

JPA, DAO, Persistence Layer

Martin Ledvinka

`martin.ledvinka@fel.cvut.cz`

Winter Term 2025



Contents

- 1 Persistence Layer
- 2 Data Access Objects
- 3 Tasks



Persistence Layer



Persistence Layer

- In a layered architecture, persistent data are handled by the *persistence layer*
- Data storage itself is typically handled by a database management system (DBMS), the persistence layer provides access to it
- Separates business logic from database access
 - Isolation and encapsulation of data access code
 - Possible to develop components separately
 - Clearly defined layer API



Persistence Layer in Layered System

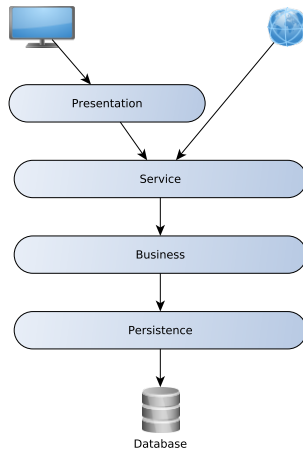


Figure: Layered architecture visualization.



Data Access Objects



Data Access Object (DAO)

- Persistence layer usually consists of *data access objects* (DAOs)

DAO

- Encapsulates access to data via a particular provider
- Provides clear API
- Create, Retrieve, Update, Delete (CRUD) operations + complex operations and queries
- No business logic!
- Entity class $\neq$ dedicated DAO



DAO vs JPA

- `EntityManager` is essentially a generic DAO
- But specific to JPA (relational databases)
- Wrapping JPA/Hibernate in application-specific DAO allows hiding the implementation details of its usage



DAO vs Repository

- *Repository* pattern
- Alternative to Data Access Object
- Repository $\sim$ a collection of objects of one type
- Origin in *domain-driven design*



Tasks



Syncing Your Fork

- ❶ Ensure you have no uncommitted changes in the working tree
 - `git status` → your branch is up to date, nothing to commit
- ❷ Fetch branches and commits from the upstream repository (EAR/B251-eshop)
 - `git fetch upstream`
- ❸ Check out the task branch from **upstream** (one line!)
 - `git checkout -b b251-seminar-04-task upstream/b251-seminar-04-task`
- ❹ Do the task
- ❺ Commit and push the solution to **your** fork
 - `git push -u origin b251-seminar-04-task`



Building a Persistence Layer

- We have a `GenericDao` interface defined in e-shop
- It defines basic CRUD operations
- Let's start implementing it
 - `CartDao`
 - `ItemDao`
 - `CategoryDao`
 - `OrderDao`
 - `ProductDao`
- Does the code repeat? Then let's extract it into a common superclass!



Task: 1 Point

There is one more DAO missing: `UserDao`. Besides basic CRUD operations, it will eventually have to provide a method for getting user based on their username (for login).

Also, we are still missing update and delete methods in our DAOs.

- 1 Create class `UserDao` and implement method `findByUsername` which retrieves user given the specified username.
 - *Hint: use a named query.*
 - *Hint: do not forget to annotate the DAO with `@Repository` (we will find out more about it next time).*
- 2 Add the missing implementation of update and removal into our DAOs.

Acceptance criteria: Project is buildable by Maven (`package`) – tests pass.



The End



The End

Thank You



Resources

- https://cw.fel.cvut.cz/b251/_media/courses/b6b36ear/lectures/lecture-02-architectures-s.pdf
- https://cw.fel.cvut.cz/b251/_media/courses/b6b36ear/lectures/lecture-03-jpa-s.pdf
- <https://www.baeldung.com/java-dao-pattern>
- Evans E.: Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley Professional, 2003

