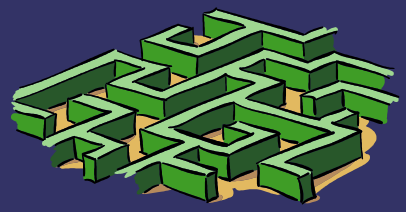
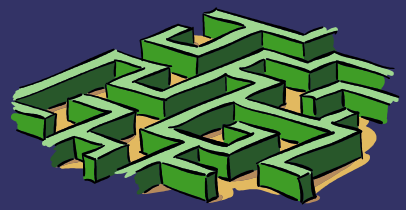
Spring JDBC Template

JUG Ethiopia/Wurfl-PRO



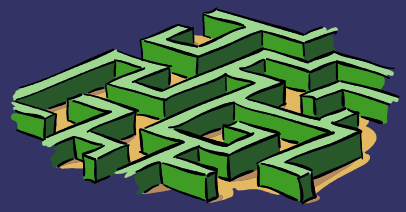
Overview

- Introduction
- Before DAO
- Traditional DAO
- Spring JDBC



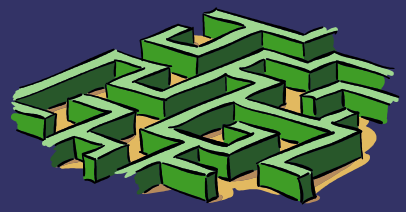
Introduction

Anche con l'avvento di tools orm per la persistenza, l'uso diretto di api jdbc è ancora la strada intrapresa da molti programmatori.



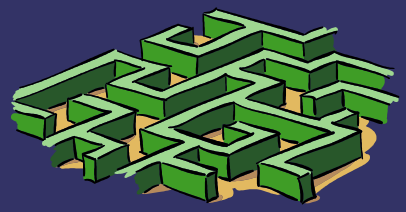
Before DAO

- Spesso e volentieri ci si ritrova ad avere codice relativo alla persistenza all'interno della logica di business
- Questa va contro ogni logica di buona programmazione object-oriented
- Esiste una soluzione?

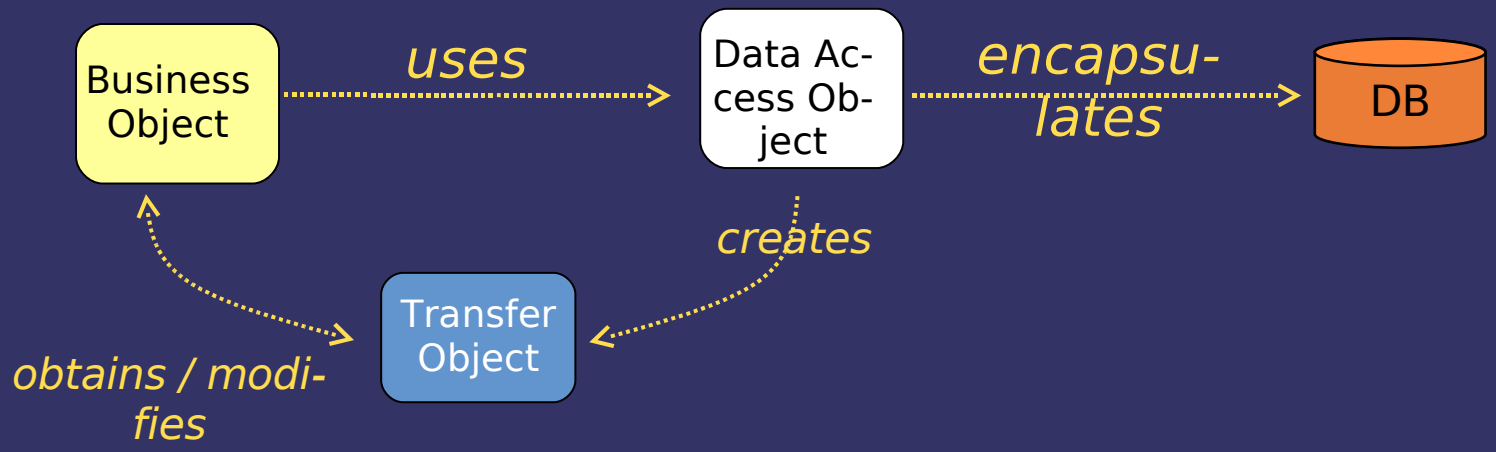


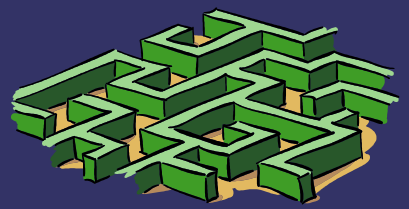
DAO design pattern

- Integration tier design pattern
- introduce astrazione tra lo strato di business e la persistenza incapsulando l'accesso ai dati e il codice di mani-polazione in un oggetto apposito
- l'interazione tra i due strati avviene tramite interfacce ben definite (programming against interface)



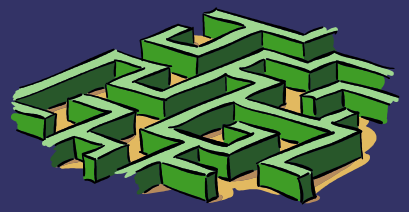
DAO design pattern(2)





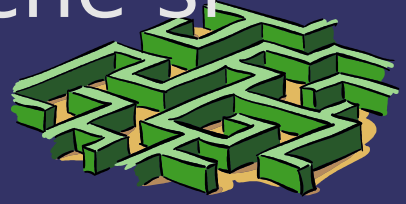
DAO(3)

- Incapsulano tutte le interazioni con le JDBC api
- incapsulano il dato restituito in un transfer-object api-neutro da passare al business layer per ulteriori trat-tamenti
- Gestione delle eccezioni
- Gestione delle risorse (connection, statement)



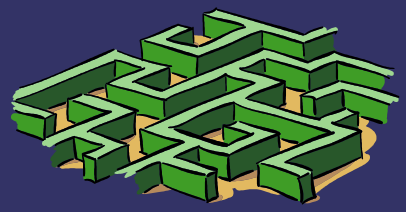
DAO Advantages

- flessibilita : idealmente una modifica della sorgente dei dati (cambio del vendor o tipo di dbms) richiederebbe solamente modifica nel DAO con un impatto minore
- Testing: facilita di testare i servizi in quanto separati
- Possibilita di lavorare in parallelo su uno stesso progetto una volta che si sono definite le interfacce



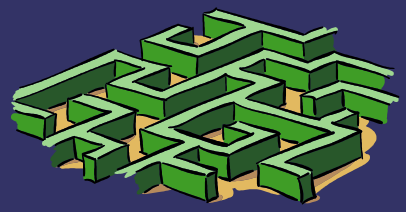
DAO disadvantages

- Ripetizione di codice relativi a connessione, statement, gestione eccezioni ecc...
- I diversi vendor identificano errori dello stesso tipo con codici diversi: ciò costituisce un problema per la portabilità
- Bisogna ricordarsi di rilasciare le risorse...



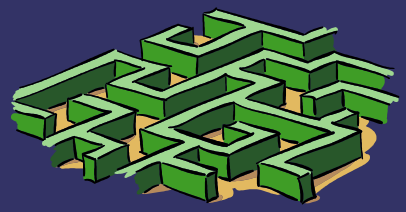
Template/callback

- Secondo i GoF
  - “Define the skeleton of an algorithm in an operation, deferring some steps to sub-classes. Redefine the steps in an algorithm without changing the algorithm's structure”.
    - Ci aiuta quindi ad identificare la base comune fra l'insieme delle procedure differenti incapsulando le differenze nelle classi derivate.



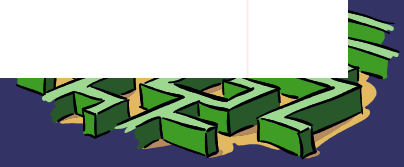
Spring JDBC Template

- è una forma di inversione di controllo in quanto sposta il controllo nel framework
  - acquisizione e rilascio di risorse
  - data access exception
  - Interfacce semplici per l'accesso e mani-polazione dei dati nel db
  - callback methods per l'implementazione di parte del workflow di jdbc



Esempi

```
 *
 * @author fanta
 */
public class JdbcPersonDAO extends JdbcDaoSupport implements IPersonDAO
{

    private static final String FIND_SQL = "select * from persons where id = ?";
    public Person find(Integer id)
    {
        return (Person)getJdbcTemplate()
            .queryForObject(FIND_SQL, new Object[]{id}, new PersonMapper());
    }

    private static final String FIND_ALL_SQL = "select * from persons";
    public List findAll()
    {
        return getJdbcTemplate().queryForList(FIND_ALL_SQL);
    }
```

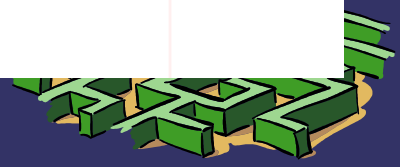


Callback

```
// callback
private static class PersonMapper implements RowMapper
{
    public Object mapRow(ResultSet resultSet, int i) throws SQLException
    {
        Person person = new Person();
        person.setId(resultSet.getInt("id"));
        person.setUsername(resultSet.getString("usernane"));
        person.setPassword(resultSet.getString("password"));
        person.setFirstname(resultSet.getString("firstname"));
        person.setLastname(resultSet.getString("lastname"));

        return person;
    }

}
```

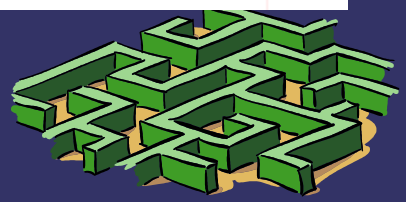
MappingSqlQuery

```
public class PersonQuery extends MappingSqlQuery
{
    private static final String FIND_SQL = "select * from persons where id = ?";
    public PersonQuery(DataSource dataSource)
    {
        super(dataSource, FIND_SQL);
        declareParameter(new SqlParameter("id", Types.INTEGER));
        compile();
    }
    public Person find(Integer id)
    {
        return (Person) findObject(new Object[] {id});

    }

    public Object mapRow(ResultSet resultSet, int i) throws SQLException
    {
        Person person = new Person();
        person.setId(resultSet.getInt("id"));
        person.setUsername(resultSet.getString("usernane"));
        person.setPassword(resultSet.getString("password"));
        person.setFirstname(resultSet.getString("firstname"));
        person.setLastname(resultSet.getString("lastname"));

        return person;
    }
}
```



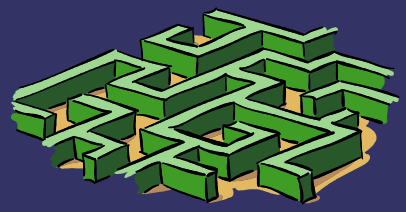
update

```
public static final String UPDATE_PERSON_SQL =
        "update persons set username = ?, password = ?, firstname = ?, lastname
        "where id = ?";
public void updatePerson(Person person)
{
    getJdbcTemplate().update(UPDATE_PERSON_SQL,
            new Object[] {person.getUsername(),
                          person.getPassword(),
                          person.getFirstname(),
                          person.getLastname(),
                          person.getId()});
}
```



Exception Handling

```
public class MySqlErrorTranslator extends SQLErrorCodeSQLExceptionTranslator
{
    protected DataAccessException customTranslate(
            String task, String sql, SQLException sqlEx)
    {
        if(sqlEx.getErrorCode() == -12345)
            return new DeadlockLoserDataAccessException(task, sqlEx);
        return null;
    }

}
```

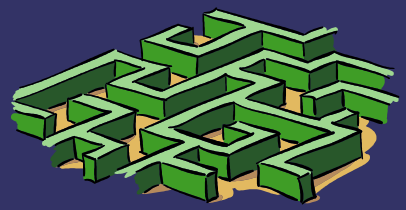


un po di configurazione applicationContext-dao.xml

```xml
<beans>

    <bean id="propertyConfigurer"
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations"
            value="classpath:net/fantayeneh/springday/dao/jdbc.properties" />
    </bean>

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
          destroy-method="close">
        <property name="driverClassName" value="${driver}"/>
        <property name="url" value="${url}"/>
        <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
    </bean>

    <bean id="personDAO" class="net.fantayeneh.springday.dao.JdbcPersonDAO">
        <property name="dataSource" ref="dataSource" />
    </bean>
</beans>
```

applicationContext-service.xml

```
<beans>
    <bean id="personService"
        class="net.fantayeneh.springday.service.PersonService">
        <property name="personDAO" ref="personDAO" />
    </bean>

</beans>
```



Conclusione

- Abbiamo visto i pro e contro del design pattern DAO
- Abbiamo visto come Spring risolve quelli che sono i maggiori problemi del DAO attraverso
  - template
  - callback
  - data access exception