by their values. These are typically small objects that are used to describe the properties of an entity. For this case, there are some value objects like:

a. Employee Role

b. Total Amount

c. Transaction Date

d. Stock Level

e. Report Date

f. Money

5. Create a model for entities using the value objects provided in this case. Explain your

reasons!

-> **Entities** have unique identifiers and are mutable, meaning they can change over time. For this case there are some entities like:

a. Employee

- Identity: Employee ID (like EM001)

- Value Objects: Employee name and role (like cashier, inventory manager)

b. Transaction

- Identity: Transaction ID (like TR001)

- Value Objects: Transaction Date, Total Amount

c. Inventory

- Identity: Inventory ID (like IN001)

- Value Objects: Stock Level

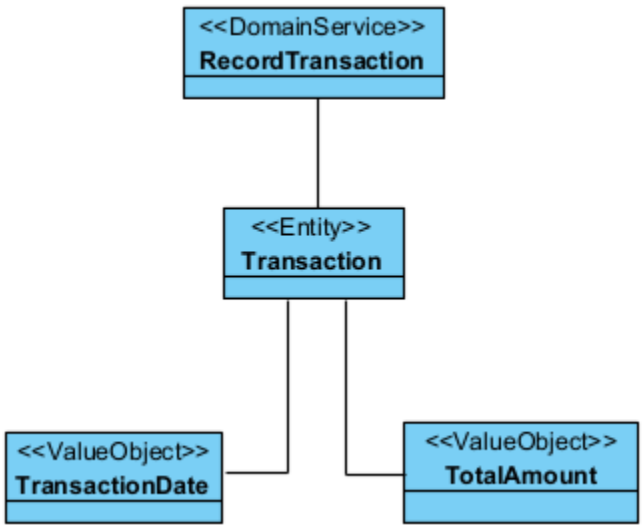
d. Report

- Identity: Report ID (like RE001)

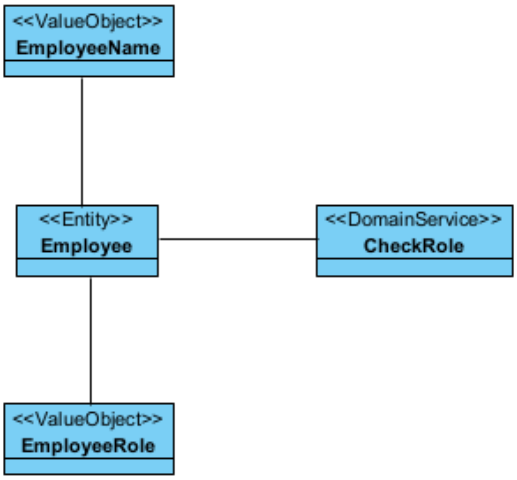
- Value Objects: Report Date

6. Create a Class Diagram for each bounded context as a design of the context model (example: if you create 4 contexts, then create 4 class diagrams, 1 for each context). Mark each class to distinguish whether the class is an Entity, Value Object, or Domain Services.

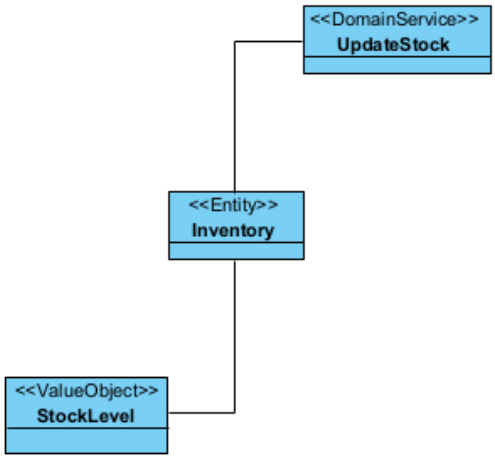
1. Transaction Context



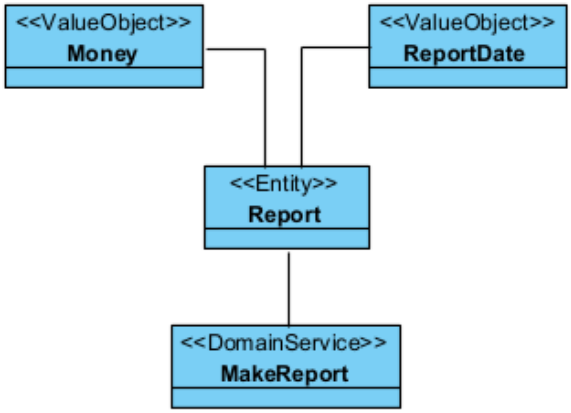
2. Employee Context



3. Inventory Context



4. Report Context



Additional Information:

1. Class Diagrams do not need to show attributes and methods. Only the class names and relationships between classes (association, generalization, aggregation, composition) are required.
2. Requirements are the features that must be included in your answers. You may add additional features or details according to your own opinion, as long as all the requirements above are met in your design.