CRUD USING VARIOUS DESGN PATTERNS

DEVENDRA LAD

I have completed Mitchell Coding challenge and here's the description.

Technologies used

1. Java
2. Jersey 2 framework for RESTful web services
3. Junit testing framework for java for test driven development
4. Java Collection (Hash Map) is used for persistence storage

Tasks Completed

1. CRUD operations on Vehicle Entity as per given Interface using **S.O.L.I.D.**
2. Unit tests using Junit
3. Rest services using Jersey framework
4. Validation for the service like Null and Max-Min constraints

Approach for Design

1. Since our focus here was elegance and extensibility I have tried using very flexible, scalable design, maintainable with a readable code base and welcoming to the changing requirements.
2. The project is segregated into 5 packages
    a. Controller – Takes care of client requests and their responses
    b. Service – Provides Controller with the functionalities
    c. Model – Contains Entities in project (Here we just have Vehicle)
    d. DAO – Has all the classes related to Data and its management
    e. Test – Has test cases written for CRUD operations
3. I have used controller/Services pattern where each controller will take care of some kind of functionality and talk to services for it, therefore not knowing underneath implementations.
4. If we add more entities later on, there will be different Service interfaces

for these so this implements *Interface separation.*

5. I have used *dependency Inversion* principle which makes design very flexible. Here High level module which is CRUD doesn't depend on Low level module like InMemory, SQL or any other DB for that matter. We can just add one more class and good to go without making substantial changes In already existing code.

```
@Override
public void contextInitialized(ServletContextEvent arg0) {

    //context Root-> initialize everything here.
    //dependency injection, we can inject DAOFactory.SQL to perform SQL crud
    DAOFactory inMemoryDAOFactory = DAOFactory.getDAOFactory(DAOFactory.INMEMORY)
    //DAOFactory inSQLDAOFactory = DAOFactory.getDAOFactory(DAOFactory.SQL);

    IVehicleService vehicleDAO = inMemoryDAOFactory.getVehicleDAO();
    //IEmployeeService employeeDAO = inSQLDAOFactory.getEmployeeDAO();

    //context for the configured IVehicleService is passed to the controller
    //to use and work on, similarly we can do for employeeService
    VehicleController.vehicleService = vehicleDAO;
    //EmployeeController.employeeService = employeeDAO;

}
```

In above code snippet, we can see that Once we get vehicleDAO we pass it to our service, It implements IVehicleService which is our CRUD interface. So Service doesn't really care if the vehicleDAO is coming from inMemoryDAOFactory or inSQLDAOFactory. Similarly, we can do this for other entities as well and we could also use different data sources for different entities if application demands.
So both Low level and High level modules depend on Abstraction than being tightly coupled together.

References

1. DAO pattern <- http://www.oracle.com/technetwork/java/dataaccessobject-138824.html
2. S.O.L.I.D. <- http://www.c-sharpcorner.com/uploadfile/damubetha/solid-principles-in-c-sharp/

3. Dependency injection <- http://www.c-sharpcorner.com/UploadFile/3d39b4/constructor-dependency-injection-pattern-implementation-in-c/
4. Jersey <- https://jersey.java.net/documentation/latest/user-guide.html