JAVA BACKEND DEVELOPMENT PROGRAM

Spring Data Access

OUTLINE

- Spring Boot
    - Introduction
    - Auto-configuration
- JDBC
- JdbcTemplate

SPRING BOOT

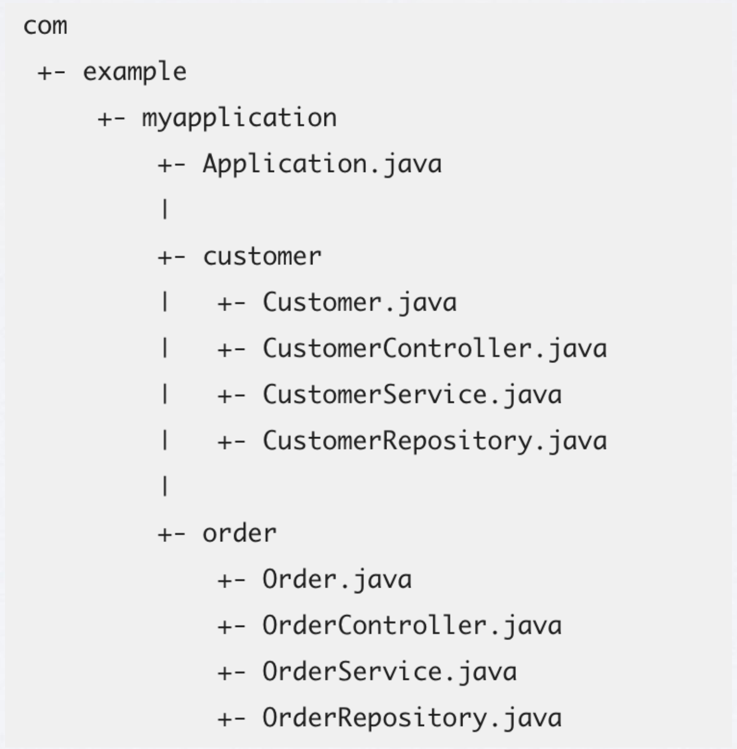
- Spring Boot makes it easy to create stand-alone, production-grade Spring-based Applications that you can just run.
- Benefits:
  - It's easy to develop Spring based application
  - Spring Boot takes less time, improve overall productivity
  - No need to write templet code, XML configuration or redundant annotation
  - Easy to integrate with other Spring feature such as Spring JDBC, Spring ORM, Spring AOP, Spring Security and etc
  - Provides embedded HTTP server such as Tomcat, easy for web app development and test

SPRING BOOT

- How does it work?
  - Spring Boot automatically configures your application based on the dependencies you have added to the project by using @EnableAutoConfiguration annotation.
  - The entry point of the spring boot application is the class contains @SpringBootApplication annotation and the main method.
    - @EnableAutoConfiguration: enable Spring Boot's auto-configuration mechanism
    - @ComponentScan: enable @Component scan on the package where the application is located
    - @Configuration: allow to register extra beans in the context or import additional configuration classes

SPRING BOOT

- Auto-Configuration
  - The @SpringBootApplication annotation is often placed on your main class, and it implicitly defines a base "search package" for certain items



```
com
 +- example
     +- myapplication
         +- Application.java
         |
         +- customer
         |   +- Customer.java
         |   +- CustomerController.java
         |   +- CustomerService.java
         |   +- CustomerRepository.java
         |
         +- order
             +- Order.java
             +- OrderController.java
             +- OrderService.java
             +- OrderRepository.java
```

SPRING BOOT

- Auto-Configuration
  - Spring Boot auto-configuration attempts to automatically configure your Spring application based on the jar dependencies that you have added
  - Gradually Replacing Auto-configuration
    - Auto-configuration is non-invasive — For example, if you add your own DataSource bean, the default embedded database support backs away
  - Disabling Specific Auto-configuration Classes
    - If you find that specific auto-configuration classes that you do not want are being applied, you can use the exclude attribute of @EnableAutoConfiguration to disable them

```
@Configuration(proxyBeanMethods = false)
@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})
public class MyConfiguration {
}
```

    - Here is the link for all auto-configuration classes: https://docs.spring.io/spring-boot/docs/2.2.3.RELEASE/reference/html/appendix-auto-configuration-classes.html#auto-configuration-classes

OUTLINE

- Spring Boot
  - Introduction
  - Auto-configuration
- JDBC
- JdbcTemplate

INTRODUCTION TO JDBC

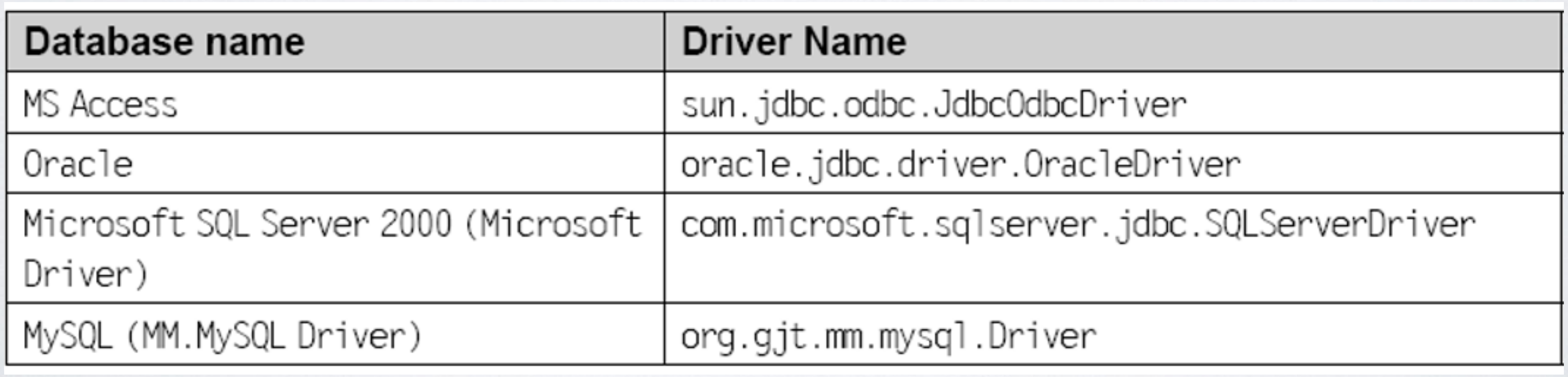
- JDBC stands for "Java Database Connectivity".
- It is an API (Application Programming Interface) which consists of a set of Java classes, interfaces and exceptions and a specification to which both JDBC driver vendors and JDBC developers (like us) adhere when developing applications.

HOW TO USE JDBC

- JDBC defines a few steps to connect to a database and retrieve/insert/update databases.
  - Load the driver (optional after JDBC 4.0)
  - Establish connection
  - Create statements
  - Execute query and obtain result
  - Iterate through the results
  - Close connection

LOAD THE DRIVER

- The driver is loaded with the help of a static method,
  - Class.forName(drivername)
- Every database has its own driver.

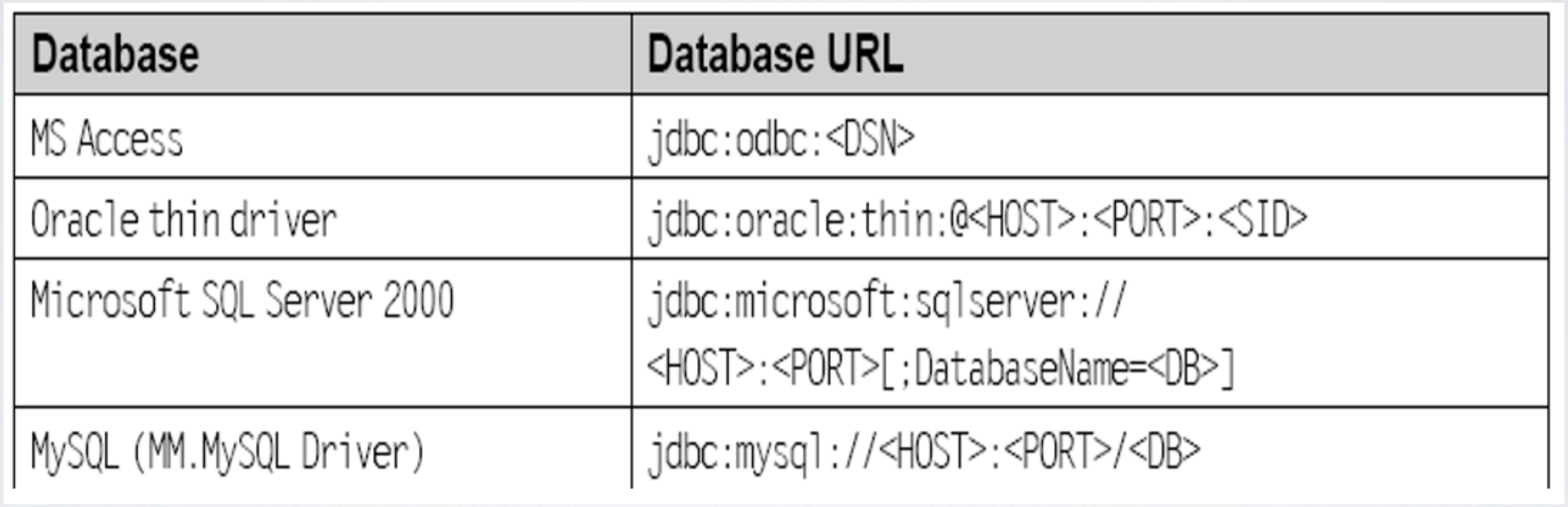


Database name	Driver Name
MS Access	sun.jdbc.odbc.JdbcOdbcDriver
Oracle	oracle.jdbc.driver.OracleDriver
Microsoft SQL Server 2000 (Microsoft Driver)	com.microsoft.sqlserver.jdbc.SQLServerDriver
MySQL (MM.MySQL Driver)	org.gjt.mm.mysql.Driver

ESTABLISH A CONNECTION

- A connection to the database is established using the static method *getConnection(databaseUrl)* of the DriverManager class.
- The DriverManager class is class for managing JDBC drivers.
- The database URL takes the following shape jdbc:subprotocol:subname.
- If any problem occurs during accessing the database, an SQLException is generated, else a Connection object is returned which refers to a connection to a database.
- Connection is actually an interface in java.sql package.
    - *Connection con=DriverManager.getConnection(databaseUrl);*

FEW DATABASE URLS



Database	Database URL
MS Access	jdbc:odbc:<DSN>
Oracle thin driver	jdbc:oracle:thin:@<HOST>:<PORT>:<SID>
Microsoft SQL Server 2000	jdbc:microsoft:sqlserver://<HOST>:<PORT>[;DatabaseName=<DB>]
MySQL (MM.MySQL Driver)	jdbc:mysql://<HOST>:<PORT>/<DB>

CREATE STATEMENT

- The connection is used to send SQL statements to the database.
- Three interfaces are used for sending SQL statements to databases
  - Statement
  - PreparedStatement
  - Callable Statement
- Three methods of the Connection object are used to return objects of these three statements.
  - conn.createStatement() => Statement Object
  - conn.preparedStatement() => PreparedStatement Object
  - conn.prepareCall() => CallableStatement Object

CREATE STATEMENT

- Statement
    - A Statement object is used to send a simple SQL statement to the database with no parameters
    - If n rows need to be inserted, then the same statement gets compiled n number of times
- PreparedStatement
    - A PreparedStatement object sends precompiled statements to the databases with or without IN parameters
    - Only the values that have to be inserted are sent to the database again and again
    - Increase efficiency
- CallableStatement
    - A CallableStatement object is used to call stored procedures
    - Better performance due to pre-compilation

EXECUTE QUERY

- Three methods are used
  - ResultSet executeQuery(String sqlQuery) throws SQLException
  - int executeUpdate(String sqlQuery) throws SQLException
  - boolean execute(String sqlQuery) throw SQLException

EXECUTE QUERY

- boolean execute(String sqlQuery)
  - returns boolean value
  - indicates if an ResultSet object is available for the given query.

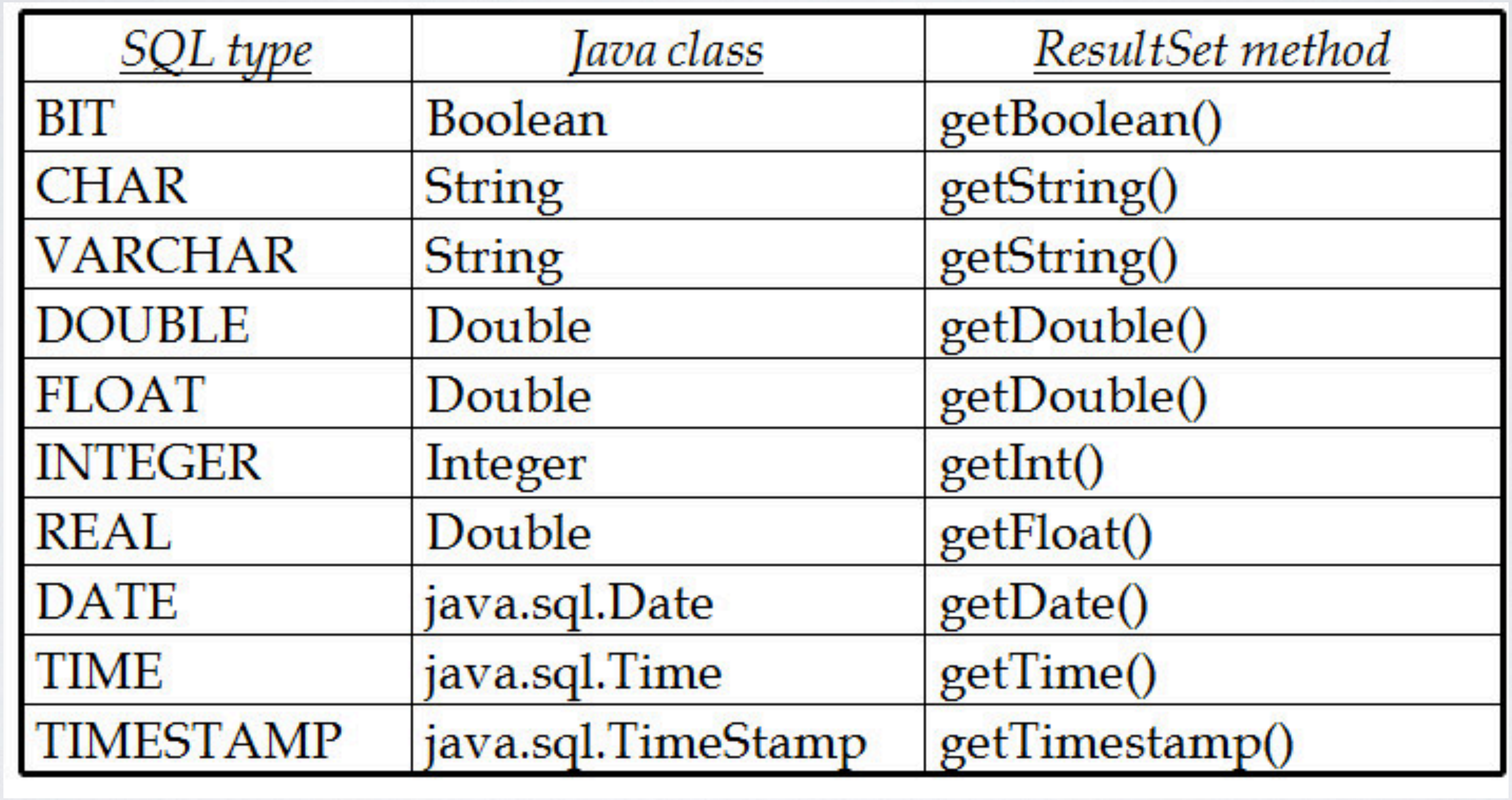
EXECUTE QUERY

- int executeUpdate(String sqlQuery)
  - for DDL & DML
  - returns an integer value
  - indicates the number of rows affected by the given query
  - returns 0 for DDL statements which do not return anything

EXECUTE QUERY

- ResultSet executeQuery(String sqlQuery)
  - returns an ResultSet object which contains the result of the given query

MAPPING TYPES JDBC - JAVA



SQL type	*Java class*	*ResultSet method*
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

CLOSE CONNECTION

- Just like file I/O, we have to close the database connection after we finish all the jobs.
    - *statement.close()*
    - *connection.close()*

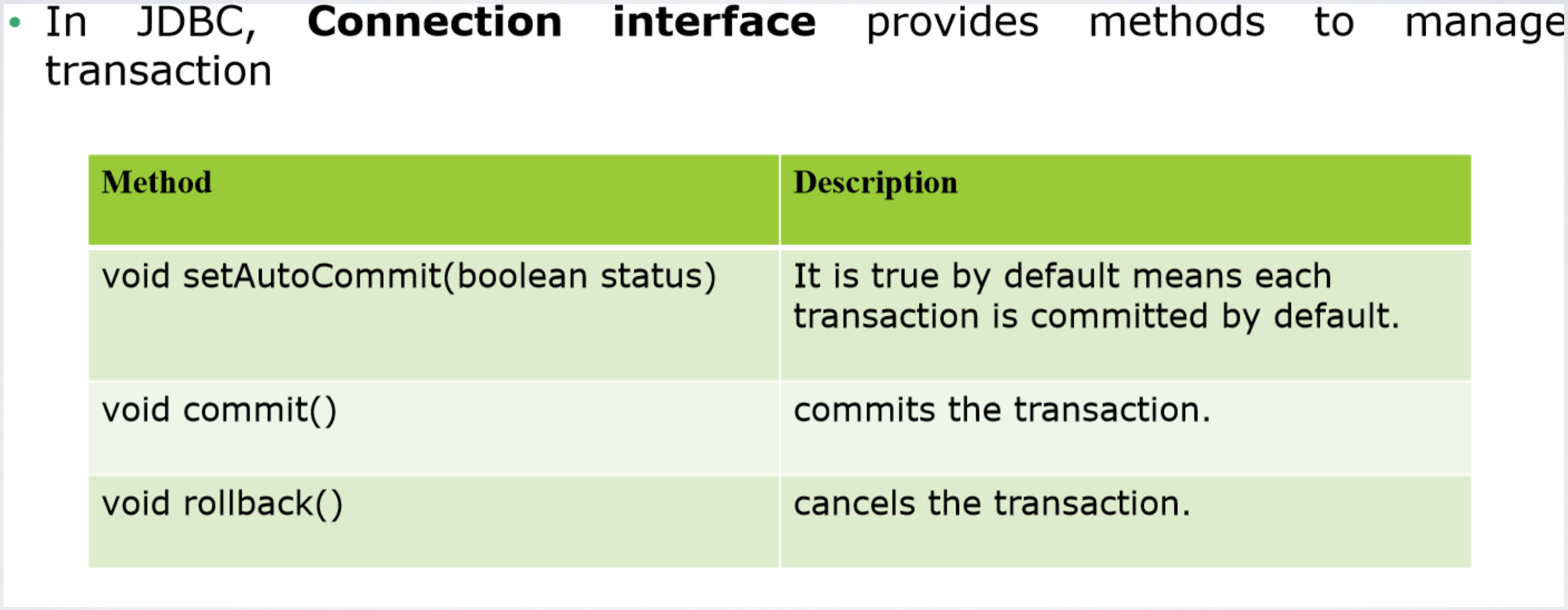
TRANSACTION MANAGEMENT IN JDBC

- Transaction represents a single unit of work.
- The ACID properties describes the transaction management well.
  - **A**tomicity: means either all successful or none.
  - **C**onsistency: ensures bringing the database from one consistent state to another consistent state.
  - **I**solation: ensures that transaction is isolated from other transaction.
  - **D**urability: means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

TRANSACTION MANAGEMENT IN JDBC

- JDBC allows SQL statements to be grouped together into a single transaction
- Transaction control is performed by the Connection object,
  - default mode is auto-commit, I.e., each sql statement is treated as a transaction
- We can turn off the auto-commit mode with con.setAutoCommit(false);
- And turn it back on with con.setAutoCommit(true);
- Once auto-commit is off, no SQL statement will be committed until an explicit is invoked con.commit();
- At this point all changes done by the SQL statements will be made permanent in the database.

TRANSACTION MANAGEMENT IN JDBC



- In JDBC, **Connection interface** provides methods to manage transaction

Method	**Description**
void setAutoCommit(boolean status)	It is true by default means each transaction is committed by default.
void commit()	commits the transaction.
void rollback()	cancels the transaction.

OUTLINE

- Spring Boot
    - Introduction
    - Auto-configuration
- JDBC
- JdbcTemplate

SPRINGBOOT SUPPORT

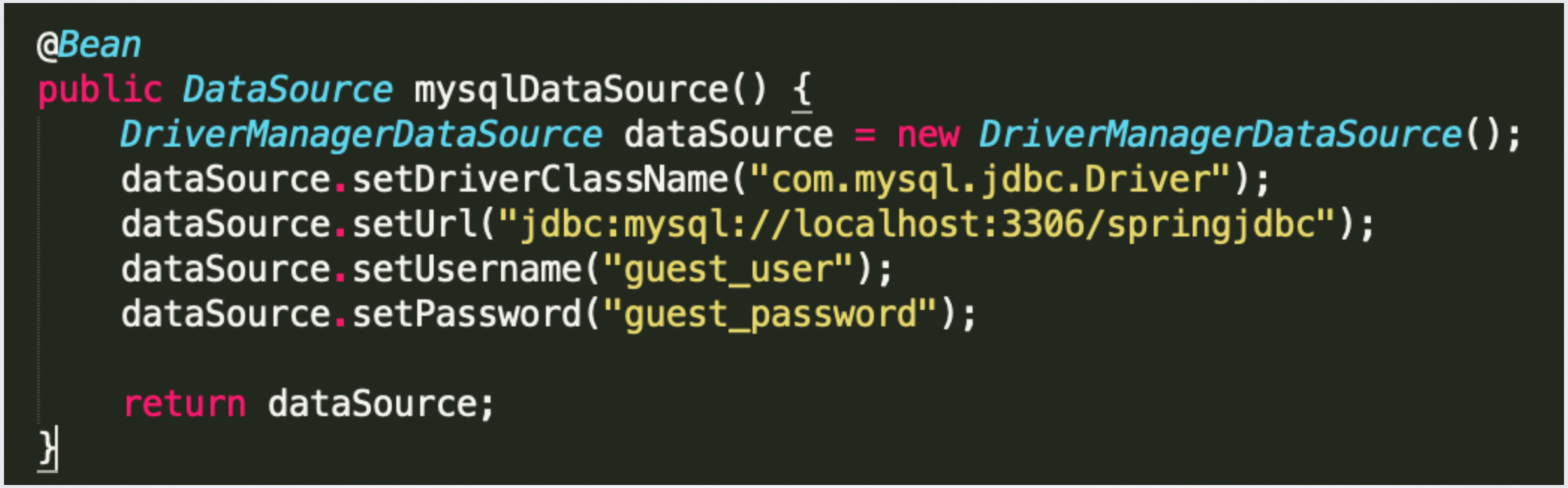
- SpringBoot is aimed at making it east to work with data access technologies
  - JdbcTemplate
  - HibernateTemplate
  - JpaRepository

JDBCTEMPLATE

Action	Spring	You
Define connection parameters.		X
Open the connection.	X	
Specify the SQL statement.		X
Declare parameters and provide parameter values		X
Prepare and execute the statement.	X	
Set up the loop to iterate through the results (if any).	X	
Do the work for each iteration.		X
Process any exception.	X	
Handle transactions.	X	
Close the connection, the statement, and the resultset.	X	

CONNECTION CONFIGURATION

- Spring obtains a connection to the database through a DataSource



```java
@Bean
public DataSource mysqlDataSource() {
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName("com.mysql.jdbc.Driver");
    dataSource.setUrl("jdbc:mysql://localhost:3306/springjdbc");
    dataSource.setUsername("guest_user");
    dataSource.setPassword("guest_password");

    return dataSource;
}
```

JDBC TEMPLATE

- Let’s take a look at Spring JdbcTemplate Example
  - Single Object — *queryForObject*
  - List Object — *query*
  - RowMapper — *mapping to domain*
  - Named Parameters — *IN Clause*

ANY QUESTIONS?