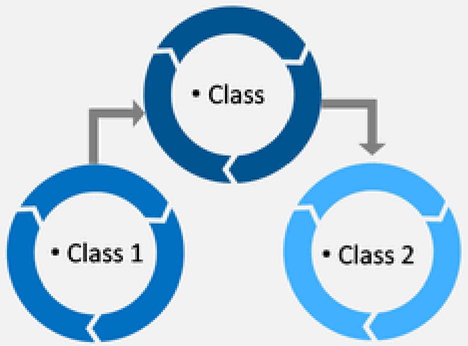
OBJECT RELATIONAL MAPPING

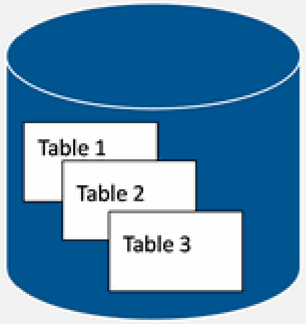
What Does ORM Do ?

Object Model

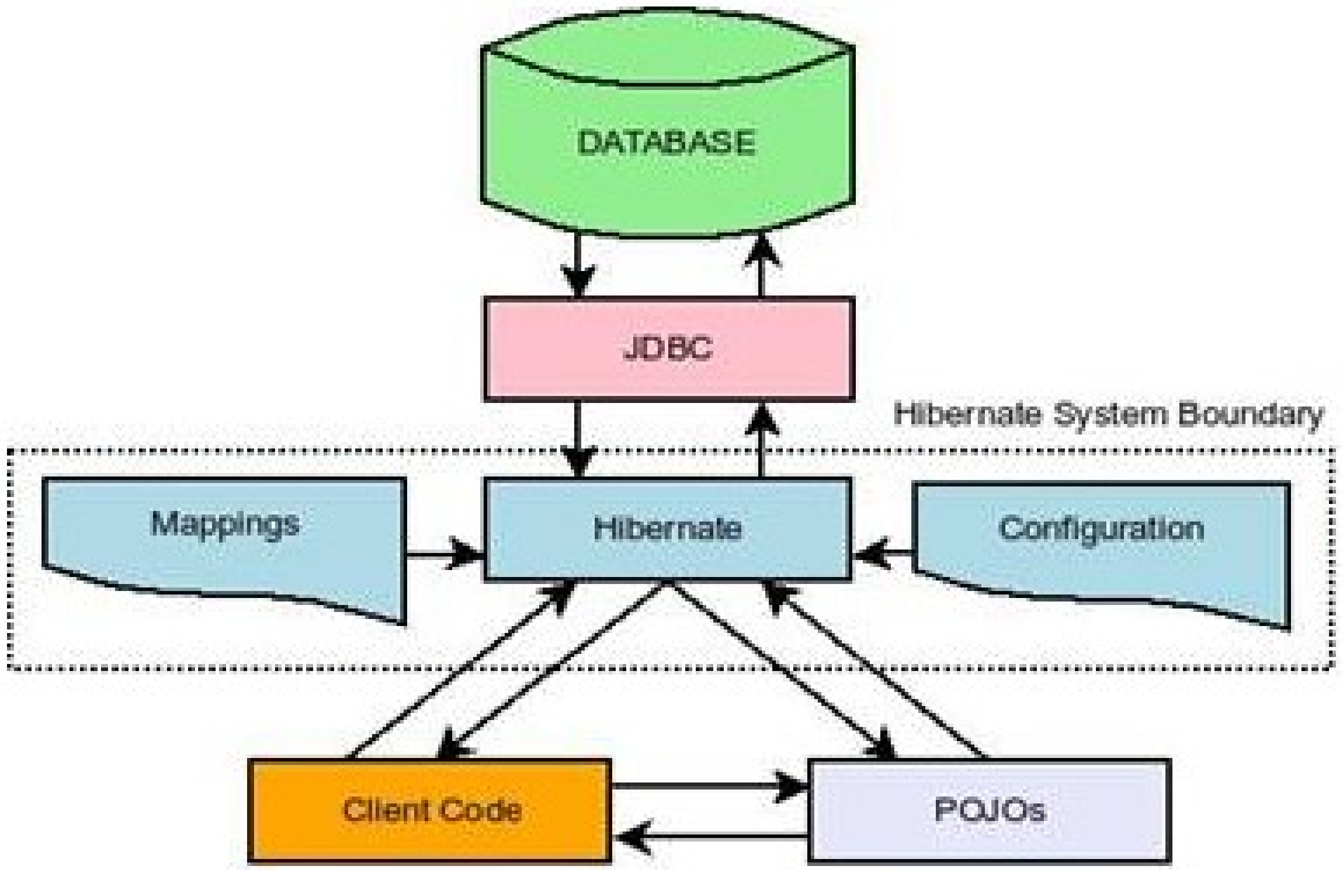


ORM

Data Model



- Maps Object Model to Relational Model.
- Resolve impedance mismatch
- Resolve mapping of scalar and non-scalar.
- Database – Independent applications.


DATABASE
JDBC
Hibernate System Boundary
Mappings
Hibernate
Configuration
Client Code
POJOs

Hibernate 5 configuration

```java
ServiceRegistry serviceRegistry=new StandardServiceRegistryBuilder().configure().build();
SessionFactory factory=new MetadataSources(serviceRegistry).buildMetadata().buildSessionFactory();

Session  session=factory.openSession();

Transaction tx=session.getTransaction();

Account account=new Account("ekta", 6000);
try {
tx.begin();
session.save(account);
tx.commit();
}catch(HibernateException ex) {
    ex.printStackTrace();
    tx.rollback();
}
```

Spring hibernate Integration

Problem with hibrenate without spring

Each DAO method must:
----

1. Obtain a EntityManager/session factory instance
2. Start a transaction
3. Perform the persistence operation
4. commit the transaction.
5. Each DAO method should include its own duplicated
   exception-handling implementation.

These are exactly the problems that motivate
us to use Spring with Hibernate

"template design patten"

Data tier implementation with Spring

----

we don't need to implement code for
obtaining Session objects, starting and committing transactions,
 and handling Hibernate exceptions.

(We use a HibernateTemplate instance to delegate
 persistence calls to Hibernate, without direct interaction
with Hibernate)

What we gains with Spring

===

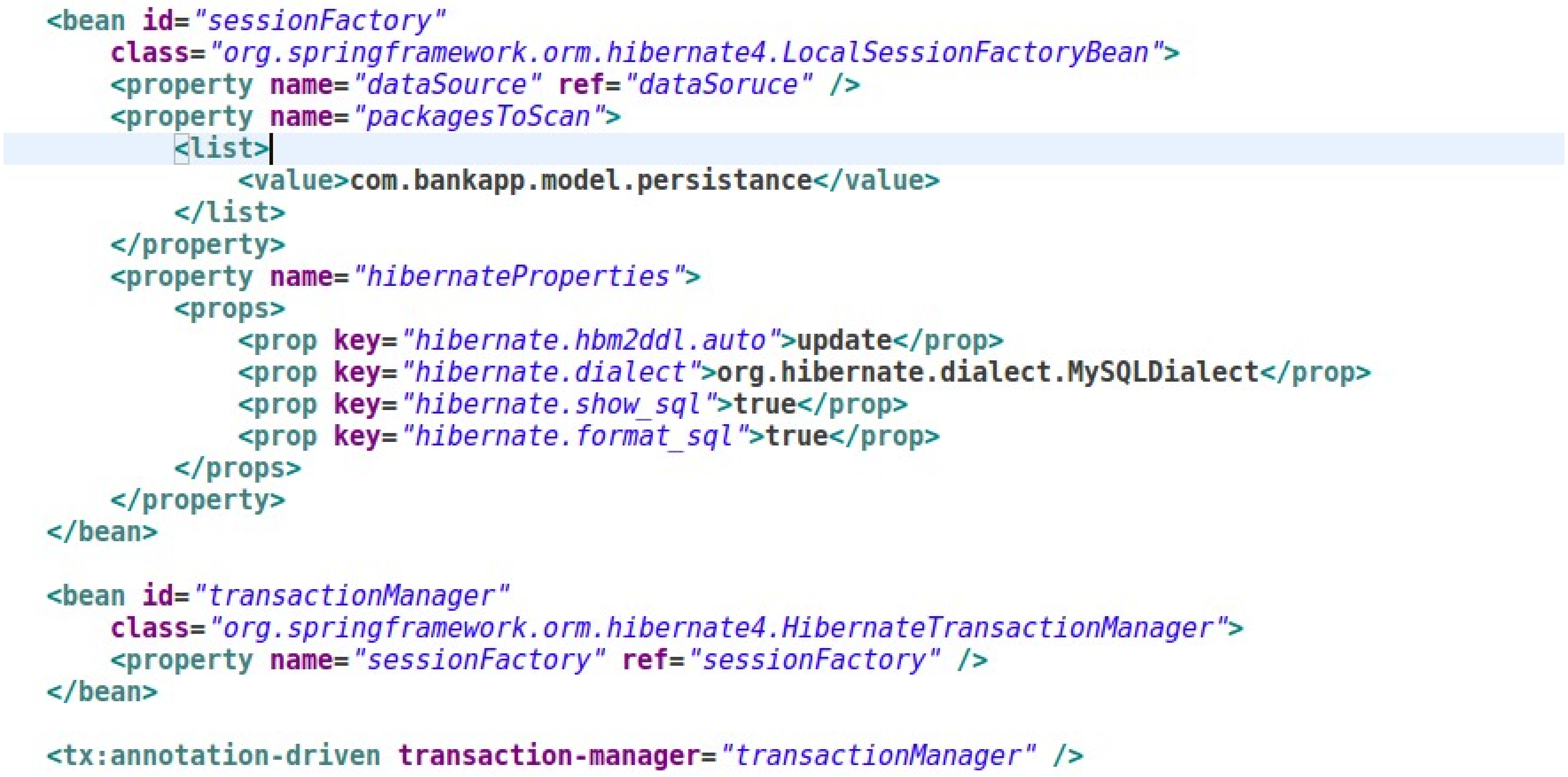
1. HibernateTemplate/JpaTemplate object removes the boilerplate code

2. Invocation of one of HibernateTemplate's methods throws
 the generic DataAccessException exception instead of   HibernateException

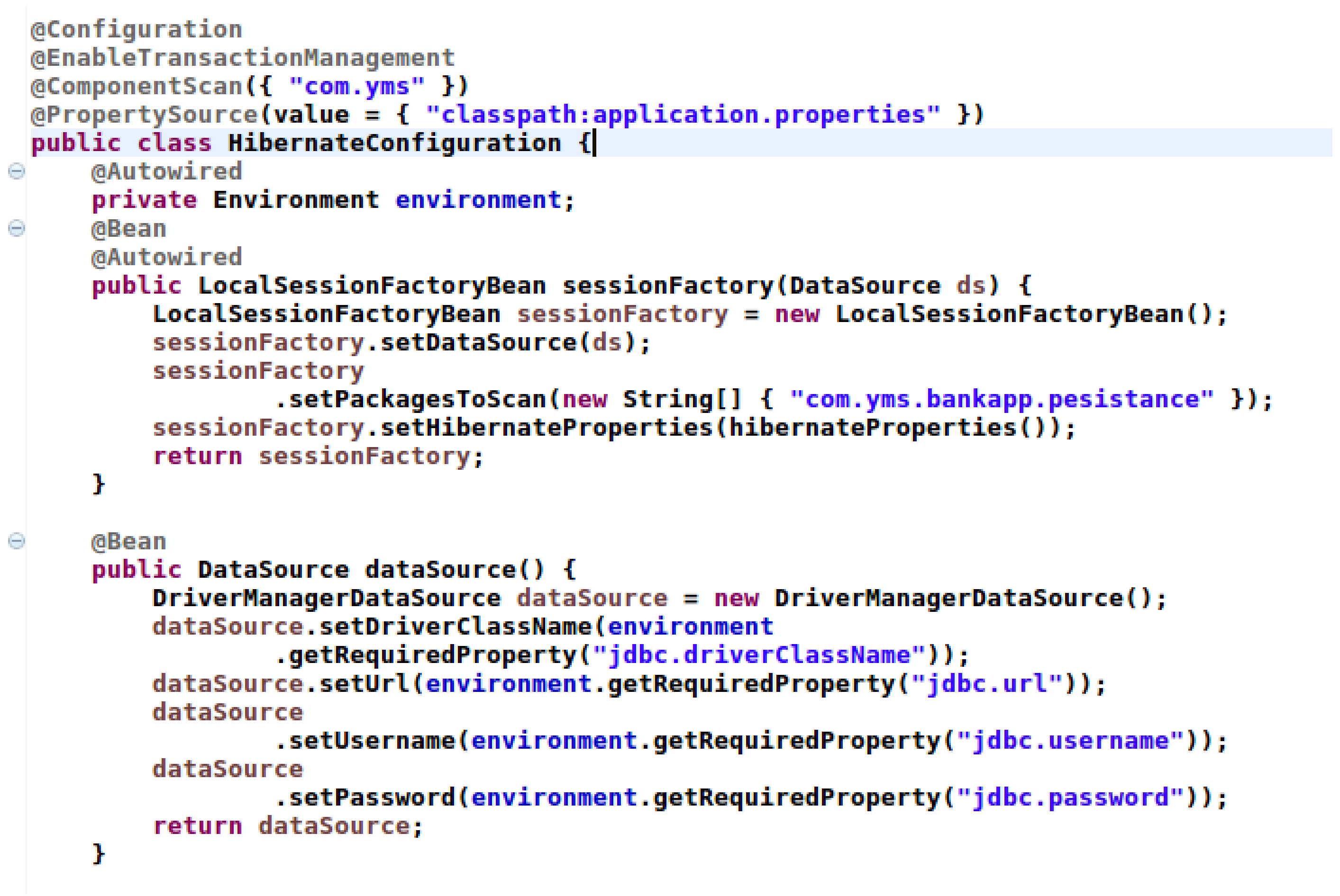
spring

HIBERNATE

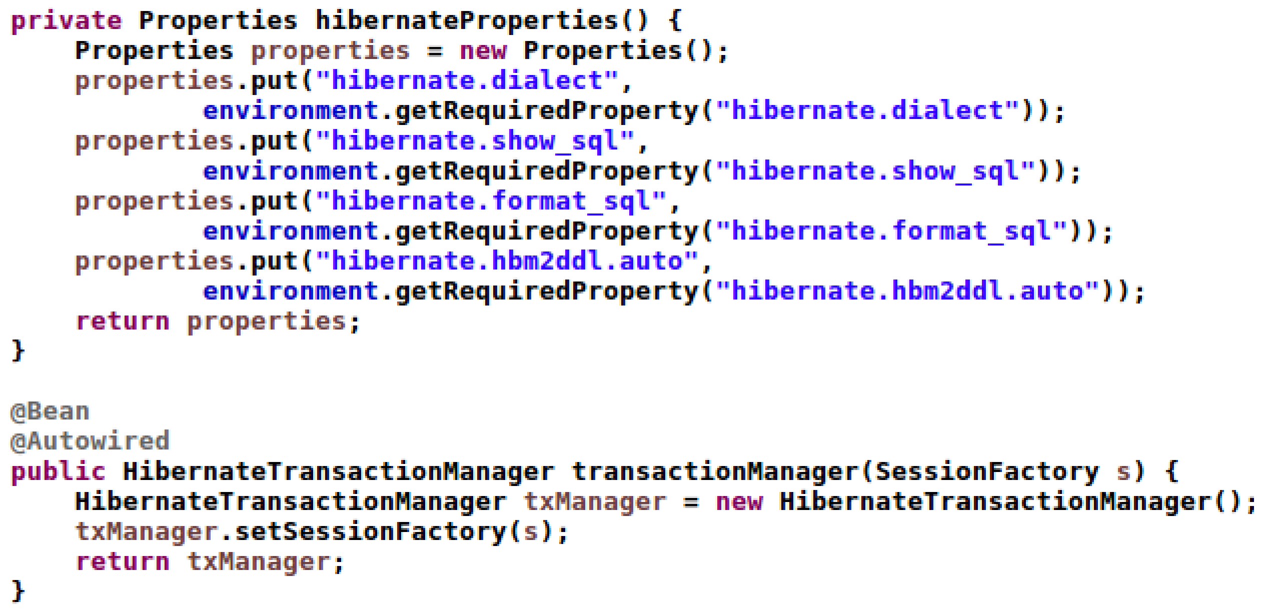
```xml
<context:annotation-config />

<context:component-scan base-package="com.bankapp" />

<bean id="dataSoruce"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="url" value="${url}" />
    <property name="driverClassName" value="${driverClassName}" />
    <property name="username" value="${username}" />
    <property name="password" value="${password}" />
</bean>

<bean
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location" value="classpath:db.properties"></property>
</bean>
```



```xml
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSoruce" />
    <property name="packagesToScan">
        <list>
            <value>com.bankapp.model.persistance</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.hbm2ddl.auto">update</prop>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.format_sql">true</prop>
        </props>
    </property>
</bean>

<bean id="transactionManager"
    class="org.springframework.orm.hibernate4.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>

<tx:annotation-driven transaction-manager="transactionManager" />
```



```java
@Configuration
@EnableTransactionManagement
@ComponentScan({ "com.yms" })
@PropertySource(value = { "classpath:application.properties" })
public class HibernateConfiguration {
    @Autowired
    private Environment environment;
    @Bean
    @Autowired
    public LocalSessionFactoryBean sessionFactory(DataSource ds) {
        LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
        sessionFactory.setDataSource(ds);
        sessionFactory
                .setPackagesToScan(new String[] { "com.yms.bankapp.pesistance" });
        sessionFactory.setHibernateProperties(hibernateProperties());
        return sessionFactory;
    }

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName(environment
                .getRequiredProperty("jdbc.driverClassName"));
        dataSource.setUrl(environment.getRequiredProperty("jdbc.url"));
        dataSource
                .setUsername(environment.getRequiredProperty("jdbc.username"));
        dataSource
                .setPassword(environment.getRequiredProperty("jdbc.password"));
        return dataSource;
    }
```



```java
private Properties hibernateProperties() {
    Properties properties = new Properties();
    properties.put("hibernate.dialect",
            environment.getRequiredProperty("hibernate.dialect"));
    properties.put("hibernate.show_sql",
            environment.getRequiredProperty("hibernate.show_sql"));
    properties.put("hibernate.format_sql",
            environment.getRequiredProperty("hibernate.format_sql"));
    properties.put("hibernate.hbm2ddl.auto",
            environment.getRequiredProperty("hibernate.hbm2ddl.auto"));
    return properties;
}

@Bean
@Autowired
public HibernateTransactionManager transactionManager(SessionFactory s) {
    HibernateTransactionManager txManager = new HibernateTransactionManager();
    txManager.setSessionFactory(s);
    return txManager;
}
```