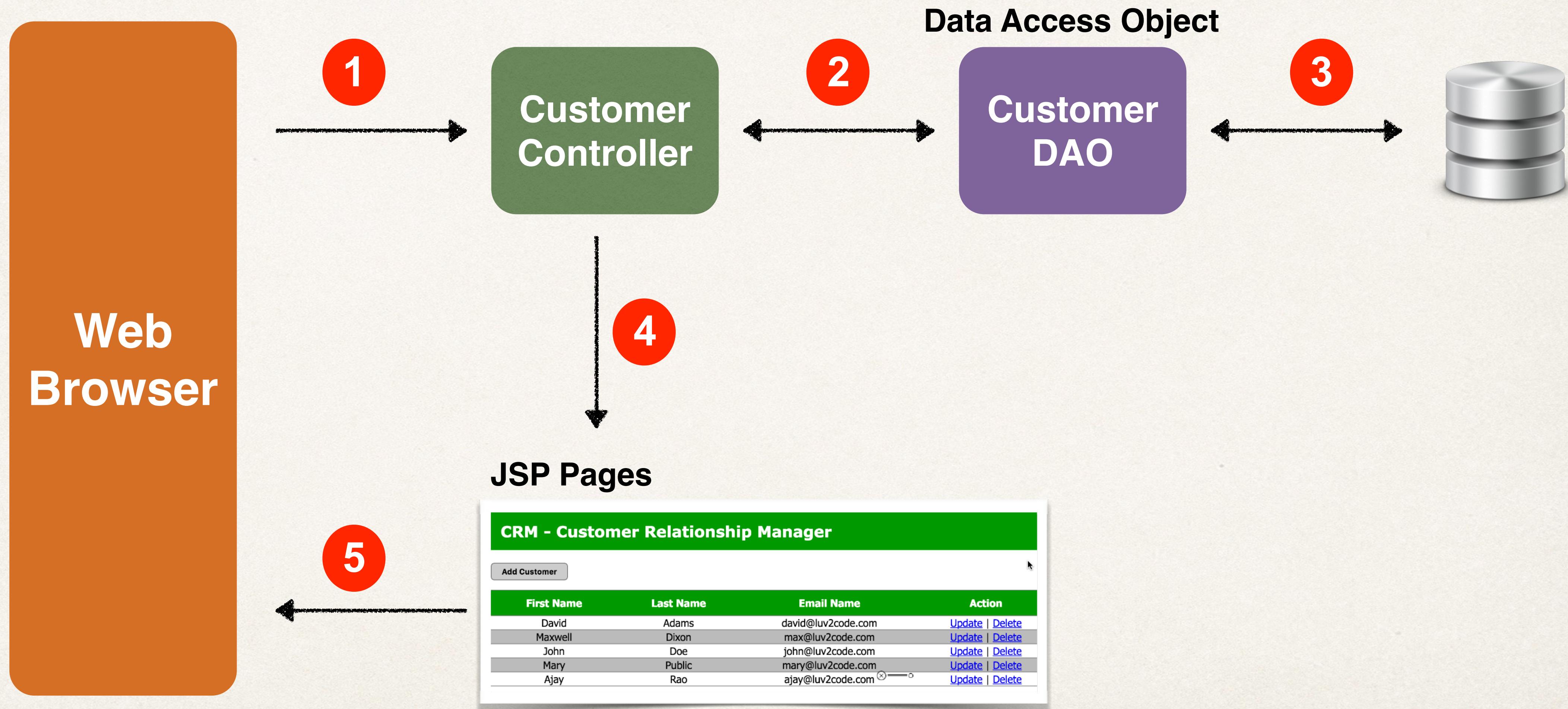


# Define Data Access Object

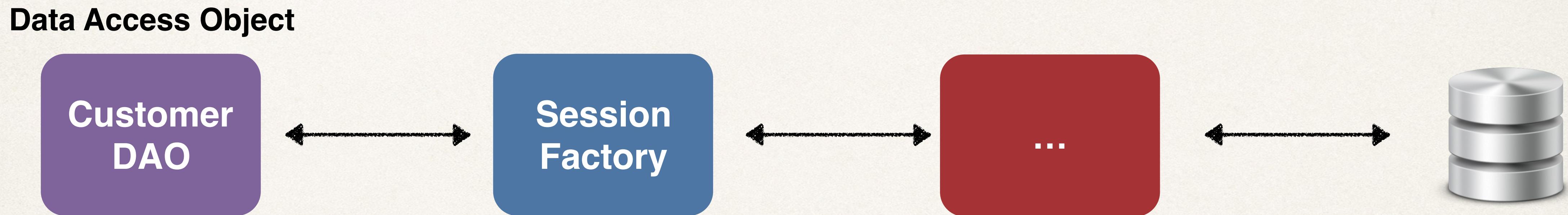


# Big Picture



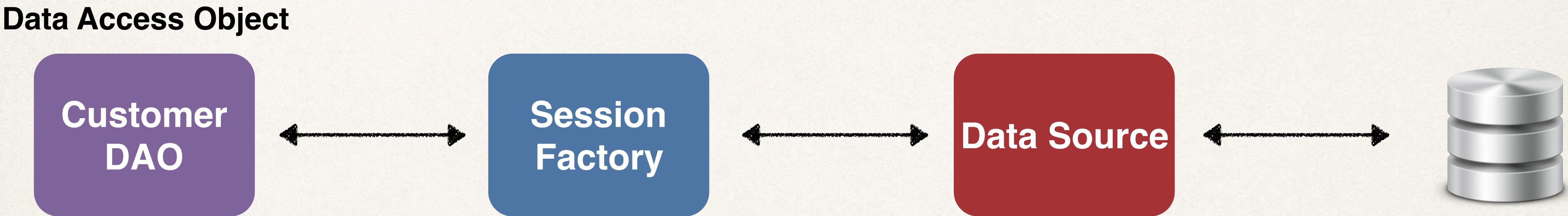
# Customer Data Access Object

- ❖ For Hibernate, our DAO needs a Hibernate SessionFactory



# Hibernate Session Factory

- ❖ Our Hibernate Session Factory needs a Data Source
  - ❖ The data source defines database connection info



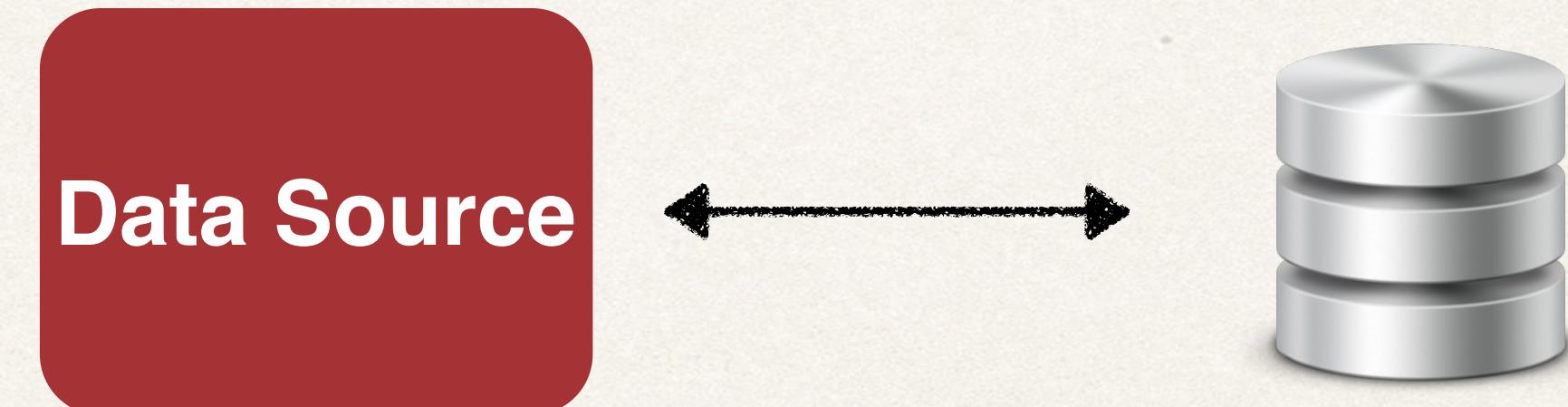
# Dependencies

**These are all dependencies!**

**We will wire them together  
with Dependency Injection (DI)**

# Data Source

```
<bean id="myDataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"  
destroy-method="close">  
  
<property name="driverClass" value="com.mysql.jdbc.Driver" />  
<property name="jdbcUrl"  
value="jdbc:mysql://localhost:3306/web_customer_tracker?useSSL=false" />  
  
... user id, password etc ...  
</bean>
```



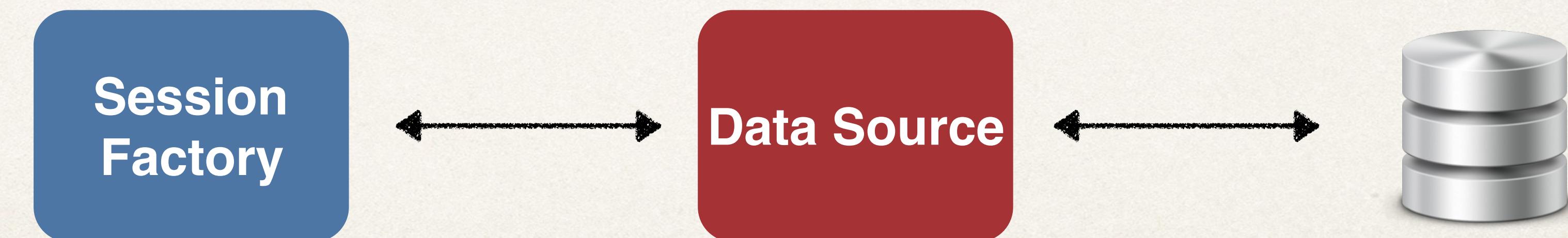
# Session Factory

```
<bean id="sessionFactory"
      class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">

    <property name="dataSource" ref="myDataSource" />

    ...

</bean>
```



# Customer DAO

*Step-By-Step*

1. Define DAO interface
2. Define DAO implementation
  - Inject the session factory

## Data Access Object



# Step 1: Define DAO interface

```
public interface CustomerDAO {  
    public List<Customer> getCustomers();  
}
```

# Step 2: Define DAO implementation

```
public class CustomerDAOImpl implements CustomerDAO {  
  
    @Autowired  
    private SessionFactory sessionFactory;  
  
    public List<Customer> getCustomers() {  
        ...  
    }  
}
```

# Spring @Transactional

- Spring provides an **@Transactional** annotation
- **Automagically** begin and end a transaction for your Hibernate code
  - No need for you to explicitly do this in your code
- This Spring **magic** happens behind the scenes

# Flash back - Standalone Hibernate code

```
// start a transaction  
session.beginTransaction();
```

```
// DO YOUR HIBERNATE STUFF HERE  
// ...
```

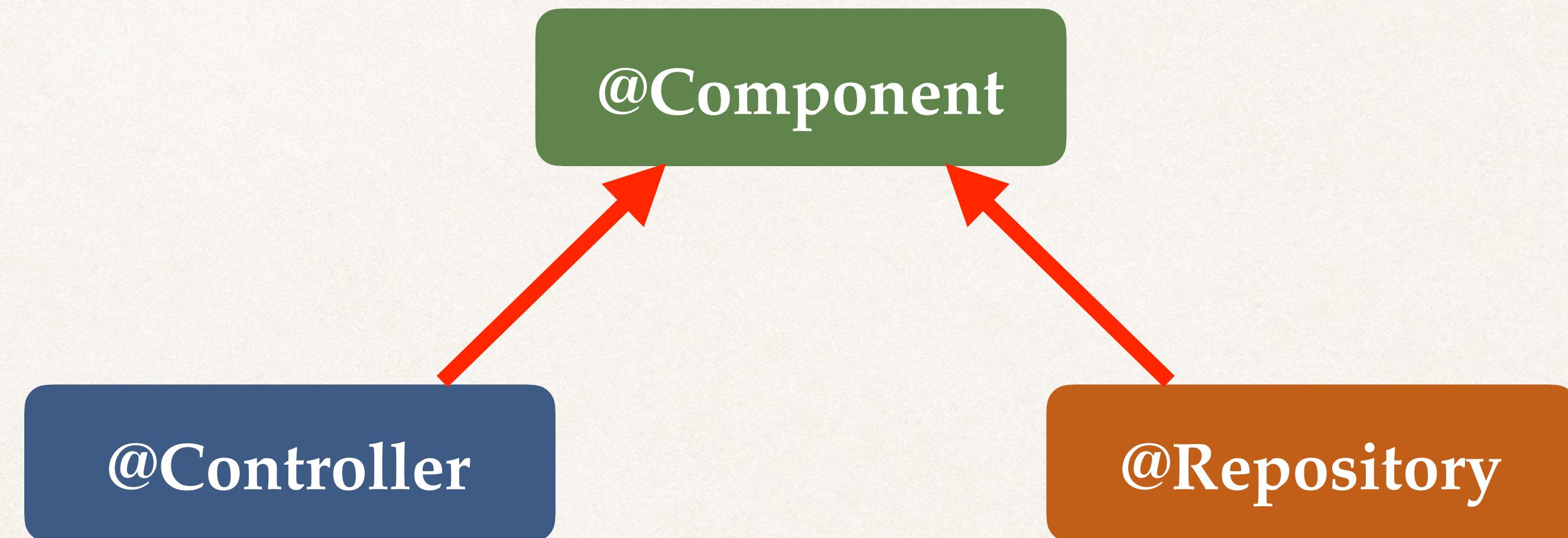
```
// commit transaction  
session.getTransaction().commit();
```

# Spring @Transactional Magic

```
@Transactional  
public List<Customer> getCustomers() {  
  
    // get the current hibernate session  
    Session currentSession = sessionFactory.getCurrentSession();  
  
    // create a query  
    Query<Customer> theQuery =  
        currentSession.createQuery("from Customer", Customer.class);  
  
    // get the result list  
    List<Customer> customers = theQuery.getResultList();  
  
    return customers;  
}
```

# Specialized Annotation for DAOs

- Spring provides the **@Repository** annotation



# Specialized Annotation for DAOs

- Applied to DAO implementations
- Spring will automatically register the DAO implementation
  - thanks to component-scanning
- Spring also provides translation of any JDBC related exceptions

# Updates for the DAO implementation

```
@Repository  
public class CustomerDAOImpl implements CustomerDAO {  
  
    @Autowired  
    private SessionFactory sessionFactory;  
  
    @Transactional  
    public List<Customer> getCustomers() {  
        ...  
    }  
}
```