Spring DAO

(Data Access Module of Spring Framework)

Introduction

- The Spring Data Access Object (DAO) support makes it easy to work with data access technologies like JDBC, Hibernate or JDO in a standardized way

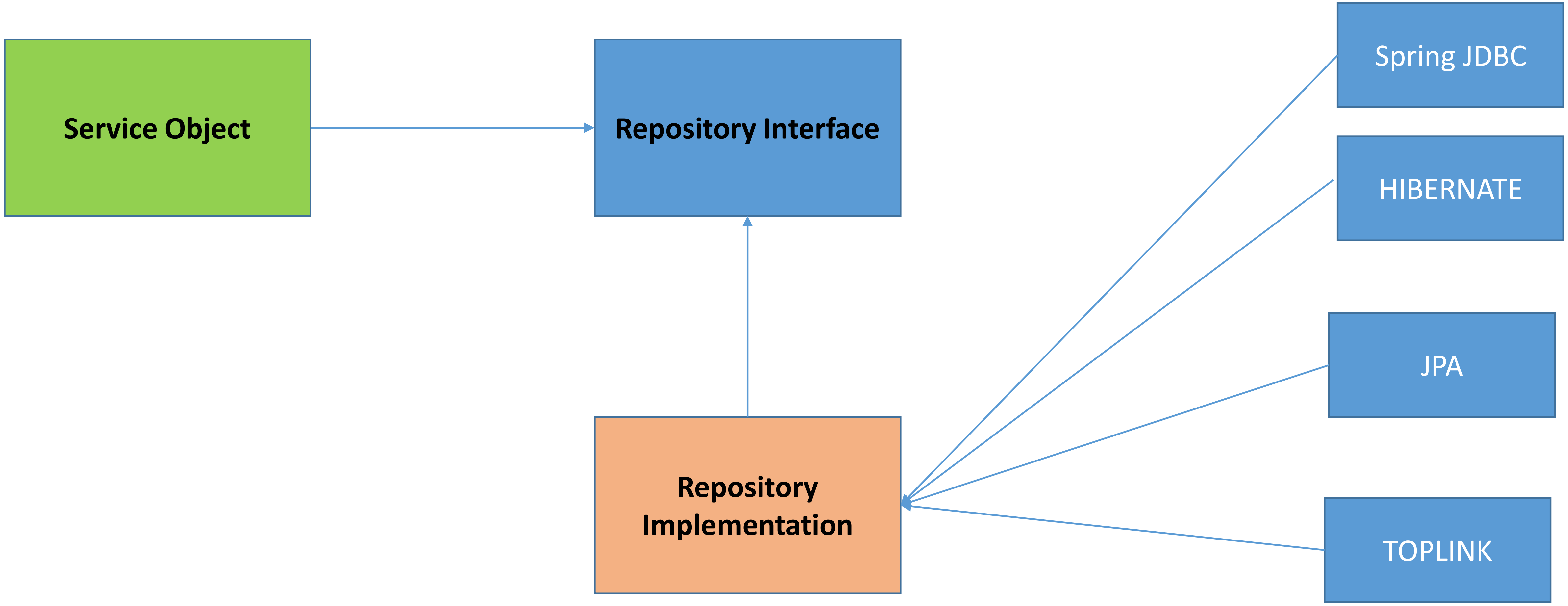
- Makes switching between databases easy and simple

- Code without Worrying about catching exceptions
  - Spring will do it for you!!!

Introduction

- Spring comes with a family of data-access frameworks that integrate with a variety of data-access technologies.
- You can persist your data via direct JDBC or an object-relational mapping (ORM) framework such as Hibernate.
- Spring removes the tedium of data access from your persistence code. Instead, you can lean on Spring to handle the

How Spring Applications Access Data?



Spring’s data-access exception hierarchy

The Bad SQLException in Data Access

- You can’t do anything with JDBC without being forced to catch **SQLException.**

- SQLException means something went wrong while trying to access a database.

- There’s little about that exception that tells you what went wrong or how to deal with it.

- Many of the problems that trigger a SQLException can’t be remedied in a catch block.

Spring JDBC Consistent Exception Hierarchy

- Spring JDBC provides a hierarchy of data-access exceptions that solve the problems associated with SQLException

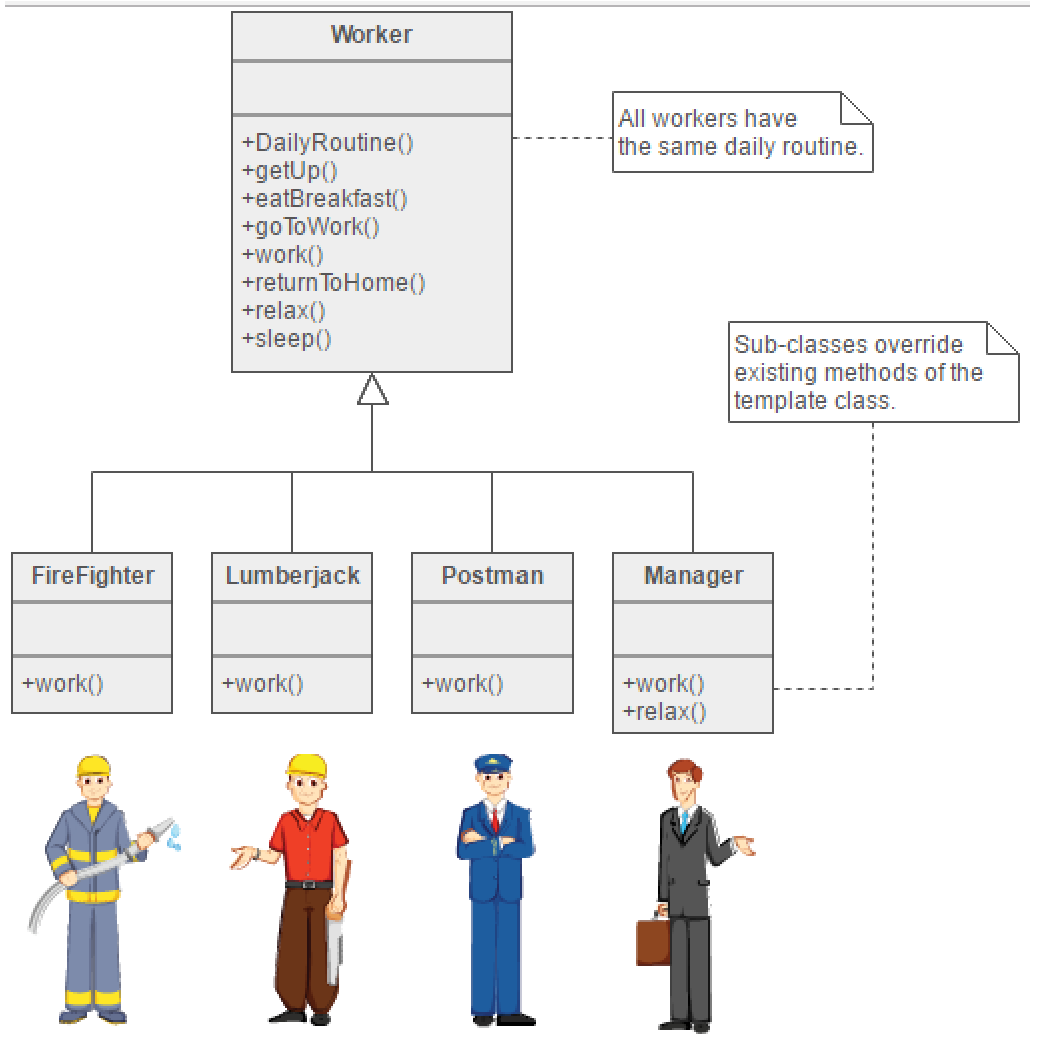
- Spring provides several data-access exceptions, each descriptive of the problem for which they’re thrown.
  - Spring Data Access Exceptions are Platform Agnostic
  - it isn’t associated with any particular persistence solution.
  - You can count on Spring to throw a consistent set of exceptions, regardless of which persistence provider you choose

- Spring’s Data Access Exceptions are unchecked exceptions
  - You are not forced to catch them

Templating data access

Template Method Pattern

- A template method defines the skeleton of a process.
- Defers some steps to subclasses.
- In Template Method, some steps are Invariants (remains same) and some are Variants (Changes).



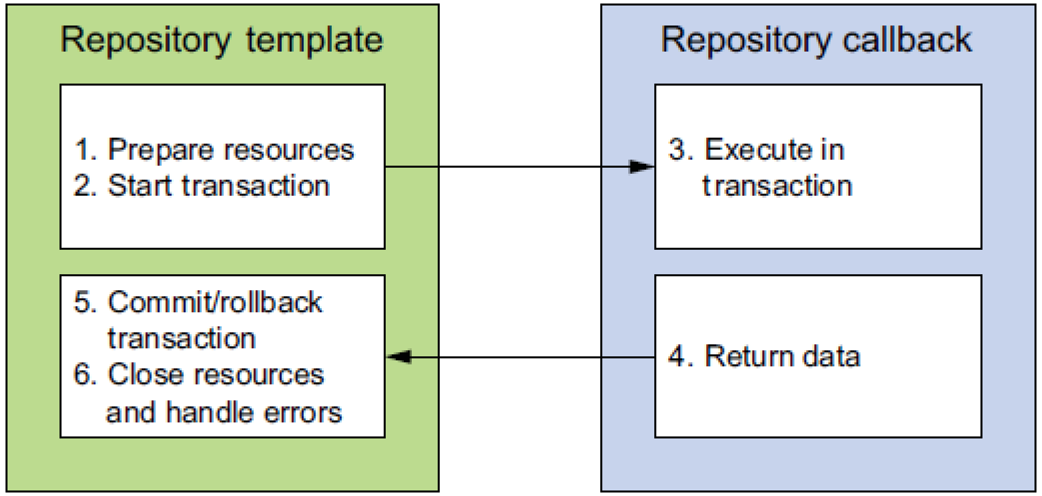
Template Method Pattern in Spring Data Access

- No matter what technology you’re using, certain data-access steps are required.
- Certain fixed steps are required by all data access technologies
  - Obtain a connection
  - Clean up resources
- There are certain steps which are different for different data access technologies (e.g. JDBC, Hibernate, JPA…)
  - You query for different objects and update the data in different ways.
  - These are the variable steps in the data-access process

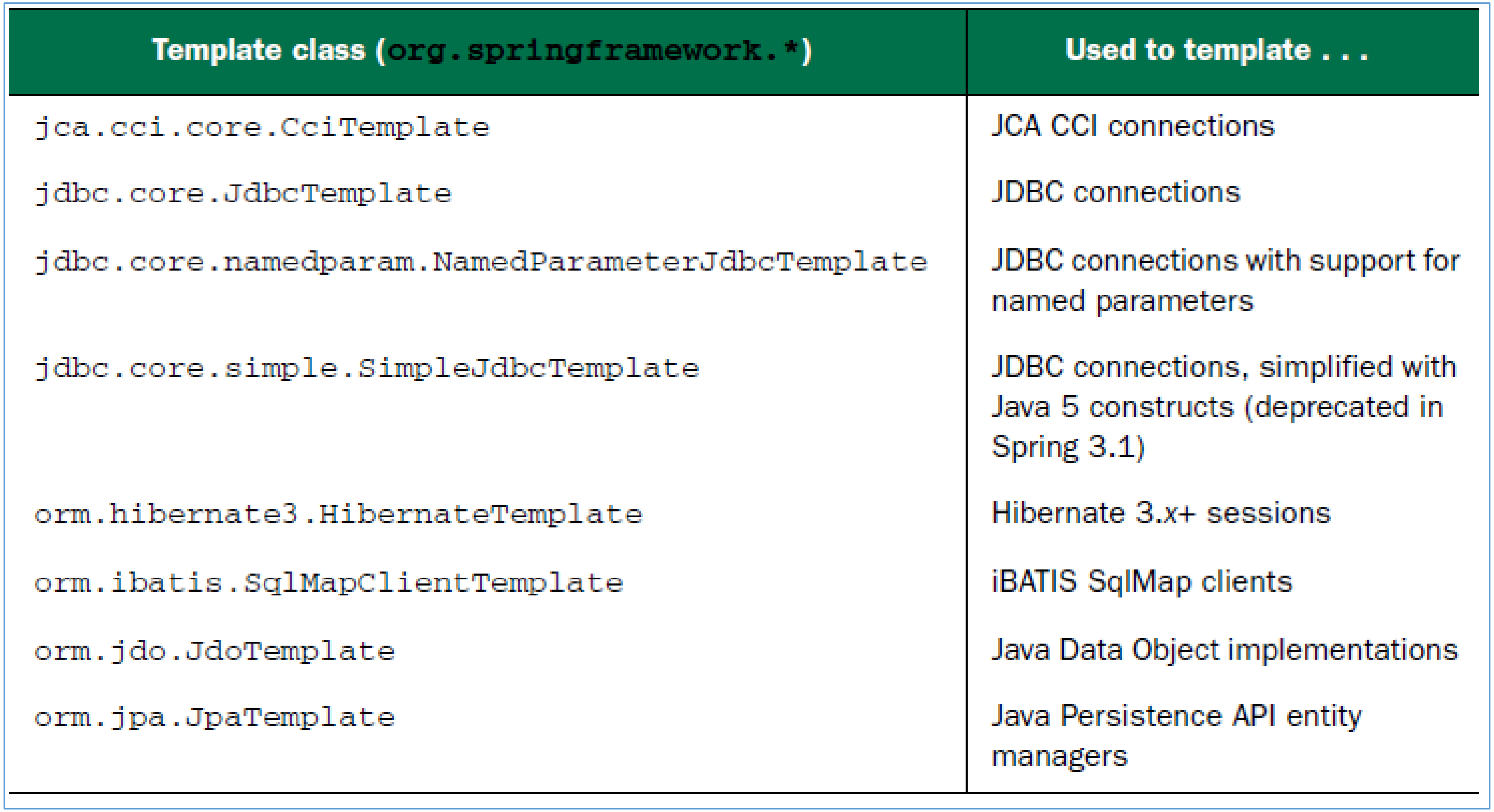
Template Method Pattern in Spring Data Access

- The same Template Method pattern is used in Spring's Data Access.
- Spring separates the fixed and variable parts of the data-access process into two distinct classes:
  - *templates*
  - *Callbacks*
- Templates manage the fixed part of the process, whereas your custom data-access code is handled in callbacks

Spring's data-access template classes take responsibility for common data-access duties. For application-specific tasks, it calls back into a custom callback object.



Spring’s Data Access Template Classes

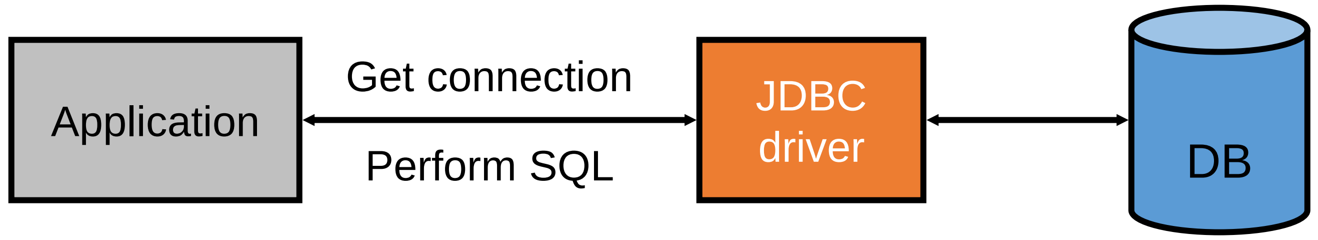


Template class (`org.springframework.*`)	Used to template . . .
`jca.cci.core.CciTemplate`	JCA CCI connections
`jdbc.core.JdbcTemplate`	JDBC connections
`jdbc.core.namedparam.NamedParameterJdbcTemplate`	JDBC connections with support for named parameters
`jdbc.core.simple.SimpleJdbcTemplate`	JDBC connections, simplified with Java 5 constructs (deprecated in Spring 3.1)
`orm.hibernate3.HibernateTemplate`	Hibernate 3.*x*+ sessions
`orm.ibatis.SqlMapClientTemplate`	iBATIS SqlMap clients
`orm.jdo.JdoTemplate`	Java Data Object implementations
`orm.jpa.JpaTemplate`	Java Persistence API entity managers

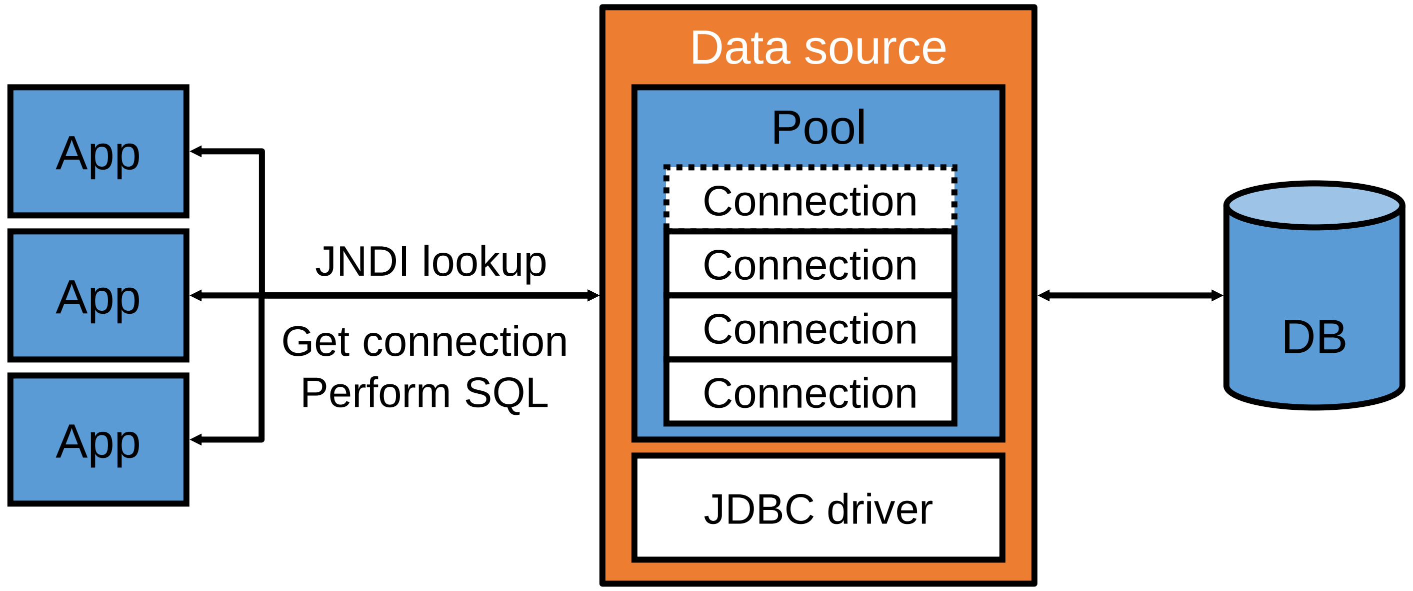
Configuring a DataSource

What is a DataSource ?

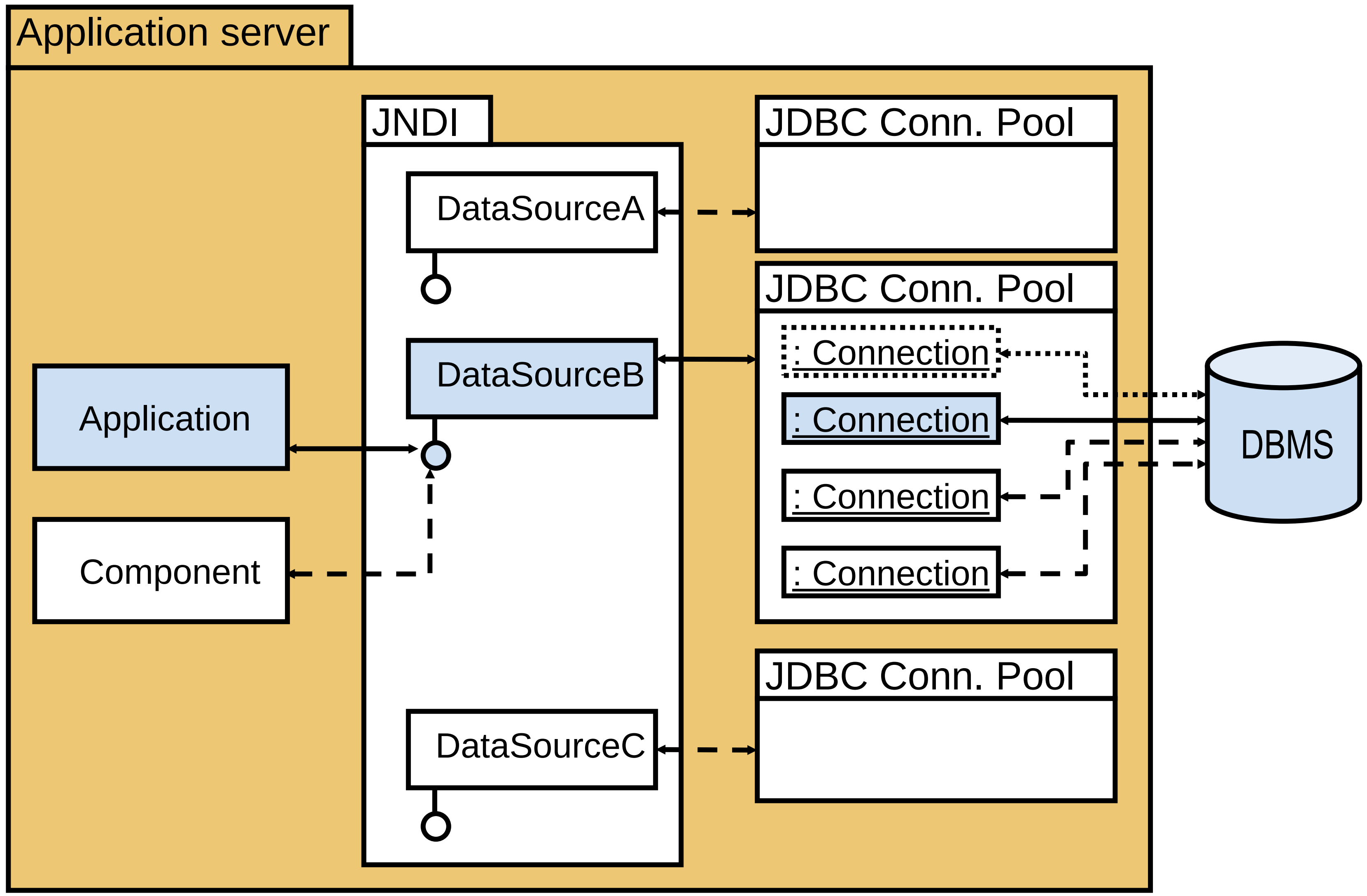
JDBC Review



JDBC Data Sources



JDBC Connection Pooling

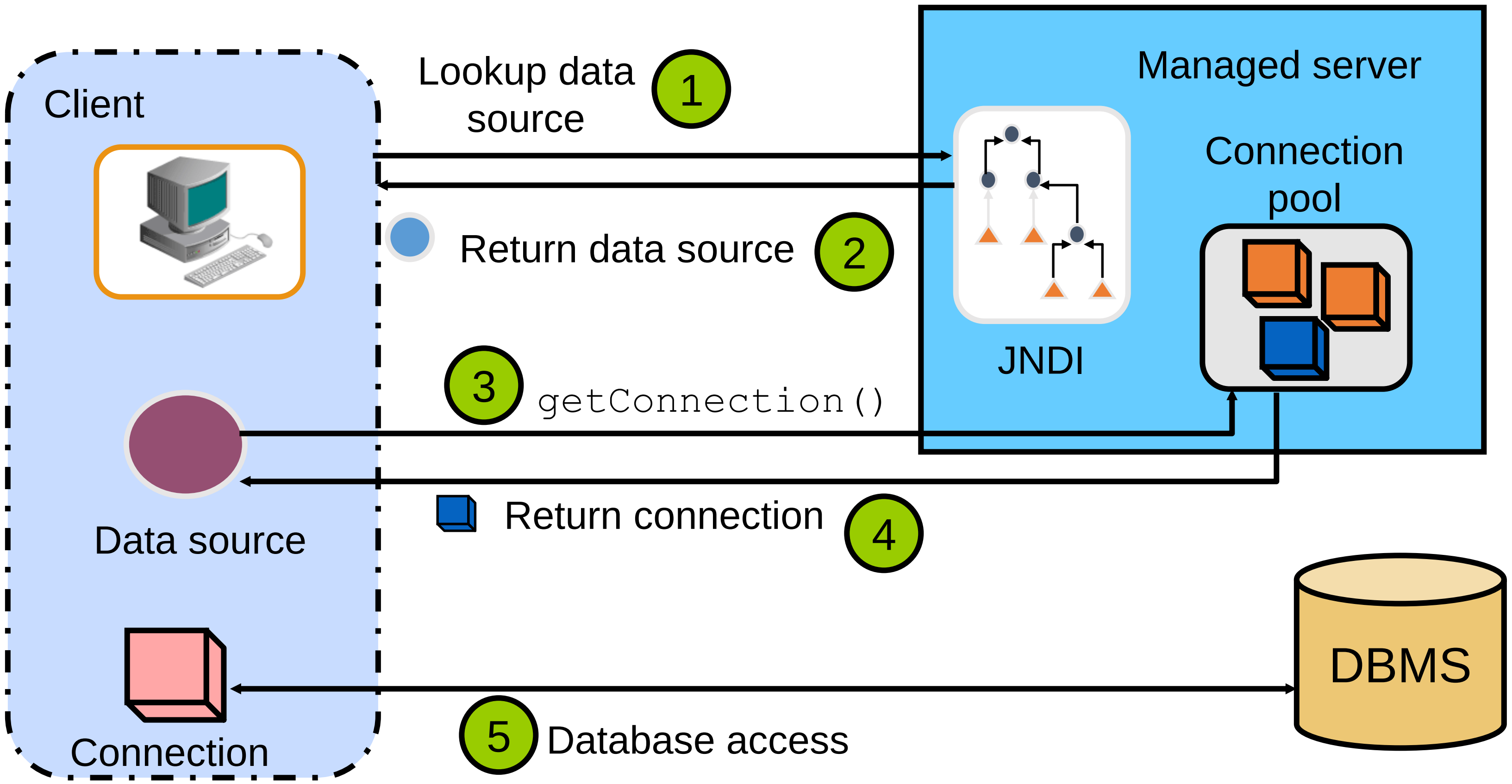


Benefits of Connection Pools

- Connection time and overhead are saved by using an existing database connection.
- It facilitates easier management because connection information is managed in one location.
- The number of connections to a database can be controlled.
- The DBMS can be changed without the application developer having to modify the underlying code.
- A connection pool allows an application to “borrow” a DBMS connection.

How Data Source Connection Pools Are Used

- A client retrieves a data source through a JNDI lookup and uses it to obtain a database connection.



DataSource Support in Spring Framework

- Spring has excellent support for DataSource
- Spring DAO Module implements DataSource interface and gives you the benefits of Using DataSource without the need of Web Container or Application Server to create it
- The out of the box implemented datasources are primarily used for development and testing your application.
- When you deploy your application in production server, you are recommended to use datasource resourced from Web Server (e.g Tomcat) or Application Server (e.g. WebLogic).

Configuring a data source

- For spring-supported data access you use, you’ll likely need to configure a reference to a data source.
- Spring offers several options for configuring data-source beans in your Spring application, including these:
  - Data sources that are defined by a JDBC driver
  - Data sources that are looked up by JNDI
  - Data sources that pool connections
- For production-ready applications, It is recommended to use a data source that draws its connections from a connection pool.

Using JNDI data sources

- When a Spring application is deployed in a JEE Server (Application Server) like Jboss or WebLogic, or even a web container like Tomcat, it can get a DataSource resourced from JNDI.
- Spring can wire a DataSourec in JEE environment using

```
<jee:jndi-lookup id="dataSource"
jndi-name="/jdbc/myDS“ resource-ref="true" />
```

- Alternatively, if you're using Java configuration, you can use JndiObjectFactoryBean to look up the DataSource from JNDI

```
@Bean
public JndiObjectFactoryBean dataSource() {
JndiObjectFactoryBean jndiObjectFB = new JndiObjectFactoryBean();
jndiObjectFB.setJndiName("jdbc/myDS");
jndiObjectFB.setResourceRef(true);
jndiObjectFB.setProxyInterface(javax.sql.DataSource.class);
return jndiObjectFB;
}
```

Using a pooled data source (XML Configuration)

- You can configure a pooled data source directly in Spring.
- Although Spring doesn’t provide a pooled data source, plenty of suitable ones are available.
- A few popular open source options:
  - Apache Commons DBCP (http://jakarta.apache.org/commons/dbcp)
  - c3p0 (http://sourceforge.net/projects/c3p0/)
  - BoneCP (http://jolbox.com/)
- DBCP’s BasicDataSource is configured in XML as

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
p:driverClassName="org.h2.Driver"
p:url="jdbc:h2:tcp://localhost/~/demodb"
p:username="sa"
p:password=""
p:initialSize="5"
p:maxActive="10" />
```

Using a pooled data source (Java Configuration)

```
@Bean
public BasicDataSource dataSource() {
BasicDataSource ds = new BasicDataSource();
ds.setDriverClassName(“com.mysql.jdbc.Driver");
ds.setUrl(“jdbc:mysql://localhost:3306/demodb");
ds.setUsername(“root");
ds.setPassword(“root");
ds.setInitialSize(5);
ds.setMaxActive(10);
return ds;
}
```

Using JDBC driver-based data sources

- Spring Provided simplest DataSources for Development and testing.
- Spring offers three such data-source classes to choose from.

  - **DriverManagerDataSource**
    - Returns a new connection every time a connection is requested.
    - Not pooled.

  - **SimpleDriverDataSource**
    - Works much the same as DriverManagerDataSource except that it works with the JDBC driver directly to overcome class loading issues that may arise in certain environments, such as in an OSGi container.

  - **SingleConnectionDataSource**
    - Returns the same connection every time a connection is requested.

How do We configure **DriverManagerDataSource**

```
@Bean
public DriverManagerDataSource dataSource() {
DriverManagerDataSource ds = new DriverManagerDataSource ();
ds.setDriverClassName(“com.mysql.jdbc.Driver");
ds.setUrl(“jdbc:mysql://localhost:3306/demodb");
ds.setUsername(“root");
ds.setPassword(“root");
return ds;
}
```

In XML, the DriverManagerDataSource can be configured as follows:

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource"
p:driverClassName=" com.mysql.jdbc.Driver "
p:url=" jdbc:mysql://localhost:3306/demodb "
p:username=“root"
p:password=“root" />
```

Using JDBC with Spring

Working with JDBC templates

- Spring abstracts away the boilerplate data access code behind template classes.
- For JDBC, Spring comes with three template classes to choose from:
  - **JdbcTemplate**
    - The most basic of Spring's JDBC templates, this class provides simple access to a database through JDBC and indexed-parameter queries.
  - **NamedParameterJdbcTemplate**
    - This JDBC template class enables you to perform queries where values are bound to named parameters in SQL, rather than indexed parameters.
  - **SimpleJdbcTemplate**
    - This version of the JDBC template takes advantage of Java 5 features such as autoboxing, generics, and variable parameter lists to simplify how a JDBC template is used.

If we have to Access Data using JDBC …

- Normally we write the code in the following way
  - Define connection parameters
  - Open the connection
  - *Specify the statement*
  - Prepare and execute the statement
  - Set up the loop to iterate through the results (if any)
  - *Do the work for each iteration*
  - Process any exception
  - Handle transactions
  - Close the connection

You need to do only these steps in Spring JDBC

Using JDBCTemplate

- Spring provides some out of the Box Callback interfaces which need to be implemented by the code using **JDBCTemplate**.
- Some of the callbacks are
  - **PreparedStatementCreator**
    - Creates a PreparedStatement given a connection
  - **CallableStatementCreator interface**
    - which creates callable statement
  - **RowCallbackHandler interface**
    - extracts values from each row of a ResultSet
  - **RowMapper**
    - Maps rows of a ResultSet on a per-row basis.
    - Implementations of this interface perform the actual work of mapping each row to a result object, but don't need to worry about exception handling
  - **ResultSetExtractor**
    - used by JdbcTemplate's query methods. Implementations of this interface perform the actual work of extracting results from a ResultSet, but don't need to worry about exception handling

Configuring a JdbcTemplate

```
@Bean
public JdbcTemplate jdbcTemplate(DataSource dataSource) {
return new JdbcTemplate(dataSource);
}
```

OR

```
<bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
                    <property name="dataSource" ref=“dataSource" />
</bean>
```

Demo:
Using JdbcTemplate