Spring

TECHNOLOGY

Spring Data Access

Learning Objectives

By the end of this lesson, you will be able to:

- Describe the Spring MVC framework
- Discuss the features of Spring MVC
- Define the DisptacherServlet

Learning Objectives

By the end of this lesson, you will be able to:

- Explain controllers and describe their importance
- Describe and list the ways to use RequestMapping methods
- Discuss ViewResolver and list the ViewResolver available in Spring

A Day in the Life of a Full Stack Developer

You are hired as a developer in an organization and have been assigned a project that works on transaction management. The idea is to provide a consistent abstraction for transaction management.

As a Java developer, you prefer using the Spring framework and working with Java Persistence API (JPA) and Java Transaction API (JTA).

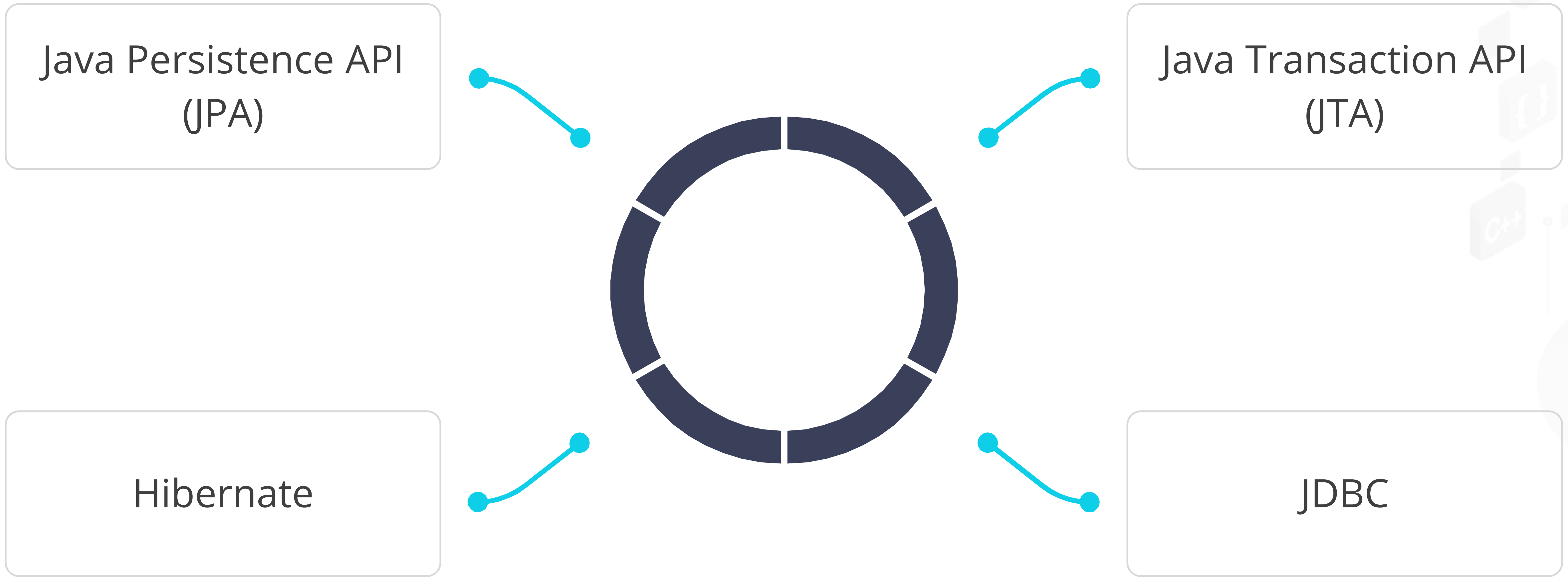
To accomplish the task, you need to explore more about the Spring MVC framework, features of MVC, DisptacherServlet, and more.

Spring and JDBC

Spring and JDBC

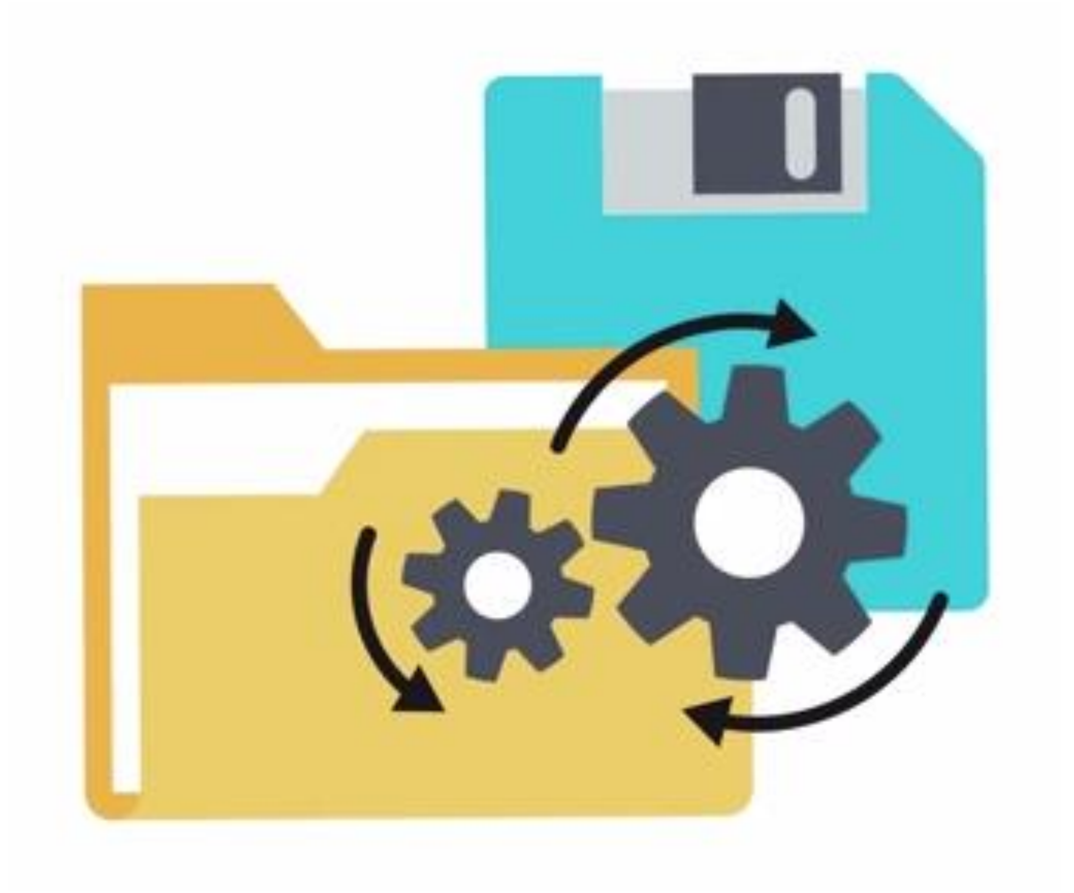
The Spring framework provides a consistent abstraction for transaction management.

This consistent programming model implements, across different transactions, APIs, such as:



Spring and JDBC

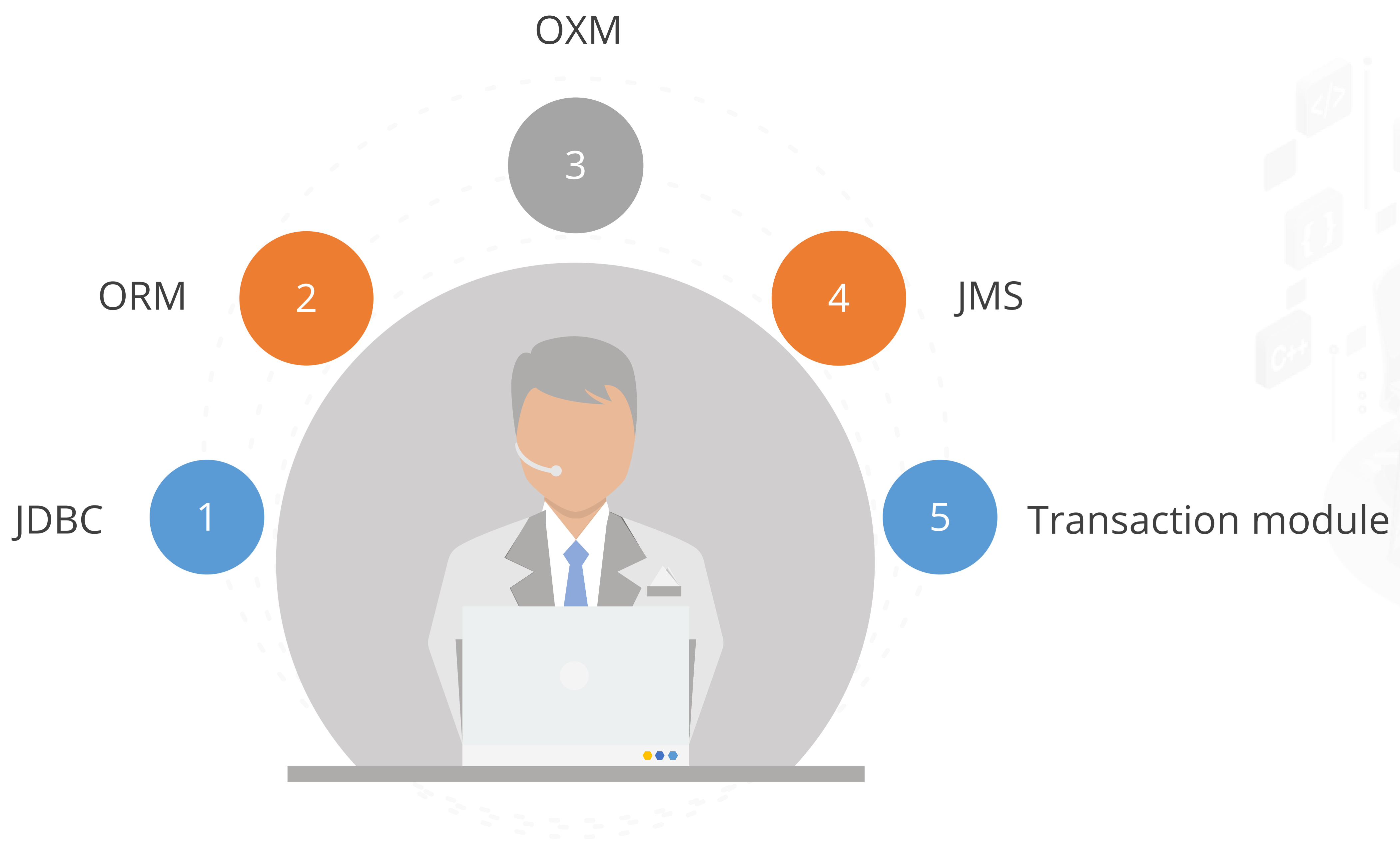
The Spring data offers a Spring-based programming model for data access.



It retains the special traits of the underlying data store.

Spring and JDBC

The data access integration layer consists of:

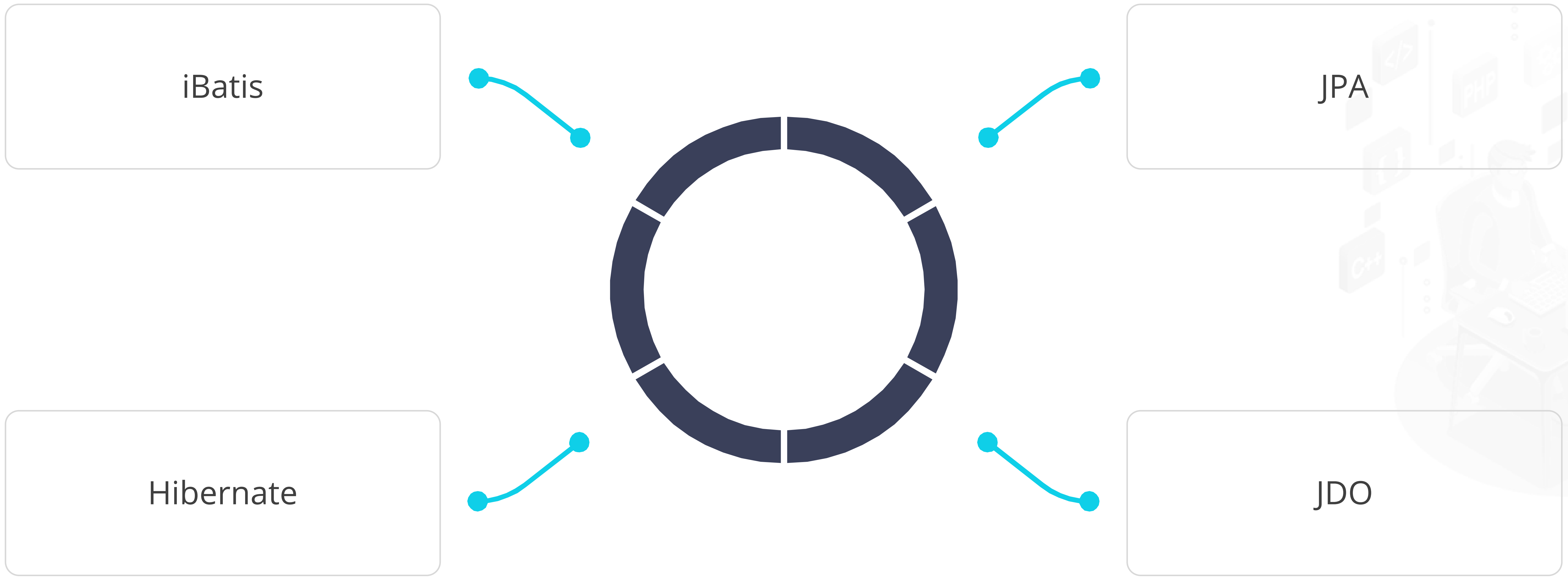


JDBC Abstraction Layer

The JDBC abstraction layer helps to remove the JDBC coding and the parsing of the database-vendor-specific error codes.

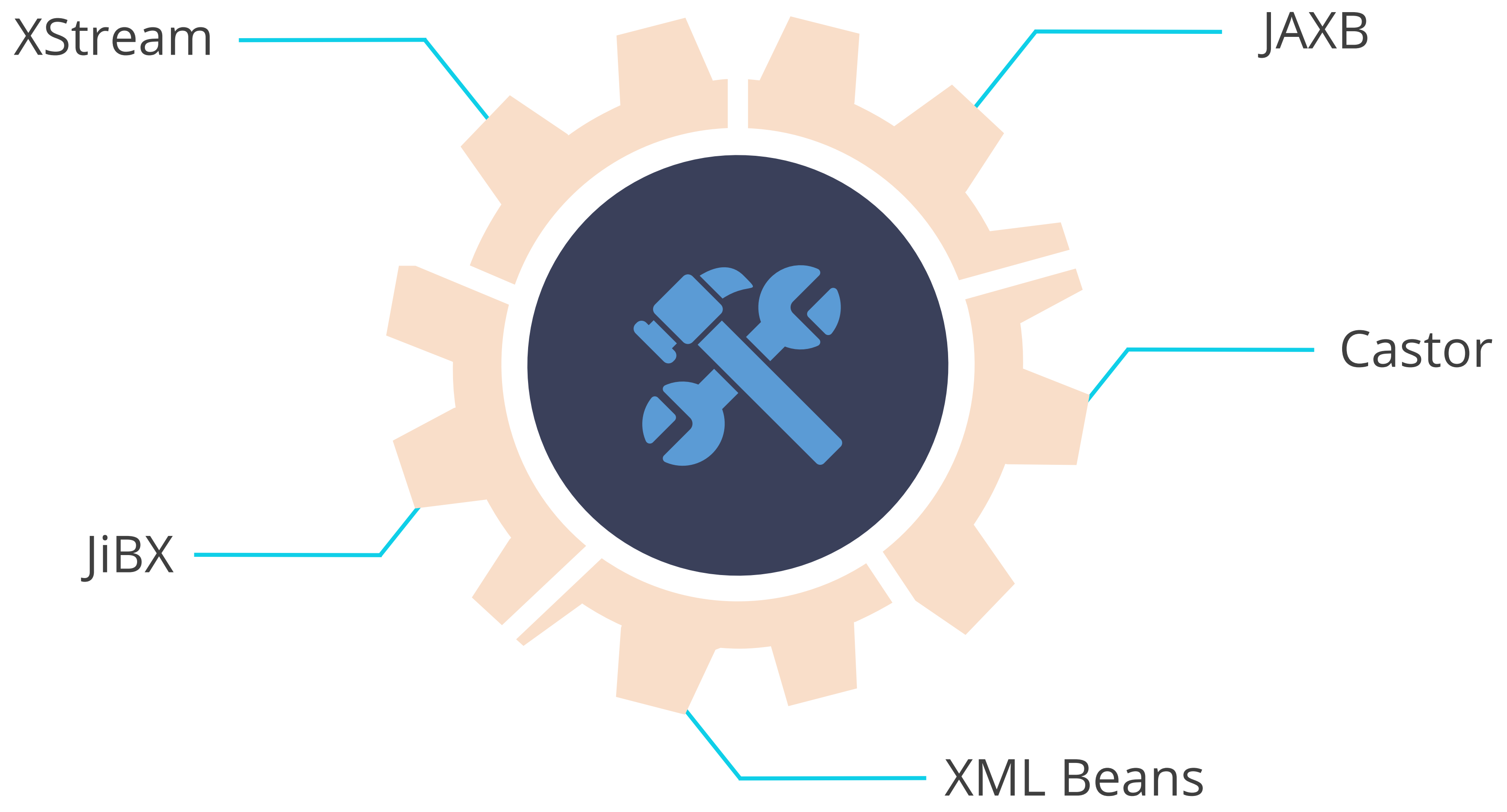
ORM Module

It contains the integration layers to facilitate object-relational mapping APIs, including:



OXM Module

It offers the abstraction layer that supports the Object/XML mapping implementations for:



JSM Module

The JMS module contains features for producing and consuming messages.

Transaction Module

The transaction module supports programmatic and declarative transaction management for classes.

Spring Data JDBC

Below are the characteristics of Spring Data JDBC. It:

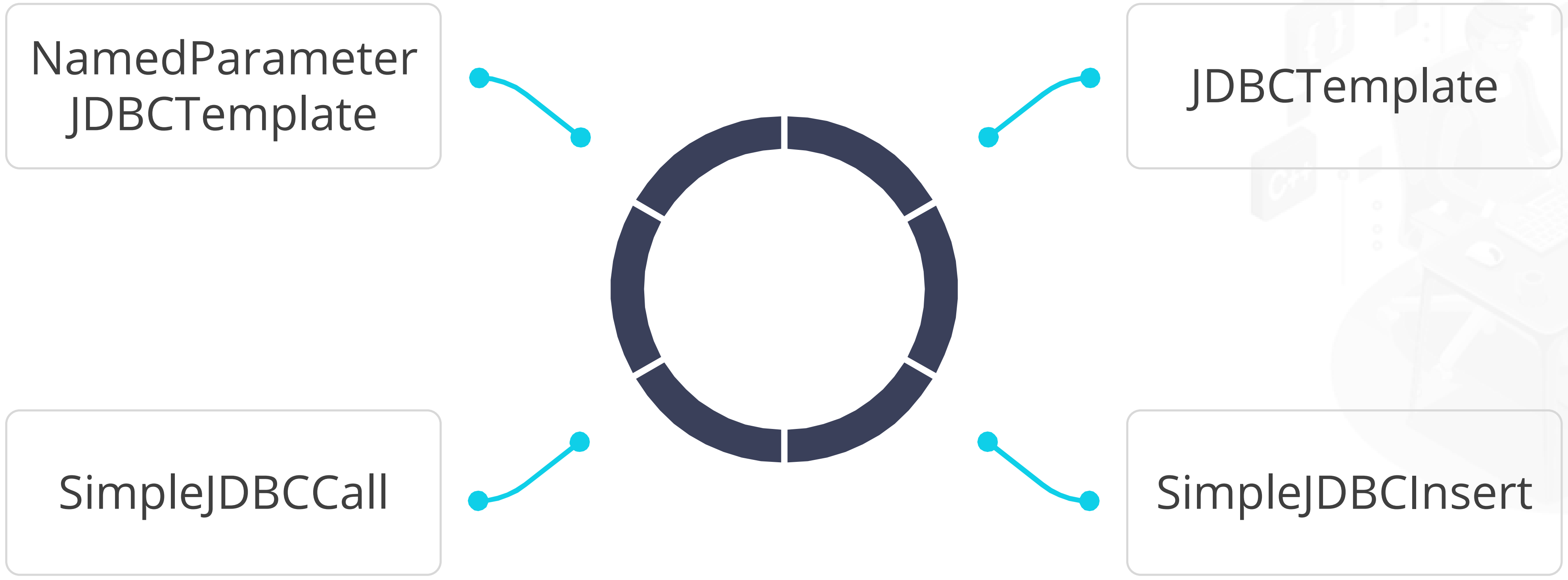
- Helps to implement JDBC-based repositories
- Deals with the enhanced support for JDBC data access layers
- Facilitates building Spring-powered applications

Spring and JDBC

Classes in the Spring JDBC are divided into four packages:

- **Core**
- Datasource
- Object
- Support

The core functionality of the JDBC and some important classes include:



Spring and JDBC

Core

Datasource

Object

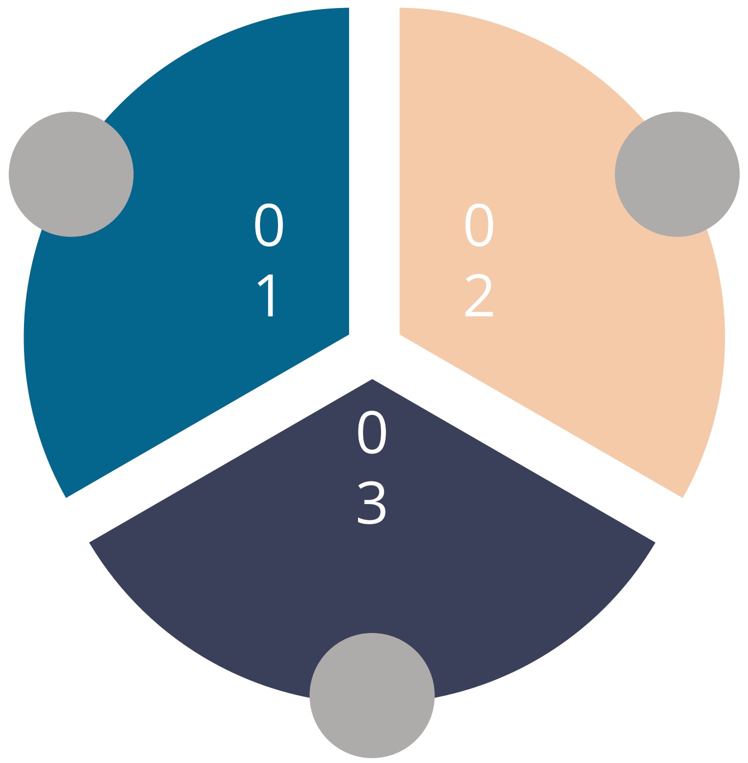
Support

Is used to access a data source

Contains various data source implementation

Spring and JDBC

- Core
- Datasource
- Object
- Support



Allows DB access to take place in an object-oriented manner

Allows to run queries and return the results

Maps the query results between the columns and the properties

Spring and JDBC

- Core
- Datasource
- Object
- **Support**

Provides support classes for the classes in the core and object packages

Spring and JDBC

A simple configuration of the data source using the MySQL database:

```
@Configuartion
@ComponentScan("..............")
Public class SpringJdbcExample {
@Bean
Public DataSource mysqlDataSource() {
DriverManagerDataSource   dataSource = new
DriverManagerDataSource();
dataSource.setDriverClassName("com.mysql.jdbc.Driver
");
dataSource.setUrl("jdbc:mysql://localhost:8000/estor
e");
dataSource.setUsername("user");
dataSource.setPassword("pwd");
Return dataSource;
}
}
```

Spring and JDBC

It creates an instance of the H2O-embedded database and pre-populates it with simple SQL scripts.

```
@Bean
Public DataSource dataSource() {
return new EmbeddedDatabaseBuilder()
.setType(EmbeddedDatabaseType.H2O)
.addScript("classpath:..........")
.addscript("classpath:...........").build();
}
```

Spring and JDBC

XML configuring for the data source:

```
<bean id="data1"
class="........................................."
destroy-method = "close">
<property name="driverClassName"
value="com.sql.jdbc.Driver"/>
<property name="url"
value=''jdbc:mysql://localhost:8000/spring/estore"/>
<property name="username" value="user">
<property name="password" value="pwd">
</bean>
```

Spring and JDBC

Problem Statement:

You have been asked to demonstrate how to use Spring JDBC to perform CRUD operations on a MySQL database.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

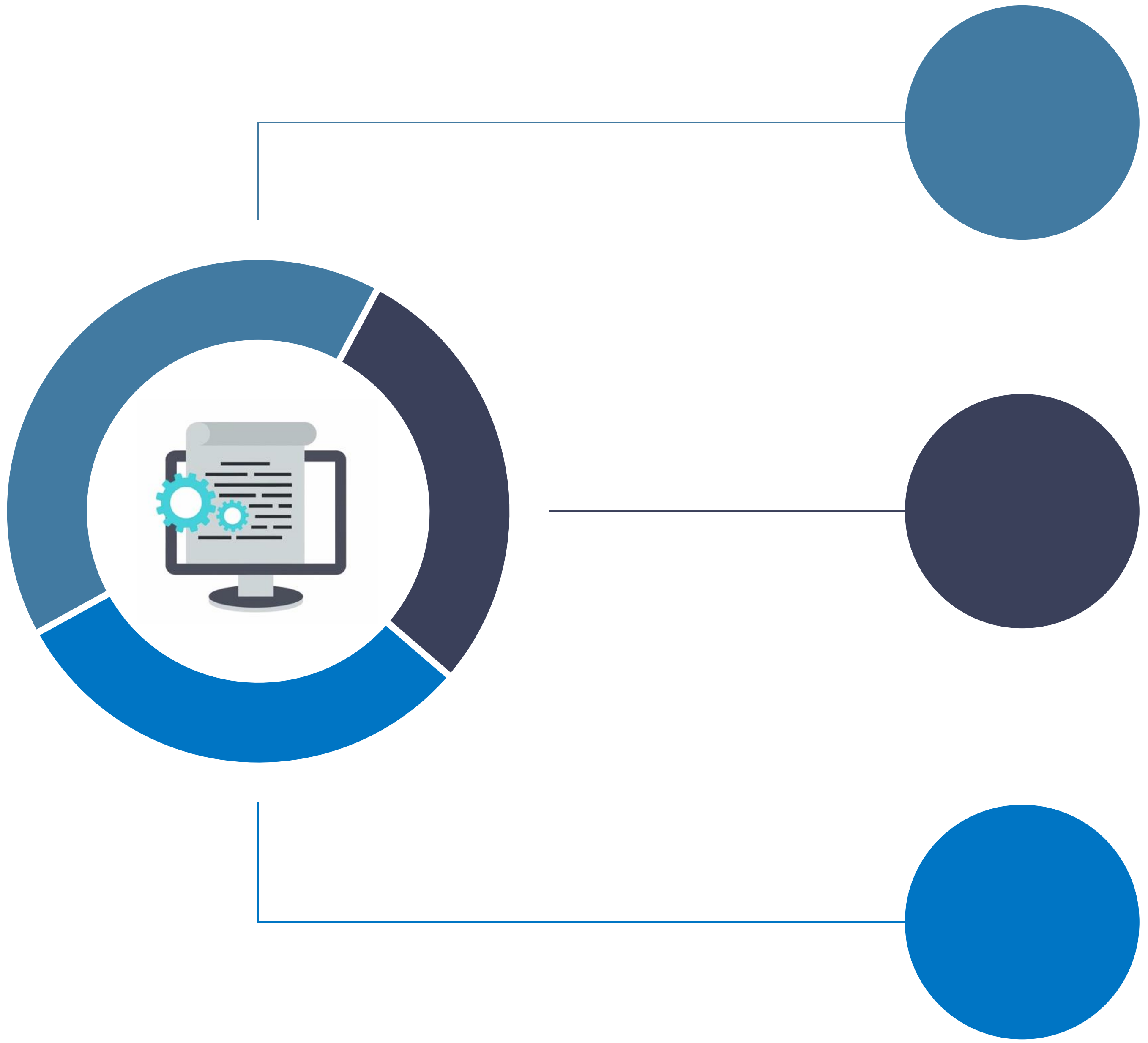
1. Setting up the Maven project and configuring the pom.xml file
2. Creating the model class for the database table
3. Configuring the MySQL database and creating the necessary table
4. Creating the DB class and configuring the JDBC template
5. Configuring the DB class in the XML file
6. Initializing the JDBC template using dependency injection
7. Performing the CRUD operations

TECHNOLOGY

JDBC Template and Runtime Queries

Basic Queries

Queries are the main API that impacts most of the functionalities.



Creating and closing the connection

Running statements and stored procedure calls

Iterating over the ResultSet and returning the results

JDBC Template and Runtime Queries

Example to see the JDBCTemplate in action:

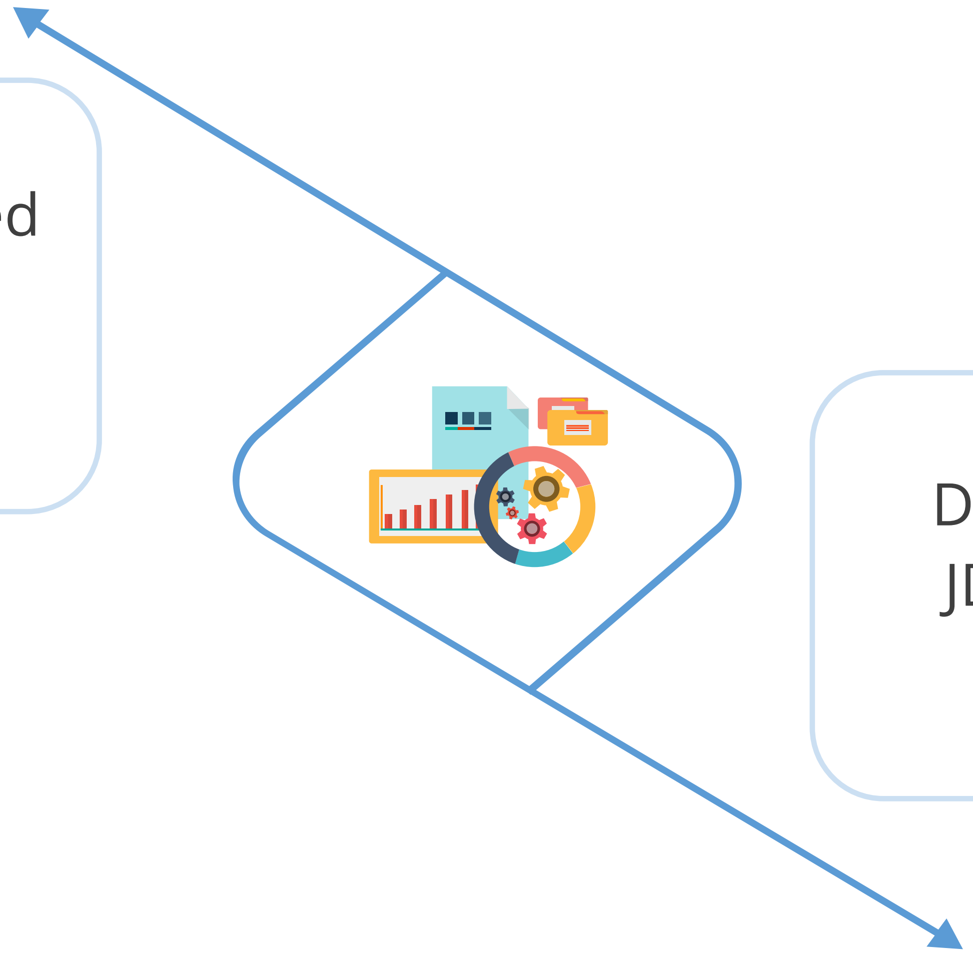
```
int  result =jdbcTemplate.queryForObject (
“SELECT COUNT(*) FROM EMPLOYEE “, Integer.class);

And now here is the simple insert

Public int addEmployee(int id) {
return jdbcTemplate.update(
“INSERT INTO EMPLOYEE VALUES ( ?, ?, ?, ? )”, id,
“Reet”, “Kaur”, “India”);
}
```

Queries with Named Parameters

Frameworks like the NamedParameterJDBCTemplate are used to support named parameters.

This wraps the JDBCTemplate and gives an alternative to the traditional syntax used to specify parameters.

Queries with Named Parameters



Helps to substitute the named parameters to the JDBC placeholder

Delegates to the wrapped JDBCTemplate to run the queries

Queries with Named Parameters

MapSqlParameterSource is employed to provide values for the named parameters.

```
SqlParameterSource namedParameters = new
MapSqlParameterSource().addValue(“id”,1);
Return namedParameterJdbcTemplate.queryForObject(
“SELECT NAME FROM EMPLOYEE WHERE ID = : id”,
namedParameters, String.class);
```

Queries with Named Parameters

Use the properties from the bean to determine the named parameters

```
Employee emp = new Employee();
emp.setName("Reet");
String SELECT_BY_ID = " SELECT COUNT(*) FROM
EMPLOYEE WHERE NAME = :Name";
SqlParameterSource namedParameters = new
BeanPropertySqlParameterSource(employee);
return namedParameterJdbcTemplate.queryForObject(
SELECT_BY_ID, namedParameters, Integer.class);
```

Note

The BeanPropertySqlParameterSource implements instead of specifying the named parameters manually.

Mapping Query Results to Java Objects

Map query results to Java objects by implementing the RowMap interface

```
Public class EmployeeRowMap implements Rowmap<Employee>  {
@Override
Public Employee mapRow(ResultSet set, introwNumber) throws SQL
Exception  {
Employee employee = new Employee();
employee.setId(set.getInt("ID"));
employee.setName(set.getString("NAME"));
employee.setCity(set.getString("CITY"));
employee.setAddress(set.getString("ADDRESS"));
return employee;
}
}
```

Mapping Query Results to Java Objects

Pass the row map to the query API and get the fully populated Java objects

```
String query = “SELECT * FROM EMPLOYEE WHERE ID =
?”;
Employee employee = jdbcTemplate.queryForObject(
query, now Object[], { id }, new EmployeeRowMap());
```

TECHNOLOGY

Exception Translation

Exception Translation

Spring has its own data exception hierarchy with the DataAccessException.

Sanity is maintained by not handling low-level persistence exceptions.

Exception Translation

- Wraps the low-level exceptions in DataAccessException
- Includes the exception-handling mechanism
- Adds implementation of SQLExceptionTranslator

Exception Translation

Use custom implementation to customize the error message resulting in the error 4044

```
public class SQLErrorExampleTranslator extends
SQLErrorCodeSQLExceptionTranslator {
@Override
Protected DataAccessException
customTranslate(String task, String sql, SQLException
sqlException) {
If (sqlException.getErrorCode() == 4044) {
return new DuplicateKeyException(
“Integrity Constraint Violation Occurred”, sqlException);
}
return null
}
}
```

Exception Translation

Assign a custom exception translator to the JDBCTemplate using the setExceptionTranslator() function

```
CustomSQLErrorCodeTranslator customSQLErrorCodeTranslator = new
CustomSQLErrorCodeTranslator();
jdbcTemplate.setExceptionTranslator(customSQLErrorCodeTranslator)
;
```

TECHNOLOGY

JDBC Operations Using the SimpleJDBC Classes

JDBC Operations Using the SimpleJDBC Classes

SimpleJDBC offers an easy way to configure and run SQL statements.

Note

SimpleJdbcInsert and the SimpleJdbcCall classes are easy ways to work with.

JDBC Operations Using the SimpleJDBC Classes

The JDBC insert statements:

Are generated based on the configuration of the SimpleJdbcInsert

Are required to add the table name, column names, and their values

JDBC Operations Using the SimpleJDBC Classes

Create the SimpleJdbcInsert class:

```
SimpleJdbcInsert simpleJdbcInsert = new Simple
JdbcInsert(dataSource).withTableName("EMPLOYEE");
```

JDBC Operations Using the SimpleJDBC Classes

Provide the column names and the values, and run the operation:

```
Public int addEmployee (Employee employee) {
Map<String, Object> parameters  = new HashMap<String, Object>();
parameters.put(“ID”, employee.getId());
parameters.put(“NAME”, employee.getName());
parameters.put(“CITY”,employee.getCity());
parameters.put(“ADDRESS”,employee.getAddress());
return simpleJdbcInsert.execute(parameters);
}
```

JDBC Operations Using the SimpleJDBC Classes

One can employ the executeAndReturnKey() API to generate the primary key.

JDBC Operations Using the SimpleJDBC Classes

It is crucial to configure the actual auto-generated column.

```
SimpleJdbcInsert  simpleJdbcInsert = new
SimpleJdbcInsert(dataSource).withTableName("EMPLOYEE
").usingGeneratedKeyColumns("ID");
Number id =
simpleJdbcInsert.executeAndReturnKey(parameters);
System.out.println(" ID is: " + id.longValue());
```

Note

Pass data by using the BeanPropertySqlParameterSource and the MapSqlParameterSource

Stored Procedure with the SimpleJdbcCall

The stored procedure uses the SimpleJdbcCall abstraction, as shown:

```
SimpleJdbcCall simplejdbcCall = new
SimpleJdbcCall(dataSource).withProcedureName(“READ_EMPLOYEE”);

Public Employee getEmployeeUsingSimpleJdbcCall(int id) {

SqlParameterSource in = new
MapSqlParameterSource().addValue(“in_id”,id);

Map<String, Object> out = simpleJdbcCall.execute(in);
Employee employee = new Employee();
employee.setName((String) out.get(“Name”));
employee.setAddress((String) out.get(“ADDRESS”));
return employee;
}
```

Batch Operations

Batch Operations

These are simple use cases that help in batching multiple operations together.

batchupdate() API

The JDBCTemplate batch operations can be run via the batchupdate() API.

Batch Operations

It involves batchpreparedstatement implementation as shown:

```
Pubic int[] batchUpadteUsingJdbcTemplate(List<Employee> employee)
{
return JdbcTemplate.batchUpdate("INSERT INTO EMPLOYEE VALUES
(?,?,?,?)", new batchPreparedStatementSetter() {
@Override
Public void setValues(PreparedStatement statement, int index )
throws SQLException {
statement.setInt(1, employee.get(index.getId());
statement.setString(2, employee.get(index).getName());
statement.setString(3, employee.get(index).getCity());
statement.setString(4,employee.get(index).getAddress();
}
@Override
Public int getBatchSize() {
return 100;
}
});
}
```

Batch Operations

The batchUpdate() API is the batching operation with the NamedParameterJdbcTemplate. It:

Is employed because this API is relatively simpler than others

Contains an internally prepared statement setter to set the parameter values

Batch Operations

The parameter values can be passed to the batchUpdate():

```
SqlParameterSource[] batch =
SqlParameteSourceUtils.createBatch(employees.toArray());
Int [] count = namedParameterJdbcTemplate.batchUpdate(
"INSERT INTO EMPLOYEE VALUES (:id, :name, :city, :address)",
batch);
return count;
```

Spring JDBC with SpringBoot

Spring JDBC with SpringBoot

It offers the starter spring-boot-starter-jdbc for using JDBC with the relational database.

Maven Dependency

It requires spring-boot-starter-jdbc dependency and also needs the dependency for the database.

```
<dependency>
<groupId>..........</groupId>
<artifactId>....................</artifactId>
</dependency>
<dependency>
<groupId>..............</groupId>
<artifactId>..................</artifactId>
<scope>...................</scope>
</dependency>
```

Configuration

Spring boot configures the data source automatically.

```
spring.datasource.url=jdbc:mysql://localhost:4400/springjdbc
spring.datasource.username=user
spring.datasource.password=pwd
```

Note

It is used to set up and run applications using only these configurations.

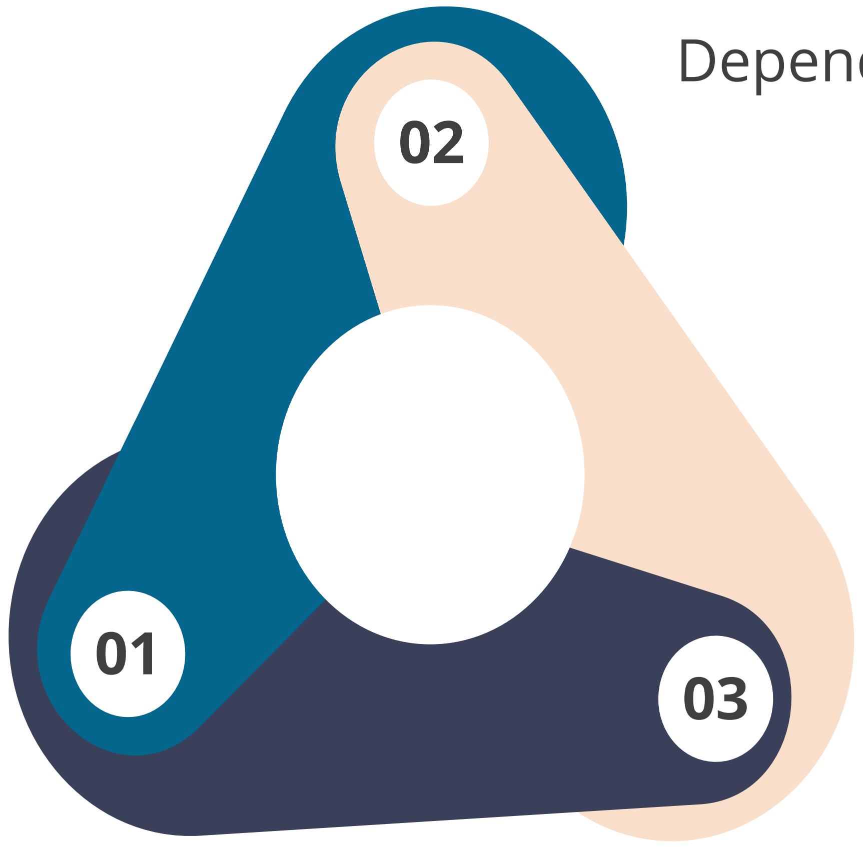
TECHNOLOGY

Spring and Hibernate

simplilearn

Spring Framework

The Spring framework helps in:



Dependency injection

Transaction management

Aspect-oriented programming

Hibernate Framework

It is used for:

Query retrieval services

Access data layers

Object-relational Persistence

Spring and Hibernate

Below is the difference between Spring and Hibernate:

Spring	Hibernate
It is an open-source framework used to develop applications.	It is a Java framework that provides object-relational mapping to an object-oriented model.
It is used to develop applications from desktop to web enterprise applications.	It offers a query retrieval service for applications for enterprise-level applications.
It provides infrastructure support to developers.	It offers object-relational mapping between Java classes and database tables.
Spring framework has no support for versioning.	Hibernate framework has versioning as an important feature.

Key Takeaways

- Spring data offers a Spring-based programming model for data access while retaining the special traits of the underlying data store.
- Spring data enables to list and describe the four packages in Spring, JDBCTemplate, and the running queries
- Spring has its data exception hierarchy with the DataAccessException as the root exception that translates all raw exceptions.
- Batch operations are simple-to-use cases that help in batching multiple operations together.

Thank You