Module 2-6

JDBC and DAO Pattern

Spring JDBC

JDBC Introduction

- JDBC stands for Java Database Connectivity, and it’s a series of specifications for allowing a Java program to interact with a database via a driver.
- Spring is a popular Java framework that implements (amongst other things) JDBC.
- In summary:
  - We use Spring JDBC, which is an implementation of JDBC, which contains JDBC Drivers, which connect to the database.

The BasicDataSource class

- The BasicDataSource class defines the database’s location and credentials.

```
BasicDataSource dataSource = new BasicDataSource();

dataSource.setUrl("jdbc:postgresql://localhost:5432/dvdstore");
dataSource.setUsername("postgres");
dataSource.setPassword("postgres1");
```

- Here we created an instance of the BasicDataSource class, and used its setters to provide the database location, username, and password.

JDBCTemplate Class (Instantiating)

The JDBCTemplate class provides the means by which a query can be made to the database and the results retrieved.

- The constructor for the JDBC Template requires that we pass in a data source object (which we talked about in the previous slide)

```
JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource)
```

JDBCTemplate Class (Sending a SQL Query)

- The .queryForRowSet(String containing SQL)method will execute the SQL query.Extra parameter constructor are available as well, allowing for any prepared statement placeholders.

```
String sqlString = “SELECT name from country”;
SqlRowSet results = jdbcTemplate.queryForRowSet(sqlString);
```

- For UPDATE, INSERT, and DELETE statements we will use the **.update** method instead of the .queryForRowSet method.

```
SqlRowSet results = jdbcTemplate.update(sqlString);
// Where sqlString contains an UPDATE, INSERT, or DELETE.
```

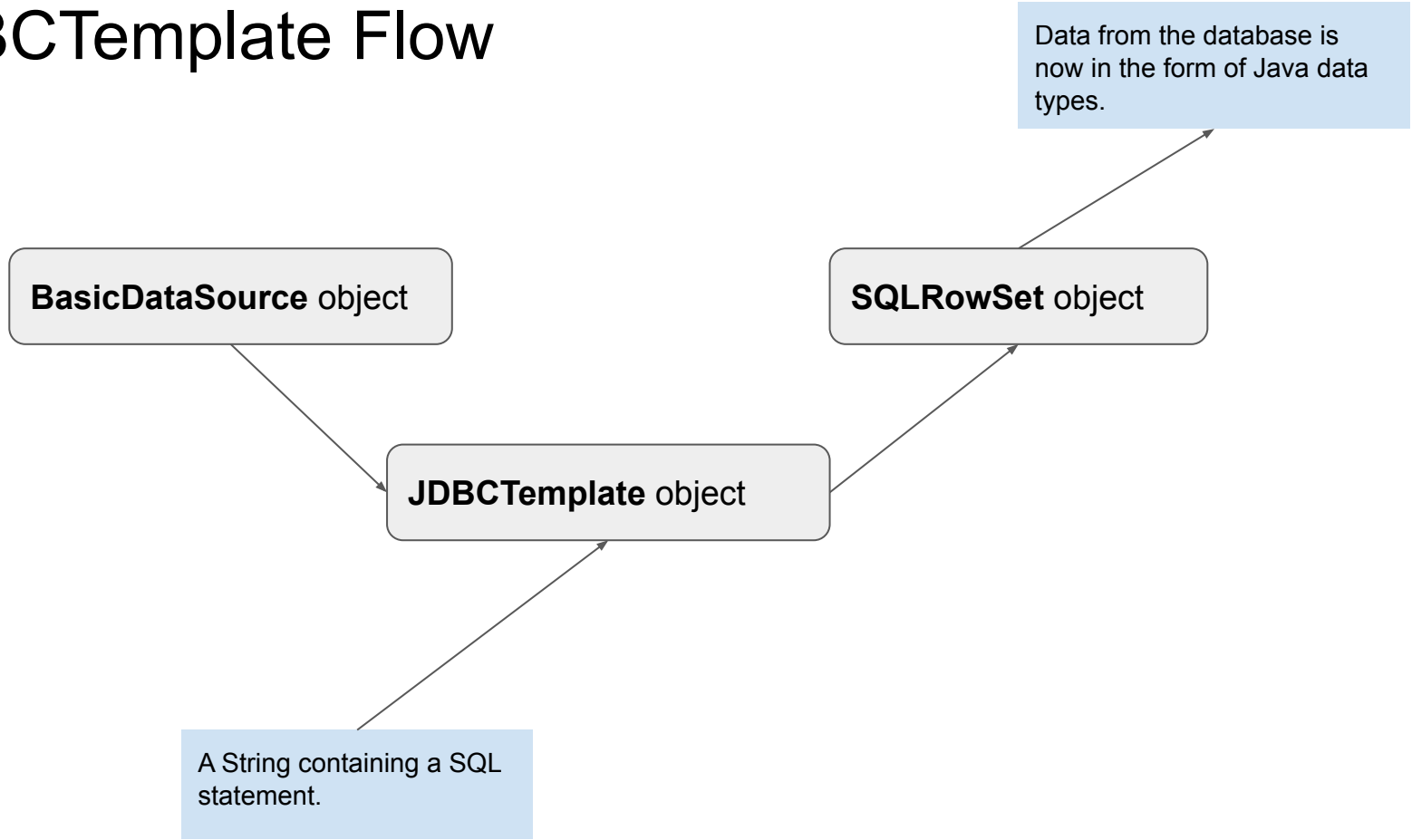
SQLRowSet Class (Accessing the Results)

The RowSET class has the following methods:

- **next()**: This methods allows for iteration if the SQL operation returns multiple rows. Using next is very similar to the way we dealt with file processing.

- **getString**(**name of column in SQL result**) , **getInt**(**name of column in SQL result**), **getBoolean**(**name of column in SQL result**) ,etc. : These get the values for a given column, for a given row.

JDBCTemplate Flow



Let’s do a quick example.

DAO Pattern

DAO Pattern

- A database table can sometimes map fully or partially to an existing class in Java. This is known as **Object-Relational Mapping**.

- We implement Object Relation Mapping with a design pattern called DAO, which is short for **Data Access Object**.

- We do this in a very specific way using Interfaces so that future changes to our data infrastructure (i.e. migrating from 1 database platform to another) are easier to manage.

The Goal

We will query from a database, and use the data from each row to create a “City” object:

*	id	name	countrycode	district	population
1	1	Kabul	AFG	Kabol	1780000
2	2	Qandahar	AFG	Qandahar	237500
3	3	Herat	AFG	Herat	186800
4	4	Mazar-e-Sharif	AFG	Balkh	127800
5	5	Amsterdam	NLD	Noord-Holland	731200
6	6	Rotterdam	NLD	Zuid-Holland	593321
7	7	Haag	NLD	Zuid-Holland	440900
8	8	Utrecht	NLD	Utrecht	234323

A City object:

id=1
name= “Kabul”
countrycode= “AFG”
...

Another City object:

id=5
name= “Amsterdam”
countrycode= “NLD”
...

DAO Pattern (Setup)

First, we have a class in Java that corresponds to the columns being retrieved from the database:

```
public class City {
        private Long id;
        private String name;
        private String countryCode;
        private String district;
        private int population;
// + getters
}
```

Table: city

world/world/public/TABLE/city

Info | Columns | Data | Row Count | Primary Key | Indexes | Grants | Row Id

TABLE_CAT	TABLE...	TABLE...	COLUMN_NAME	DATA_TYPE	TYPE_NAME	COLUMN_SIZE
(null)	public	city	id	4	serial	10
(null)	public	city	name	12	varchar	64
(null)	public	city	countrycode	1	bpchar	3
(null)	public	city	district	12	varchar	64
(null)	public	city	population	4	int4	10

DAO Pattern Step 1

- We start off with an Interface specifying that a class that chooses to implement the interface must implement methods to communicate with a database (i.e. search, update, delete). Consider the following example:

```
public interface CityDAO {

        public void save(City newCity);
        public City findCityById(long id);
}
```

DAO Pattern Step 2

- Next, we want to go ahead and create a concrete class that implements the interface:

DAO Pattern Step 2

```
public class JDBCCityDAO implements CityDAO {

	private JdbcTemplate jdbcTemplate;

	public JDBCCityDAO(DataSource dataSource) {
		this.jdbcTemplate = new JdbcTemplate(dataSource);
	}

	@Override
	public void save(City newCity) {
		String sqlInsertCity = "INSERT INTO city(id, name, countrycode, district, population) " +
								"VALUES(?, ?, ?, ?, ?)";
		newCity.setId(getNextCityId());
		jdbcTemplate.update(sqlInsertCity, newCity.getId(), newCity.getName(),newCity.getCountryCode(), newCity.getDistrict(),newCity.getPopulation());
	}

	@Override
	public City findCityById(long id) {
		City theCity = null;
		String sqlFindCityById = "SELECT id, name, countrycode, district, population "+
								"FROM city "+
								"WHERE id = ?";
		SqlRowSet results = jdbcTemplate.queryForRowSet(sqlFindCityById, id);
		if(results.next()) {
			theCity = mapRowToCity(results);
		}
		return theCity;
	}
```

The contractual obligations of the interface are met.

Let’s implement a DAO class

DAO Pattern Step 3

- In our orchestrator class, we will be using polymorphism to declare our DAO objects:

```
CityDAO dao = new JDBCCityDAO(worldDataSource);
```

The Interface Reference

The Concrete Class Constructor

DAO Pattern Step 3

- We can now call the DAO methods we declared to interact with the database:

```
City smallville = new City();
smallville.setCountryCode("USA");
smallville.setDistrict("KS");
smallville.setName("Smallville");
smallville.setPopulation(42080);

dao.save(smallville);

City theCity = dao.findCityById(smallville.getId());
```

We can now call the methods that are defined in concrete class and required by the interface.

Let’s now use the DAO class