# DAO support

## Introduction

The Data Access Object (DAO) support in Spring is aimed at making it easy to work with **data access technologies** like JDBC, Hibernate or JPA in a consistent way. This allows one to switch between the aforementioned persistence technologies fairly easily and it also allows one to code without worrying about catching exceptions that are specific to each technology.

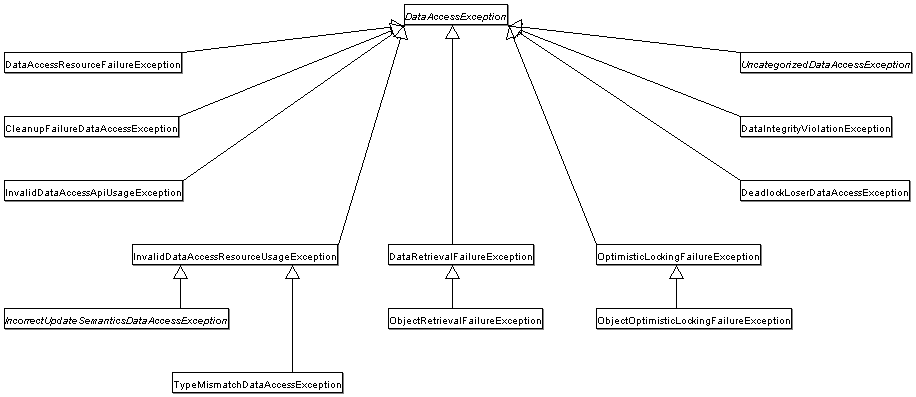
## Consistent exception hierarchy

Spring provides a convenient translation from technology-specific exceptions like SQLException to its own exception class hierarchy with the DataAccessException as the root exception. These exceptions wrap the original exception so there is never any risk that one might lose any information as to what might have gone wrong.

In addition to JDBC exceptions, Spring can also wrap Hibernate-specific exceptions, converting them to a set of focused runtime exceptions (the same is true for JPA exceptions). This allows one to handle most persistence exceptions, which are non-recoverable, only in the appropriate layers, without having annoying boilerplate catch-and-throw blocks and exception declarations in one’s DAOs. (One can still trap and handle exceptions anywhere one needs to though.) As mentioned above, JDBC exceptions (including database-specific dialects) are also converted to the same hierarchy, meaning that one can perform some operations with JDBC within a consistent programming model.

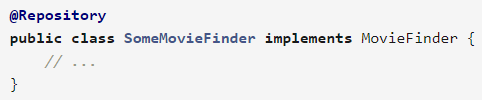
The above holds true for the various template classes in Springs support for various ORM frameworks. If one uses the interceptor-based classes then the application must care about handling HibernateExceptions and PersistenceExceptionsitself, preferably via delegating to SessionFactoryUtils’ `convertHibernateAccessException(..) orconvertJpaAccessException() methods respectively. These methods convert the exceptions to ones that are compatible with the exceptions in the org.springframework.dao exception hierarchy. As PersistenceExceptions are unchecked, they can simply get thrown too, sacrificing generic DAO abstraction in terms of exceptions though.

The exception hierarchy that Spring provides can be seen below. (Please note that the class hierarchy detailed in the image shows only a subset of the entire DataAccessException hierarchy.)

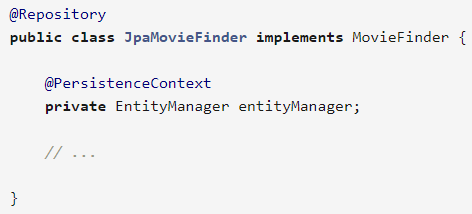


## Annotations used for configuring DAO or Repository classes

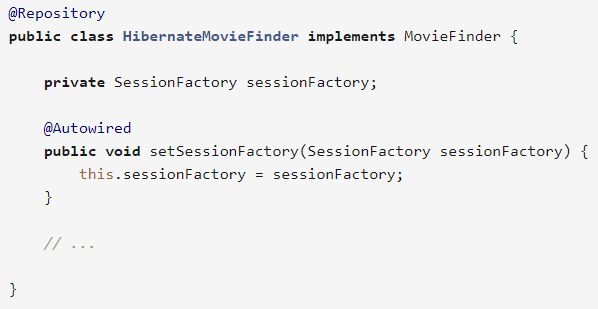
The best way to guarantee that your Data Access Objects (DAOs) or repositories provide exception translation is to use the @Repository annotation. This annotation also allows the component scanning support to find and configure your DAOs and repositories without having to provide XML configuration entries for them.



Any DAO or repository implementation will need to access to a persistence resource, depending on the persistence technology used; for example, a JDBC-based repository will need access to a JDBC DataSource; a JPA-based repository will need access to an EntityManager. The easiest way to accomplish this is to have this resource dependency injected using one of the @Autowired, @Inject, @Resource or @PersistenceContext annotations. Here is an example for a JPA repository:



If you are using the classic Hibernate APIs than you can inject the SessionFactory:



Last example we will show here is for typical JDBC support. You would have the DataSource injected into an initialization method where you would create a JdbcTemplate and other data access support classes like SimpleJdbcCall etc using this DataSource.

