

**Data Mapper**

**Design Pattern**

19K-0294 Shahriyar Ahmed

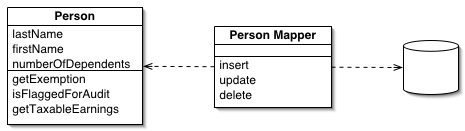
19K-1375 Shayan Mallick

19K-0197 Abdullah Bakht

**Definition:**

The Data Mapper design pattern is a software architectural pattern that separates the responsibility of data persistence and retrieval from the domain objects in an application. It provides a way to map data between different representations, such as a relational database and an object-oriented domain model.

**Structure:**



**Intent:**

To decouple the domain objects from the database or data storage layer. The data mapper pattern ensures that the domain objects are not directly aware of how they are stored or retrieved from the underlying data source. This separation helps to improve the maintainability and flexibility of the application.

By using the Data Mapper pattern, changes to the persistence layer or the domain objects can be made independently without affecting each other. The domain objects remain focused on representing the business logic, while the Data Mapper handles the storage and retrieval of data.

**Motivation:**

* Separation of concerns: The pattern separates the domain logic from the persistence logic, promoting a clear separation of concerns and improving the overall maintainability of the codebase.
* Flexibility: It allows you to change the data source or the domain object structure independently, as long as you update the corresponding mappings in the Data Mapper.
* Testability: Since domain objects are not tightly coupled with the data source, it becomes easier to write unit tests for the business logic without the need for a real database connection.

**Example:**

Let's consider a scenario where we have a web application for an online store. The application needs to handle the persistence and retrieval of Product objects from a database. This is where the Data Mapper pattern comes into play.

1. Domain Object: We have a Product class representing a product in our domain model:
2. Data Mapper: The ProductMapper class acts as a mediator between Product objects and the database:
3. Mapping: The ProductMapper handles the mapping between Product objects and the database. It knows how to transform the Product object's properties (e.g., name, price) into the appropriate format for storage in the database.
4. Persistence Layer: The persistence layer represents the database. The ProductMapper encapsulates the details of interacting with the database, such as executing SQL queries or using an ORM (Object-Relational Mapping) framework.

By using the Data Mapper pattern, the application can ensure that the Product class remains focused on representing the product's properties and behavior, while the ProductMapper handles the storage and retrieval of Product objects from the database.

For example, when a new product is added to the online store, the ProductMapper will be responsible for saving the Product object to the database. Similarly, when retrieving a product by its ID, the ProductMapper will handle querying the database and mapping the retrieved data back into a Product object.

This separation of concerns provided by the Data Mapper pattern allows for better maintainability, testability, and flexibility in the application's architecture.