Core JEE patterns - Integration tier design patterns

Data Access Object (DAO)

Purpose:

- Have a layer for persistence separated from the rest.
- Want to provide an uniform API irrespective of the underlying storage.

Solution

- All access to the data store only through DAOs.
- DAOs manage the connections to the data source.
- Client creates DAO object. DAO object access data source and create Transfer objects will be returned to the client.
- Factory method pattern could be employed to provide different DAO objects. Cache could be employed here.
- If multiple data sources have to supported, use abstract factory pattern along with the factory method.

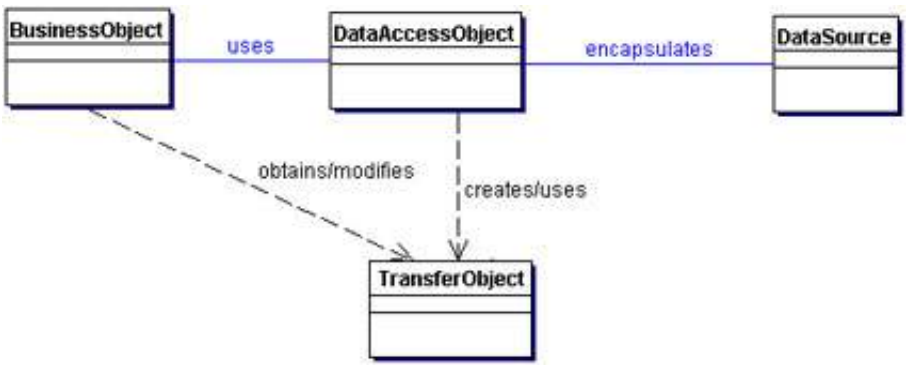
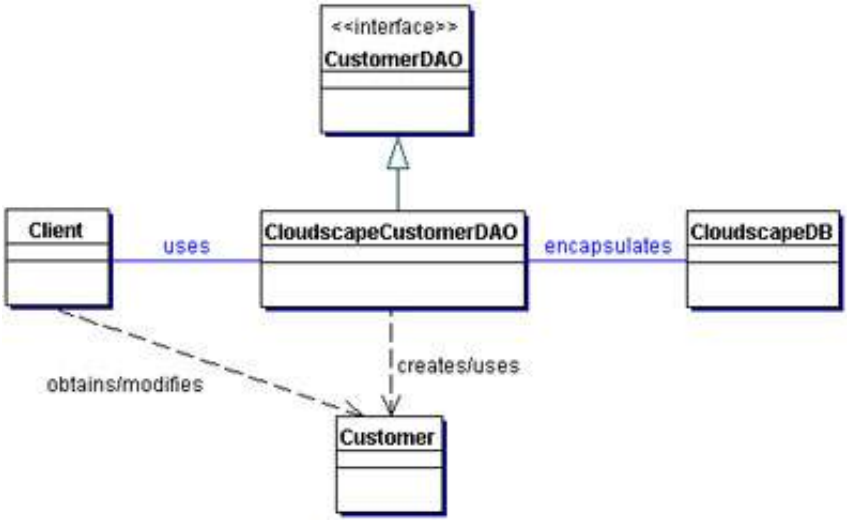
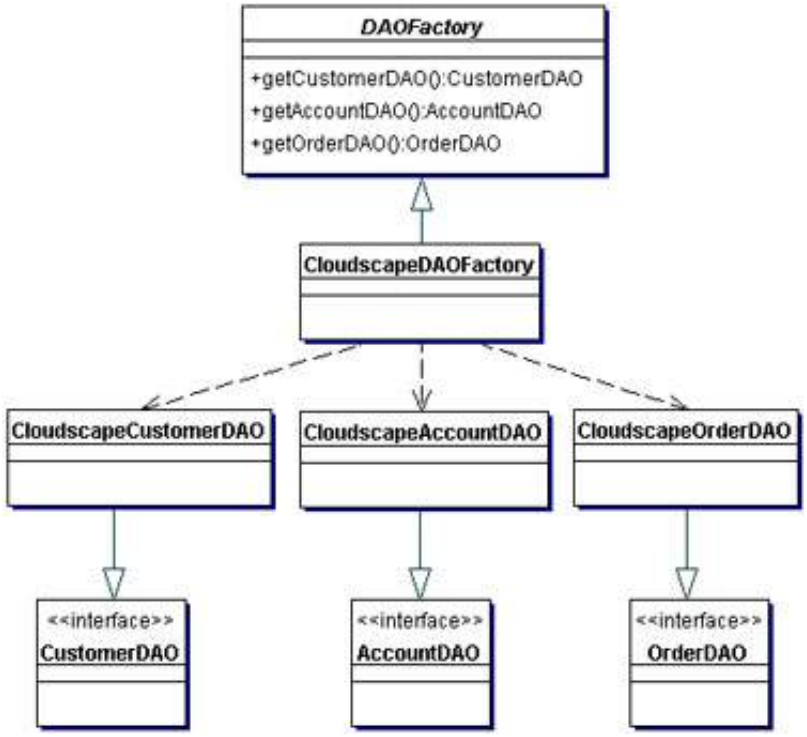


Figure -- Basic DAO implementation

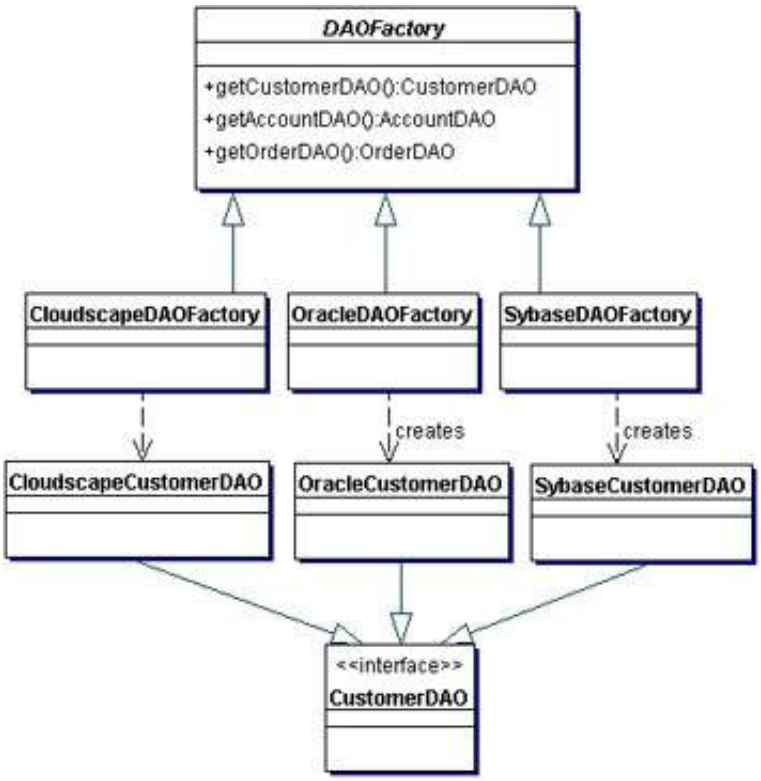
An example for above



Factory method implementation



Abstract factory implementation



Domain store

Problem

- Mechanism to persist business object through a separate persistence

Solution

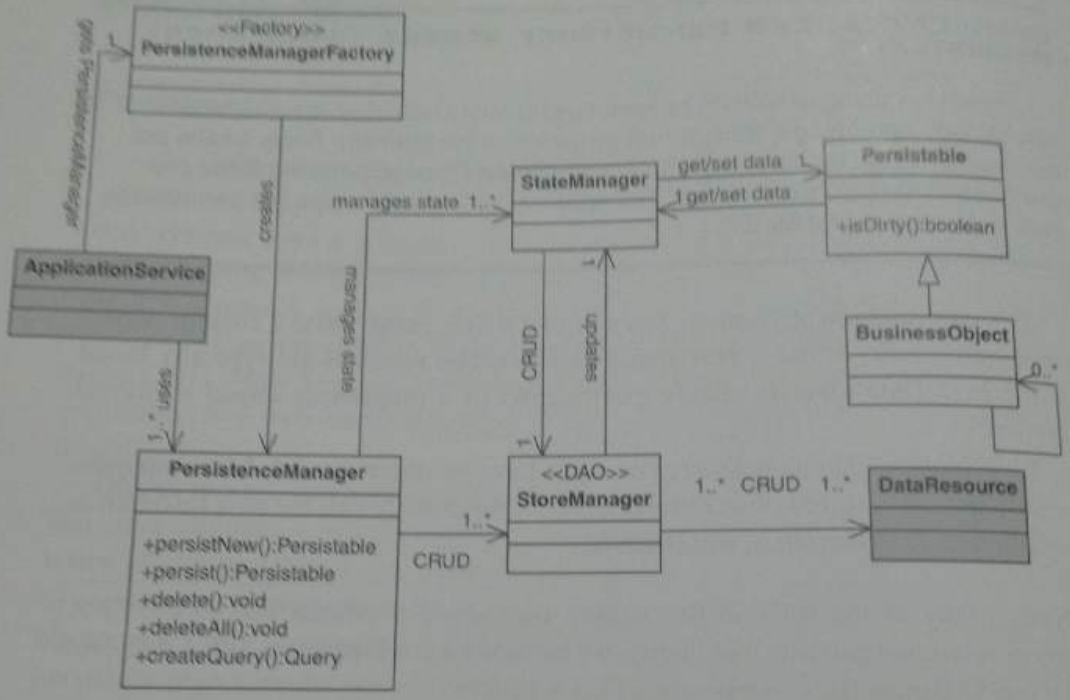


Figure 8.16 Domain Store Class Diagram

Figure 8.17 shows the *Domain Store* class diagram with the added participating classes.

Note:

- Ways to persist business object.
    - Business object uses query to fetch data from database.
    - Business object could use entity bean.
    - Code persistence into the business object itself (active record)
    - Separate persistence from business object (this pattern)

Service activator

Problem

- Mechanism to persist business object through a separate persistence

Solution

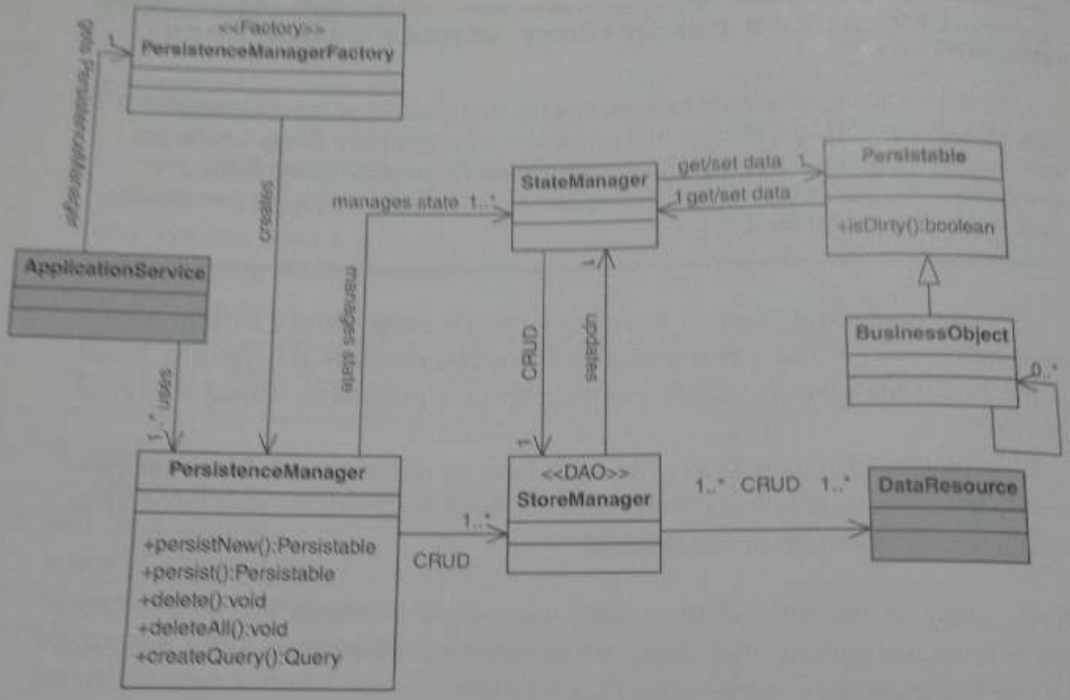


Figure 8.16 Domain Store Class Diagram

Figure 8.17 shows the *Domain Store* class diagram with the added participating classes.

Note:

- Ways to persist business object.
    - Business object uses query to fetch data from database.
    - Business object could use entity bean.
    - Code persistence into the business object itself (active record)
    - Separate persistence from business object (this pattern)