rgupta.mtech@gmail.com

Pain of JDBC

1. Define the connection parameters.
2. Access a data source, and establish a connection.
3. Begin a transaction.
4. Specify the SQL statement.
5. Declare the parameters, and provide parameter values.
6. Prepare and execute the statement.
7. Set up the loop to iterate through the results.
8. Do the work for each iteration--execute the business logic.
9. Process any exception.
10. Commit or roll back the transaction.
11. Close the connection, statement, and resultset.

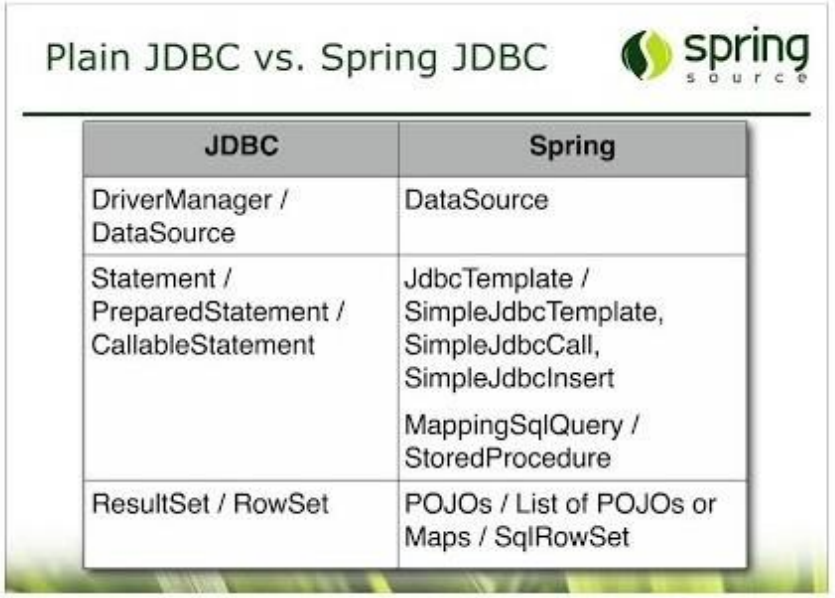
Pain of JDBC

```
    Connection connection=null;
    Statement stmt =null;
    ResultSet rs =null;
    try {
         connection = dataSource.getConnection();
         stmt = connection.createStatement();
         rs = stmt.executeQuery("select * from account");
        while (rs.next()) {
            accounts.add(new Account(rs.getInt("id"), rs.getString("name"), rs.getDouble("balance")));
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }finally {
        if(stmt!=null) {
            try {
                stmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(rs!=null) {
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }if(connection!=null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    return accounts;
}
```

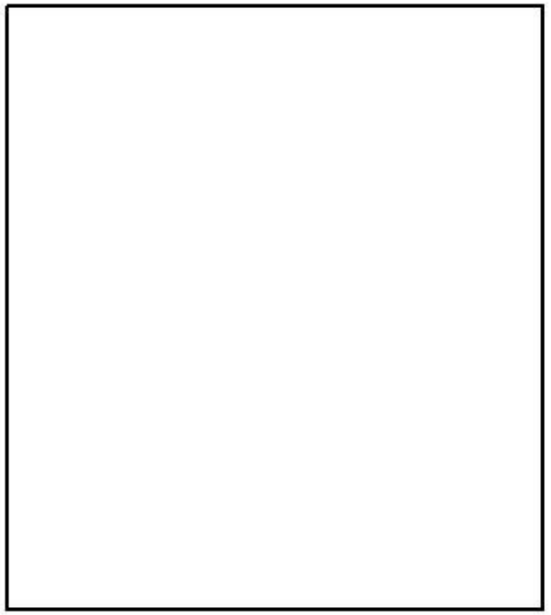
What is boilerplate code

In computer programming, **boilerplate code** or **boilerplate** refers to sections of code that have to be included in many places with little or no alteration. It is often used when referring to languages that are considered *verbose*, i.e. the programmer must write a lot of code to do minimal jobs.

For instance, a lawyer may give you a five page contract to sign, but most of the contract is boilerplate — meaning it's the same for everyone who gets that contract, with only a few lines changed here and there.



JDBC	Spring
DriverManager / DataSource	DataSource
Statement / PreparedStatement / CallableStatement	JdbcTemplate / SimpleJdbcTemplate, SimpleJdbcCall, SimpleJdbcInsert MappingSqlQuery / StoredProcedure
ResultSet / RowSet	POJOs / List of POJOs or Maps / SqlRowSet



Training Evaluation Form

Date of Presentation:

Presenter's Name:

Topic or Session:

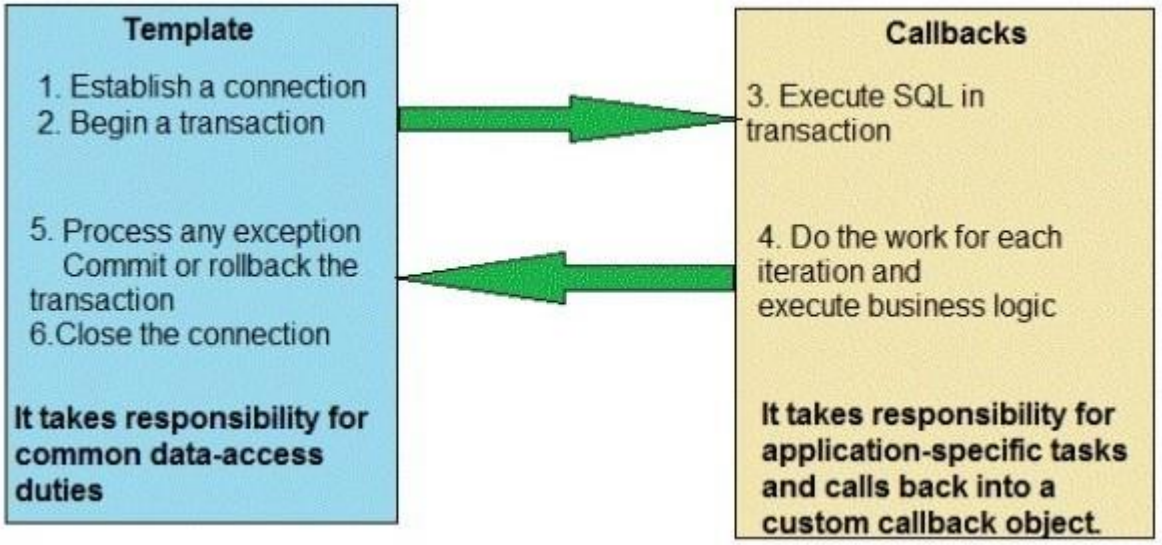
Please complete the evaluation for today's training session – your feedback is valuable
AusDBF is committed to continual improvement and suggestions will be considered

Criteria	Strongly agree 4	Agree 3	Disagree 2
Training was relevant to my needs			
Materials provided were helpful			
Length of training was sufficient			
Content was well organised			
Questions were encouraged			
Instructions were clear and understandable			
Training met my expectations			
The presenter and / or presentation was effective			

Which one is easy for you to provide feedback?

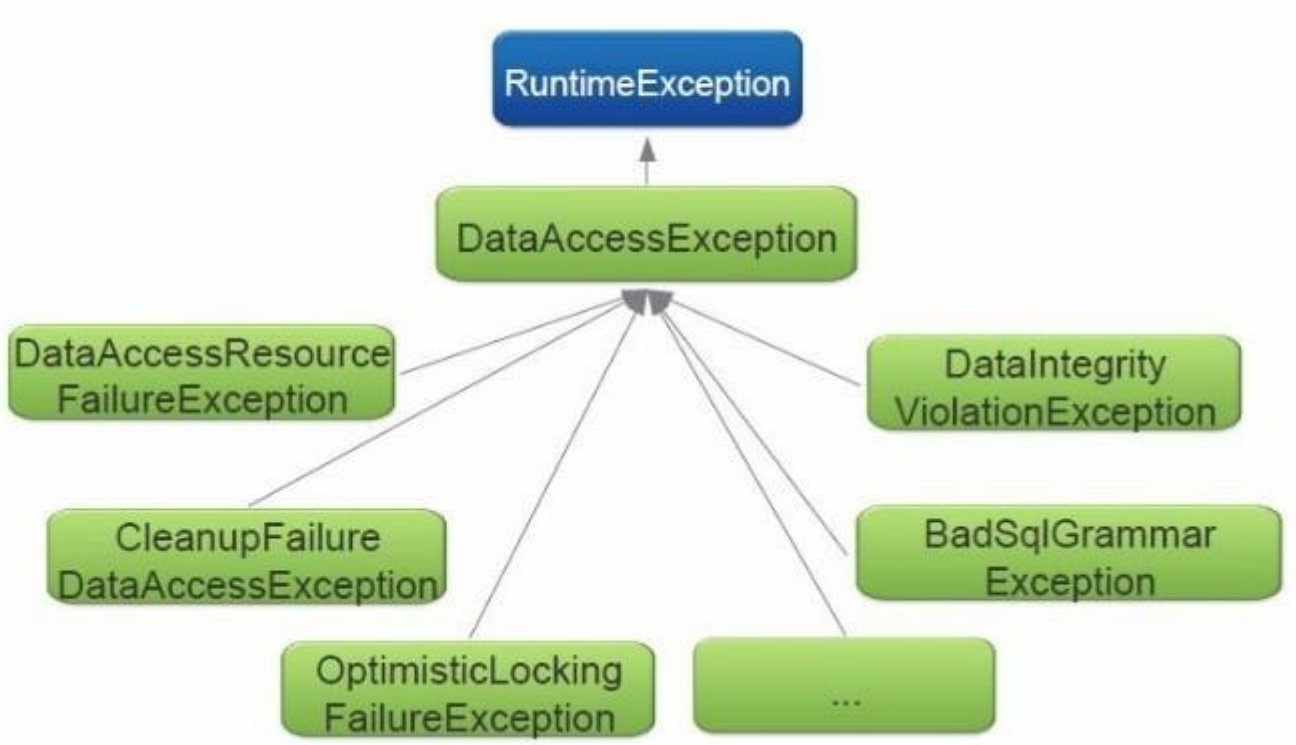
```
@Override
public void save(Account account) {
    String sql="insert into account(id, name, balance) values (?,?,?)";
    jdbcTemplate=new JdbcTemplate(dataSource);
    jdbcTemplate.update(sql, new Object[] {account.getId(),
            account.getName(), account.getBalance()});
}
```

Spring jdbc template



Exception handling

Access Exception hierarchy:



Spring jdbc configuration

```
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/b
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.3.xsd

    <context:component-scan base-package="com.bankapp.*"/>
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="${jdbc.driverName}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>

    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"></property>
    </bean>

    <context:property-placeholder location="classpath:db.properties"/>
    <tx:annotation-driven transaction-manager="transactionManager"/>
</beans>
```

```java
@Override
public List<Account> getAllAccounts() {
    List<Account> accounts = new ArrayList<Account>();
    Connection connection=null;
    try {
         connection = dataSource.getConnection();

        Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery("select * from account2");
        while (rs.next()) {
            accounts.add(new Account(Integer.parseInt(rs.getString("id")),
                    rs.getString("name"),
                    Integer.parseInt(rs.getString("balance"))));
        }
        System.out.println("conn is obtained...");
    } catch (SQLException e) {
        e.printStackTrace();
    }finally{
        if(connection!=null){
            try {
                connection.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

    return accounts;
}
```

Less code less bug

```java
@Override
public List<Account> getAllAccounts() {

    template =new JdbcTemplate(dataSource);

    List<Account>accounts=template.query("select * from account2", new AccountRowMapper());
    return accounts;

}
```

get an account

```
@Override
public Account getAccount(int id) {
    template=new JdbcTemplate(dataSource);
    Account account=template.queryForObject("select * from account2 where id=?",
            new AccountRowMapper(), id);
    |
    return account;
}
```

Update account

```
@Override
public void update(Account account) {

    template=new JdbcTemplate(dataSource);
    template.update("update account2 set balance=? where id=?", new Object[]{account.get

}
```

```
public class AccountRowMapper implements RowMapper<Account>{

    @Override
    public Account mapRow(ResultSet rs, int no) throws SQLException {
        Account account=new Account();
        account.setId(rs.getInt("id"));
        account.setBalance(rs.getInt("balance"));
        account.setName(rs.getString("name"));
        return account;
    }

}
```

```
@Override
public void addAccount(Account account) {
    String sql="insert into account(id, name, balance) values (?,?,?)";
    jdbcTemplate =new JdbcTemplate(dataSource);
    jdbcTemplate.update(sql, new Object[] {account.getId(), account.getN
```

Add account

Template DP

```
public abstract class ComputerTemplate {
    public final void buildComputer(){
        collectComponents();//ram, fan, gpu, cpu
        assembleComponents();
        installOs();
        startComputer();
        System.out.println("Computer is on");
    }

    private void collectComponents(){
        System.out.println("Computer with 4GB Ram, 1 TB HDD, 4 GB graphics card
    }
    private void startComputer(){
        System.out.println("System is booting");
    }
    public abstract void installOs();
    public abstract void assembleComponents();
}
```

Template DP

```java
public class Laptop extends ComputerTemplate {
    @Override
    public void installOs() {
        System.out.println("Installing windows");
    }

    @Override
    public void assembleComponents() {
        System.out.println("Joining all units, plus 4 HDMI");
    }
}
```

```java
public class Server extends ComputerTemplate {
    @Override
    public void installOs() {
        System.out.println("Installing Ubuntu");
    }

    @Override
    public void assembleComponents() {
        System.out.println("Joining all units, 0 hdmi, 1 VGA port");
    }
}
```