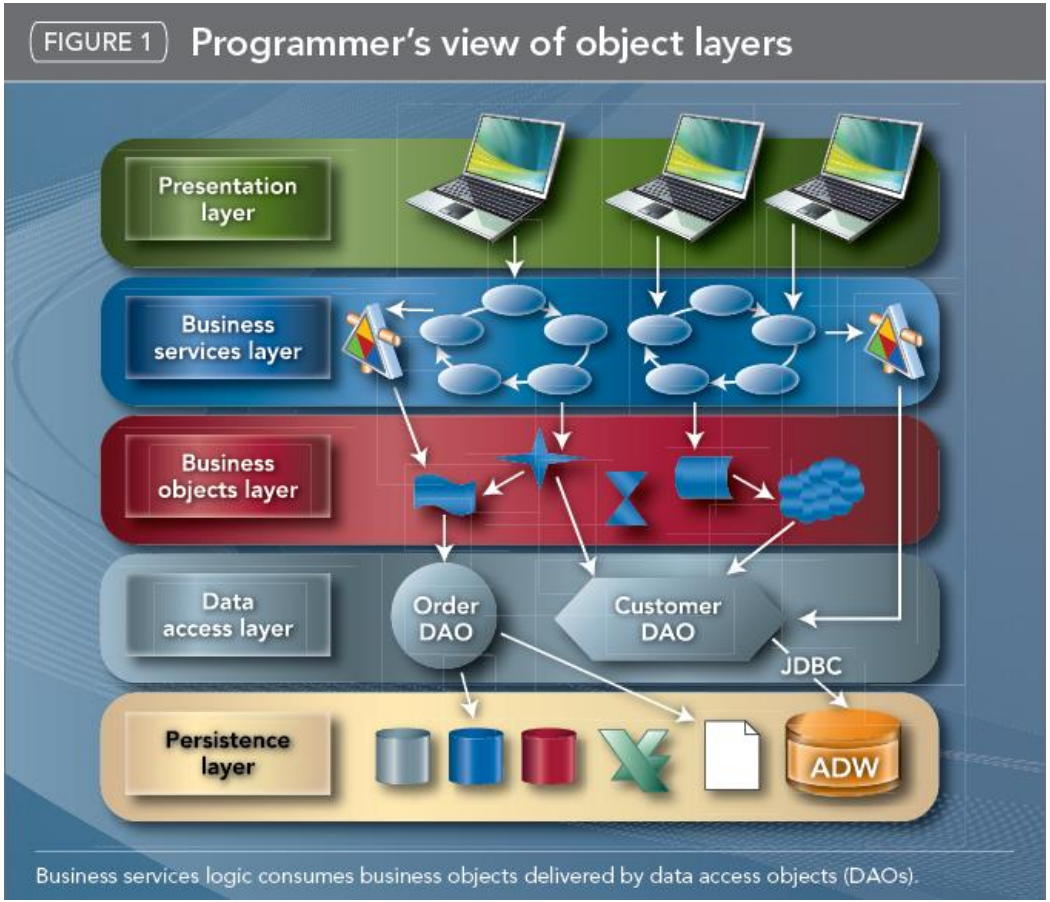
Spring Databases

Module 1
Spring JDBC

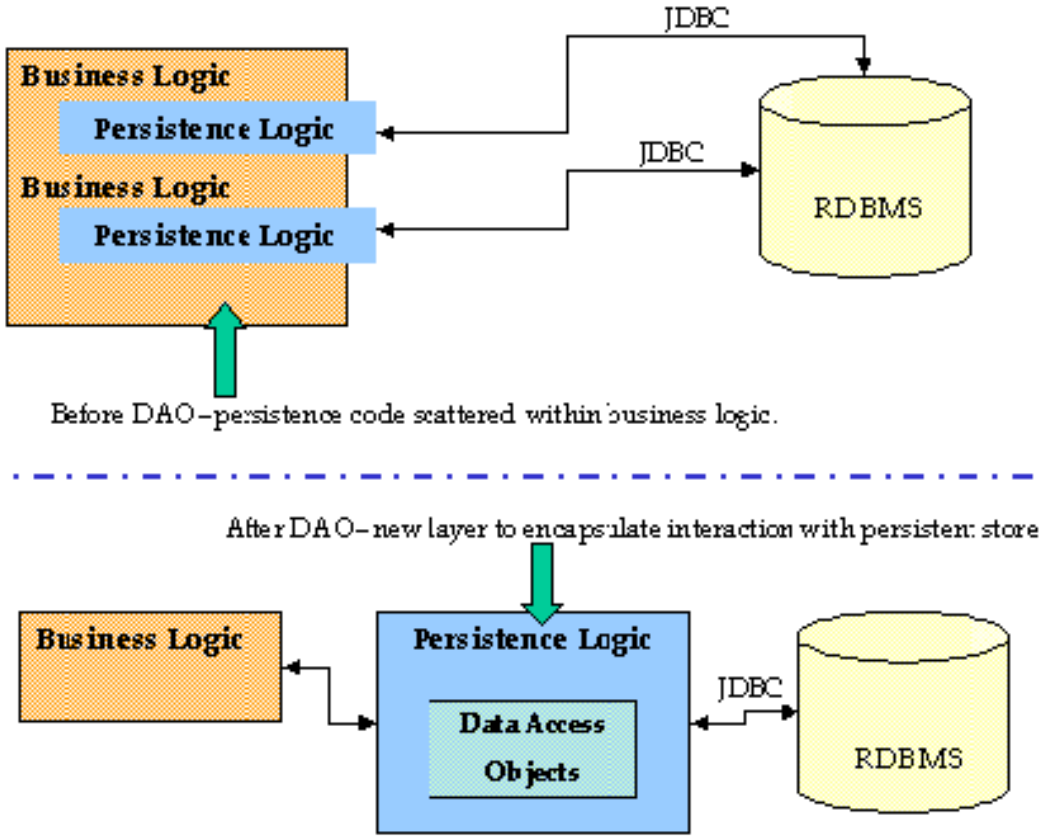
LUXOFT TRAINING

LXFT LISTED NYSE

Spring :: DAO Design Pattern



Spring :: DAO Design Pattern

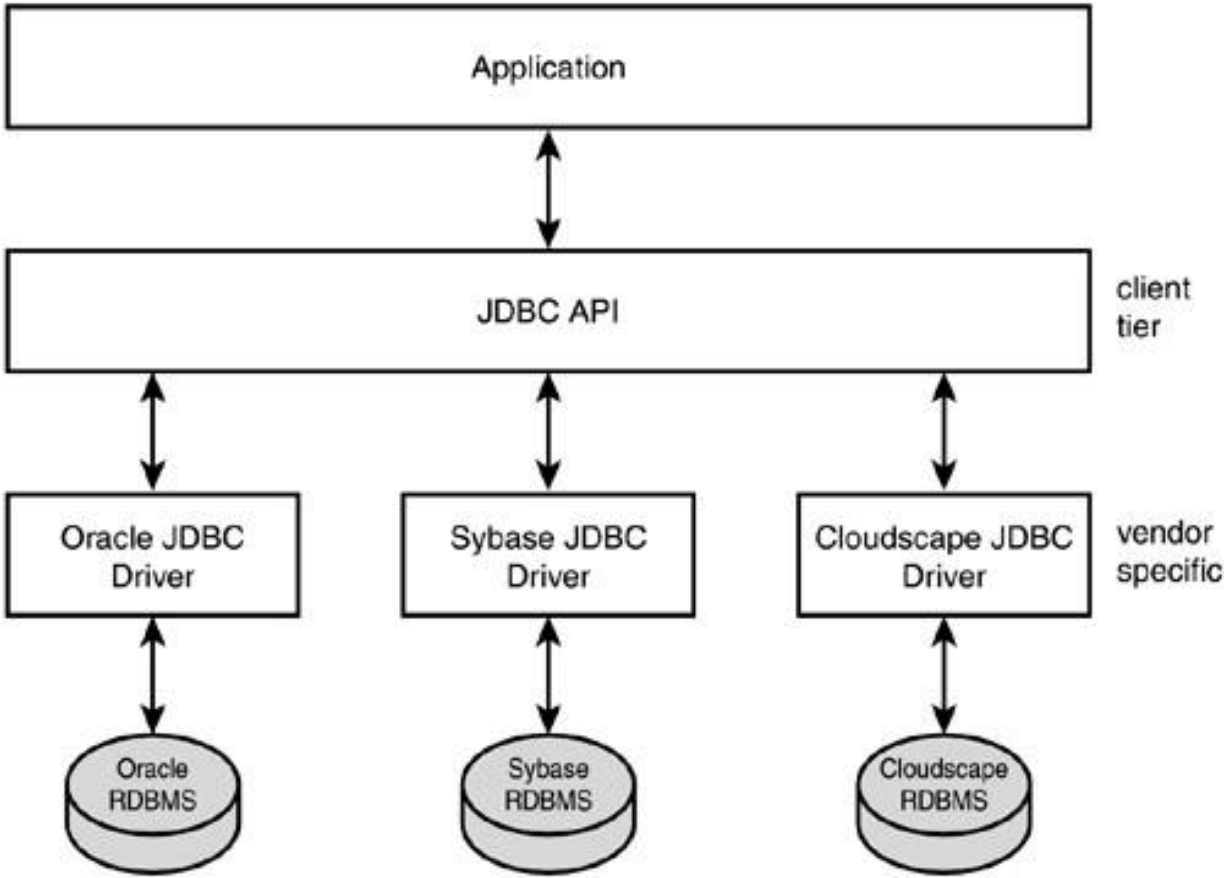


Spring :: DAO Design Pattern

```
public interface BookDao {
  public void printAll();
  public void insert(Book book);
  public void update(int id, String title);
  public void delete(Book book);
  public Book getById(int id);
}
```

BOOK
ID: integer **TITLE** : varchar **DATE_RELEASE** : timestamp

Spring :: JDBC support



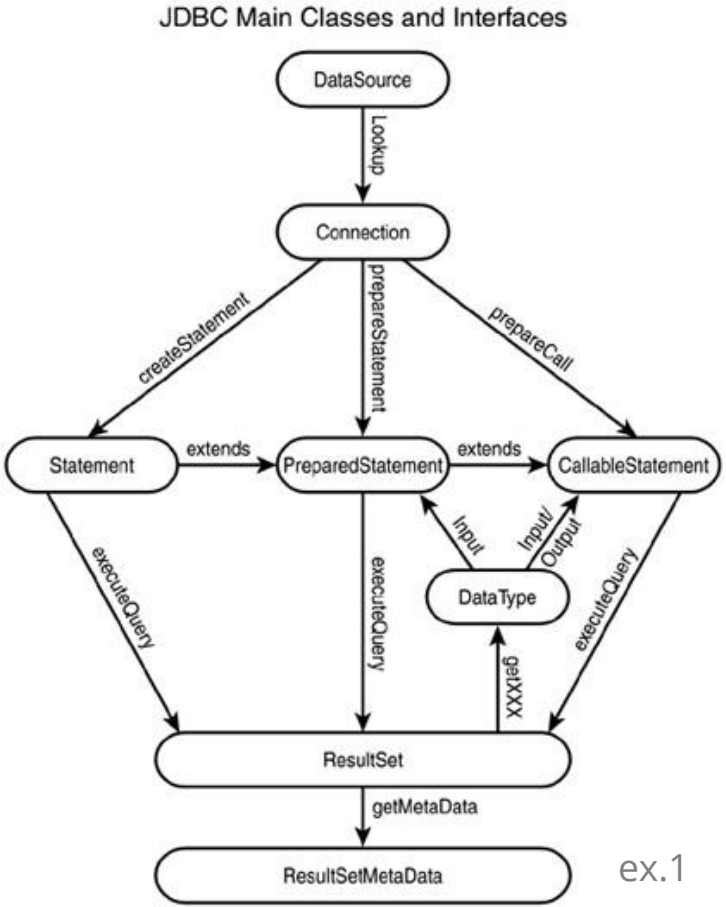
Spring :: JDBC support

- **Why use JDBC, if there is ORM?**
  - Flexibility: using all RDBMS possibilities
  - JDBC transparence – everything is under the control. ORM is creating the SQL commands by itself.
  - Performance
  - No magic

- **Why (plain) JDBC is not enough?**
  - Manual exception handling
  - Manual transaction management
  - No mapping of data to the objects
  - Big amount of the service code

Spring :: Plain JDBC example

```
@Override
public void insert(Book book) {
    String sql = "INSERT INTO BOOK (TITLE, DATE_RELEASE)
    VALUES (?, ?)";
    PreparedStatement statement;

    try {
        Connection connection = openConnection();
        statement = connection.prepareStatement(sql);
        statement.setString(1, book.getTitle());
        statement.setDate(2, new
        java.sql.Date(book.getDateRelease().getTime()));
        statement.executeUpdate();
        statement.close();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        closeConnection();
    }
}
```



ex.1

Spring :: Spring+JDBC example

```
public void setDataSource(DataSource dataSource) {
    this.dataSource = dataSource;
    jdbcTemplate = new JdbcTemplate(this.dataSource);
}

@Override
public void insert(Book book) {
    String sql = "INSERT INTO BOOK (TITLE, DATE_RELEASE) VALUES (?, ?)";
    jdbcTemplate.update(sql, new Object[] { book.getTitle(),
                  new java.sql.Date(book.getDateRelease().getTime()) });
}
```

ex.2

Spring :: JDBC Support

Without Spring:

- Define connection parameters;
- Open the connection;
- Specify the statement;
- Prepare and execute the statement;
- Iteration through the results;
- Do the work for each iteration;
- Process any exception;
- Handle transactions;
- Close the connection;

With Spring support:

- Specify the statement;
- Do the work for each iteration;

Spring :: JDBC Support

Core classes for work with JDBC in Spring:

- **javax.sql.DataSource:** controls database connections
- **JdbcTemplate** is a central class that control queries execution
- **RowMapper**: controls mapping of each query row
- **JdbcDaoSupport**: facilitates configuring and transferring parameters

Spring :: javax.sql.DataSource

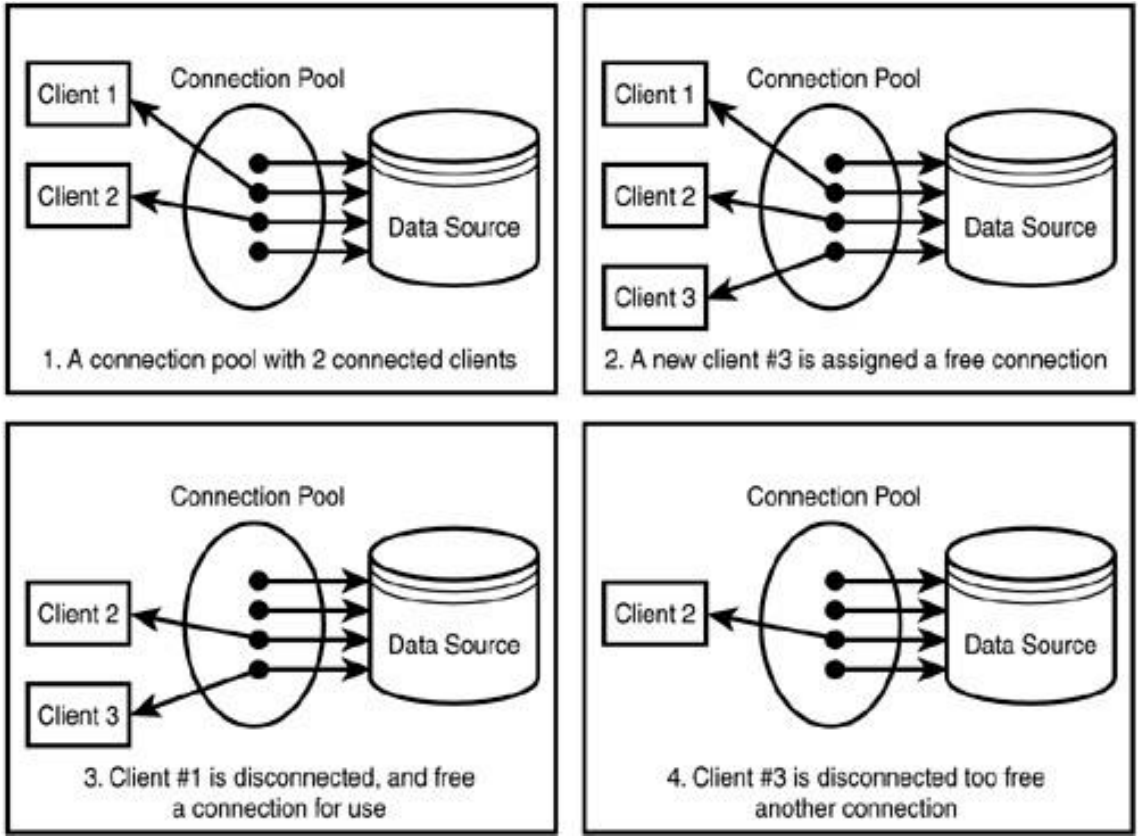
- **DataSource** interface is a part of the **JDBC** specification that can be seen as connection factory
- **Spring** connects to the database via **DataSource**
- **DataSource** allows to hide connection pooling and transaction management

Spring :: Retrieving javax.sql.DataSource

Database Connection Pool (dbcp)

- When a new user accesses the DB, it gets the connection from the pool
- Opening the connection takes the time
- If all opened connections are busy then a new connection is created
- As soon as user frees up the connection, it becomes available for other users
- If the connection is not used, it is closing

Spring :: Retrieving javax.sql.DataSource



Spring :: Configuring javax.sql.DataSource

```
<jdbc:embedded-database id="dataSource" />

<jdbc:initialize-database data-source="dataSource">
   <jdbc:script location="classpath:db-schema.sql" />
</jdbc:initialize-database>

<bean id="dataSource“ class="org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name="driverClassName" value="org.hsqldb.jdbcDriver" />
   <property name="url" value="jdbc:h2:~/book" />
   <property name="username" value="sa" />
   <property name="password" value="" />
</bean>

<bean id="bookDaoImpl" class="com.luxoft.springdb.example2.BookDaoImpl">
   <property name="dataSource" ref="dataSource" />
</bean>
```

Spring :: JdbcTemplate

JdbcTemplate is the central class in the package **org.springframework.jdbc.core**:

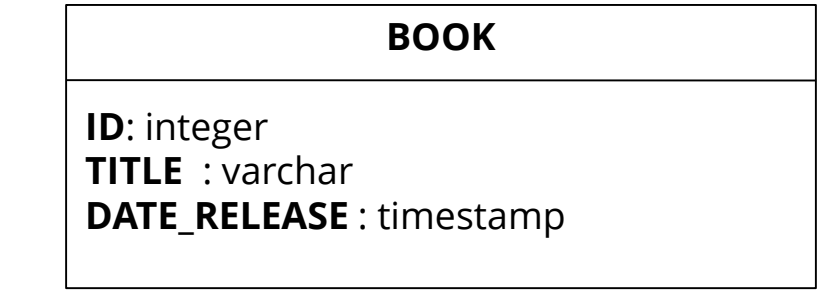
- Executes SQL queries
- Iterates over results
- Catches JDBC exceptions

Necessary parameters when executing an SQL query:

- **DataSource**
- **RowMapper**
- SQL query row

Spring :: RowMapper

Mapping data from DB to the object model




```
public class Book {
    private int id;
    private String title;
    private Date dateRelease;
```

ResultSet **Book mapRow(ResultSet rs, int rowNum)** **RowMapper**

RowMapper is doing mapping of **ResultSet** to the certain objects

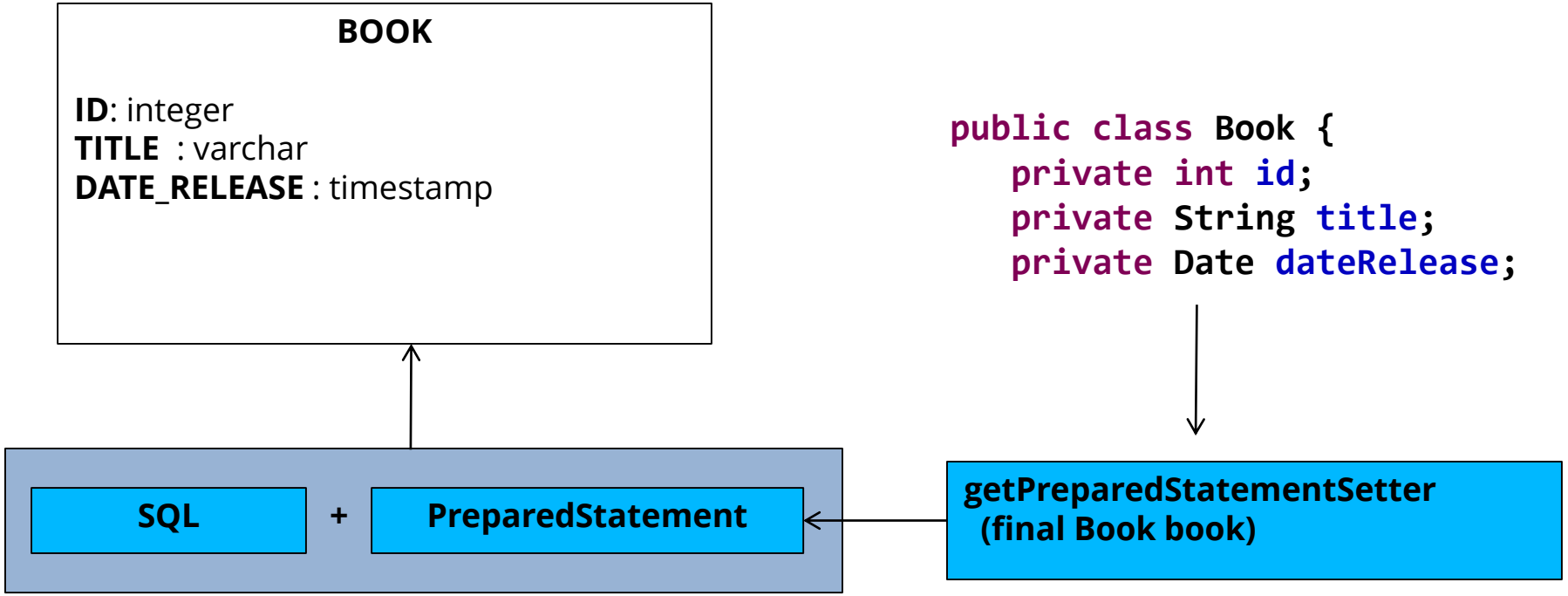
Spring :: RowMapper

```
private RowMapper<Book> rowMapper = new RowMapper<Book>() {
    public Book mapRow(ResultSet resultSet, int rowNum) throws SQLException {
        Book book = new Book();
        book.setId(resultSet.getInt("id"));
        book.setTitle(resultSet.getString("title"));
        book.setDateRelease(resultSet.getDate("date_release"));
        return book;
    }
};

@Override
public Book getById(int id) {
    String sql = "SELECT * FROM BOOK WHERE ID = ?";
    return jdbcTemplate.queryForObject(sql, rowMapper, id);
}

@Override
public List<Book> getAll() {
    return jdbcTemplate.query("SELECT * FROM BOOK", rowMapper);
}
```

ex.2

Spring :: PreparedStatementSetter

Mapping data from object model to SQL



PreparedStatementSetter is doing mapping of object to SQL request

Spring :: PreparedStatementSetter

```
private PreparedStatementSetter getPreparedStatementSetter(final Book book) {
   return new PreparedStatementSetter() {
       public void setValues(PreparedStatement preparedStatement) throws SQLException
       {
           int i = 0;
           preparedStatement.setString(++i, book.getTitle());
           preparedStatement.setDate(++i,
                           new java.sql.Date(book.getDateRelease().getTime()));
       }
   };
}

public void insert(Book book) {
   String sql = "INSERT INTO BOOK (TITLE, DATE_RELEASE) VALUES (?, ?)";
   jdbcTemplate.update(sql, getPreparedStatementSetter(book));
}
```

ex.3

Spring :: JdbcDaoSupport

- If DAO class extends **JdbcDaoSupport**, **setDataSource(..)** method will be already implemented. **JdbcDaoSupport** hides how **JdbcTemplate** is created.

```
public class BookDaoImpl extends JdbcDaoSupport implements BookDao
{

  @Override
  public void insert(Book book) {
     String sql = "INSERT INTO BOOK (TITLE, DATE_RELEASE) VALUES
     (?, ?)";
     getJdbcTemplate().update(sql,
     getPreparedStatementSetter(book));
  }
```

ex.4

Spring :: NamedParameterJdbcTemplate

- NamedParameterJdbcTemplate, configured exactly as JdbcTemplate

```
namedParameterJdbcTemplate = new NamedParameterJdbcTemplate(this.dataSource);

public Book selectBookByName(String name) {
    String sql = "SELECT * FROM BOOK WHERE TITLE = :BOOK_TITLE";
    Map<String, String> namedParameters =
          Collections.singletonMap("BOOK_TITLE", name);
    return namedParameterJdbcTemplate.queryForObject(sql,
                                        namedParameters, rowMapper);
}
```

ex.5

Spring :: JdbcTemplate :: Other SQL queries

The method `execute` from `JdbcTemplate` can be used when executing any SQL query. It is generally used for creating DDL.

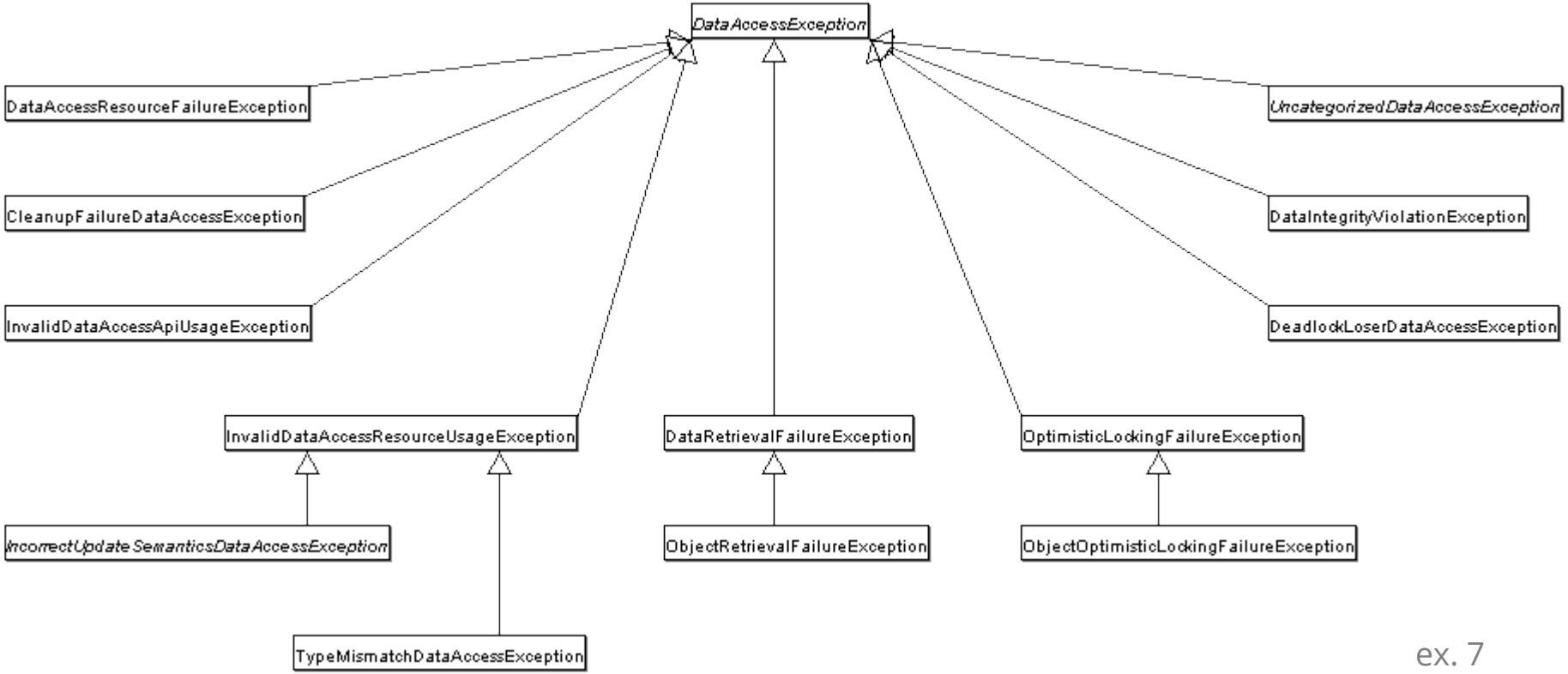
```
public void createTable() {
    String sql = "CREATE TABLE IF NOT EXISTS BOOK (ID INTEGER generated by
                  default as identity (start with 1), " +
                 "TITLE VARCHAR(50), " +
                 "DATE_RELEASE TIMESTAMP);";
    jdbcTemplate.execute(sql);
}

public void dropTable() {
    String sql = "DROP TABLE IF EXISTS BOOK;";
    jdbcTemplate.execute( sql);
}
```

ex.6

Spring :: DAO exceptions 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.

Spring :: DAO exceptions hierarchy



ex. 7

Spring :: Custom DAO exceptions translator

```
public class CustomSQLErrorCodesTranslator extends SQLErrorCodeSQLExceptionTranslator
{
    protected CustomException customTranslate(String task, String sql,
                                              SQLException sqlex) {
        if (sqlex.getErrorCode() == 42001) {
            return new CustomException(task, sql, sqlex);
        }
        return null;
    }
}

public void setDataSource(DataSource dataSource) {
     this.dataSource = dataSource;
     jdbcTemplate = new JdbcTemplate(this.dataSource);
     namedParameterJdbcTemplate = new NamedParameterJdbcTemplate(this.dataSource);
     CustomSQLErrorCodesTranslator tr = new CustomSQLErrorCodesTranslator();
     tr.setDataSource(dataSource);
     this.jdbcTemplate.setExceptionTranslator(tr);
}
```

ex.8

Exercise

Lab guide:

- Exercise 1