

PreparedStatementCallback interface

It processes the input parameters and output results. In such case, you don't need to care about single and double quotes.

Method of PreparedStatementCallback interface

It has only one method `doInPreparedStatement`. Syntax of the method is given below:

```
public T doInPreparedStatement(PreparedStatement ps) throws SQLException, DataAccessException
```

ResultSetExtractor is an interface that is used to fetch the records from the database. It's a callback interface that is used by [JDBC](#) Template's `query()` method where we need to pass the instance of `ResultSetExtractor` in order to fetch the data.

Syntax of query() method of ResultSetExtractor:

```
public T query(String sqlQuery, ResultSetExtractor<T> resultSetExtractor)
```

In order to fetch the data using `ResultSetExtractor`, we need to implement the `ResultSetExtractor` interface and provide the definition for its method. It has only one method. i.e., `extractData()` which takes an instance of `ResultSet` as an argument and returns the list.

Syntax of extractData() method:

```
public T extractData(ResultSet resultSet) throws SQL Exception, DataAccessException
```

In Spring, the **RowMapper** interface is used to fetch the records from the database using the `query()` method of the **JdbcTemplate** class.

Syntax for query() method of JdbcTemplate class:

```
public T query(String sqlQuery, RowMapper<T> rowMapper)
```

`RowMapper` is a callback interface that is called for each row and maps the row of relations with the instances to the model(user-defined) class. Unlike `ResultSetExtractor` the `RowMapper` iterates the `ResultSet` internally and adds the extracted data into a collection, And we do not need to write the code for collections as we do in `ResultSetExtractor`. It has only one method `mapRow()` which takes two arguments `ResultSet` and `rowNumber` respectively. In order to use `RowMapper`, we need to implement this interface and provide the definition for `mapRow()` method.

Syntax for rowMapper() method:

```
public T mapRow(ResultSet resultSet, int rowNum)throws  
SQLException
```

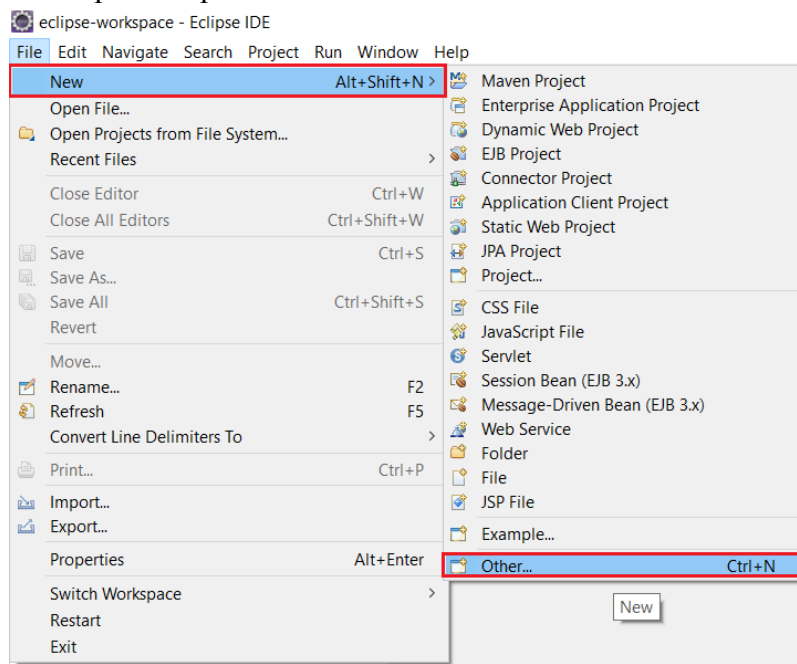
Spring JDBC

1. Write a program to insert, update and delete records from the given table.
2. Write a program to demonstrate PreparedStatement in Spring JdbcTemplate.
3. Write a program in Spring JDBC to demonstrate ResultSetExtractor Interface.
4. Write a program to demonstrate RowMapper interface to fetch the records from the database

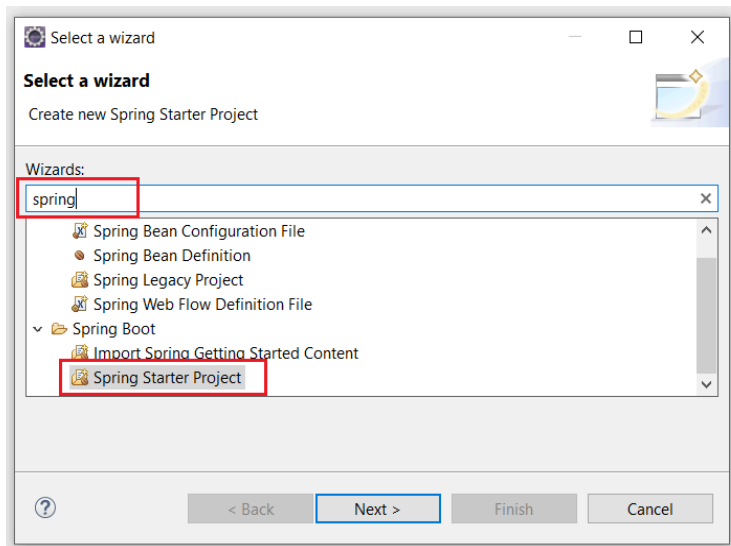
Steps to Create Spring Legacy Project

Step 1 : Creating Spring Legacy Project.

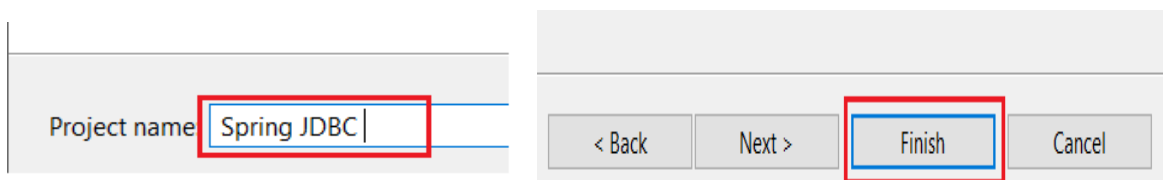
1.1 : Open Eclipse. Go To File > New > Other.



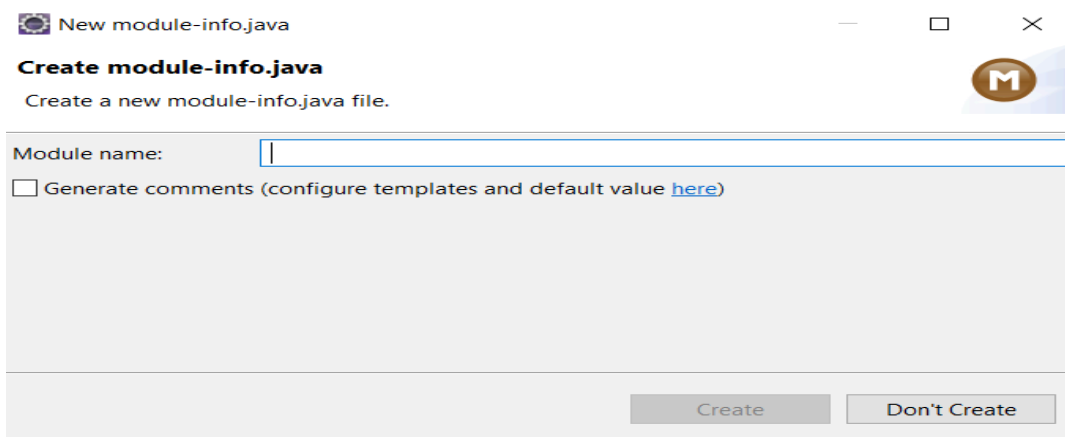
1.2: Search for 'spring' and Select 'Spring Legacy Project'. Then Click on Next.



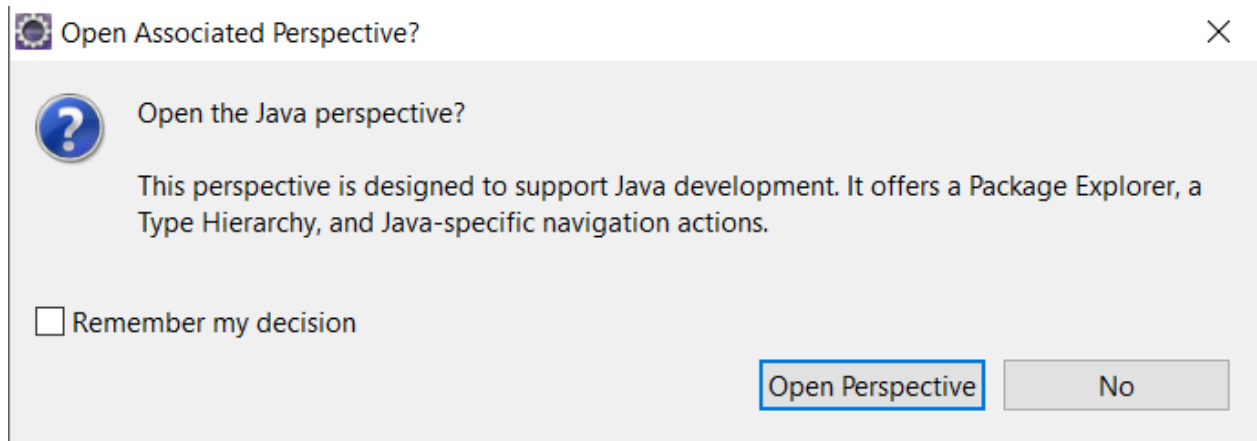
1.3 : Enter Project Name of your wish, and click on Finish.



1.4 : If asked to create module-info.java file, select 'Don't Create'.



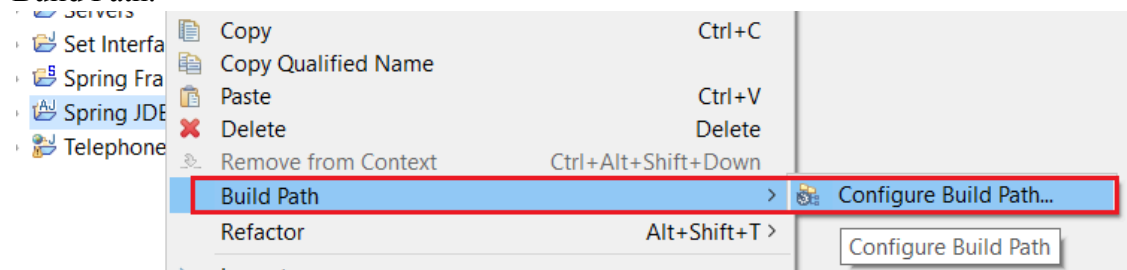
1.5 : Finally if you are asked to Open Java Perspective, just choose **NO**.



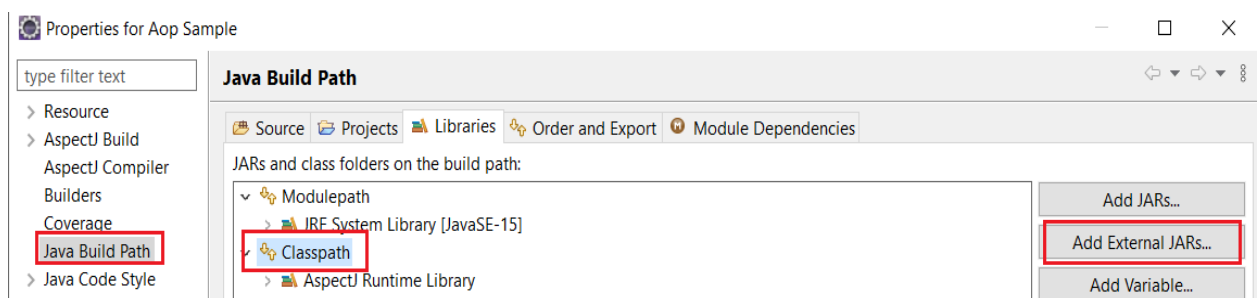
This creates your AspectJ project.

step 2 : Adding the Spring Libraries.

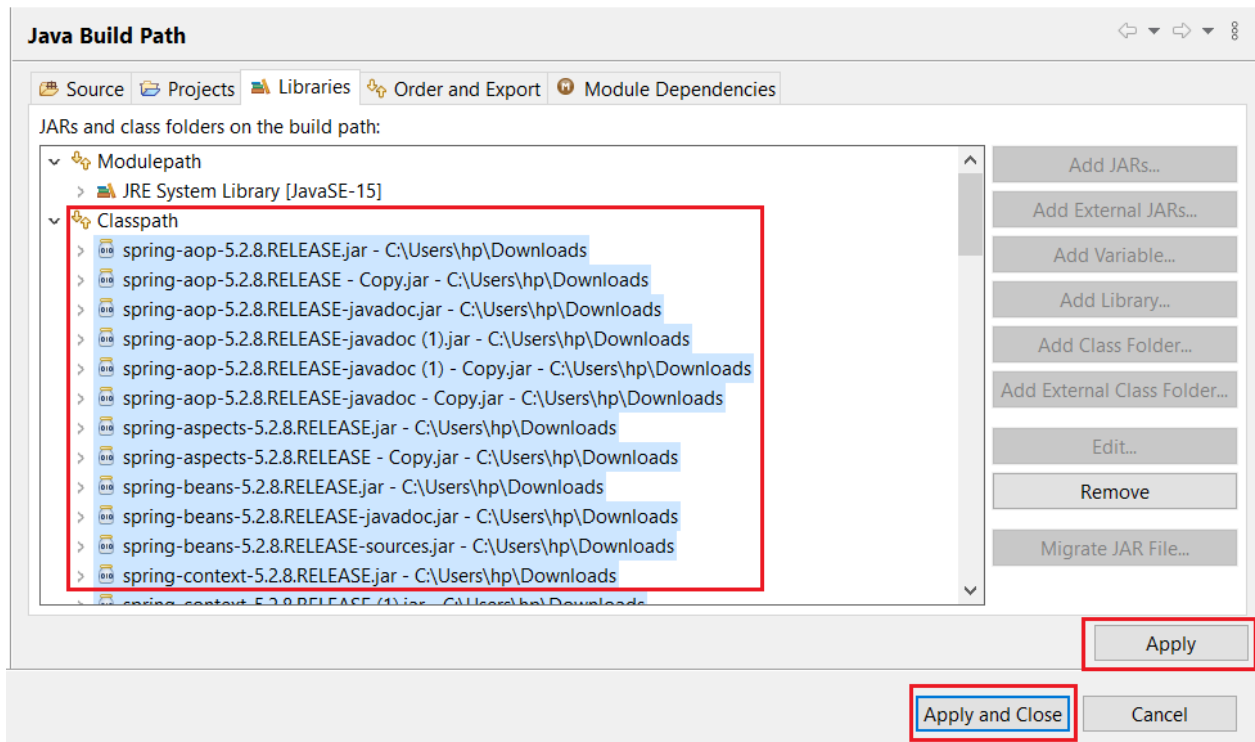
2.1 : Right click on your Newly created AspectJ project, Choose Build Path > Configure Build Path.



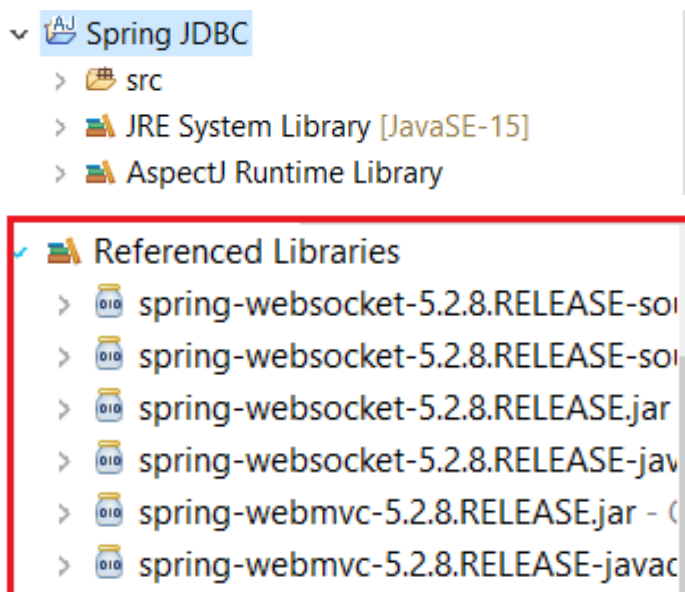
2.2 On Java Build Path wizard, Choose **Classpath** and then select **Add External JARs**.



2.3 : Choose all the Spring Libraries you've downloaded, and click on OPEN. This will add all libraries to Classpath.



2.4 Finally click on Apply & Close, now you are ready to work with Aspects in Spring.

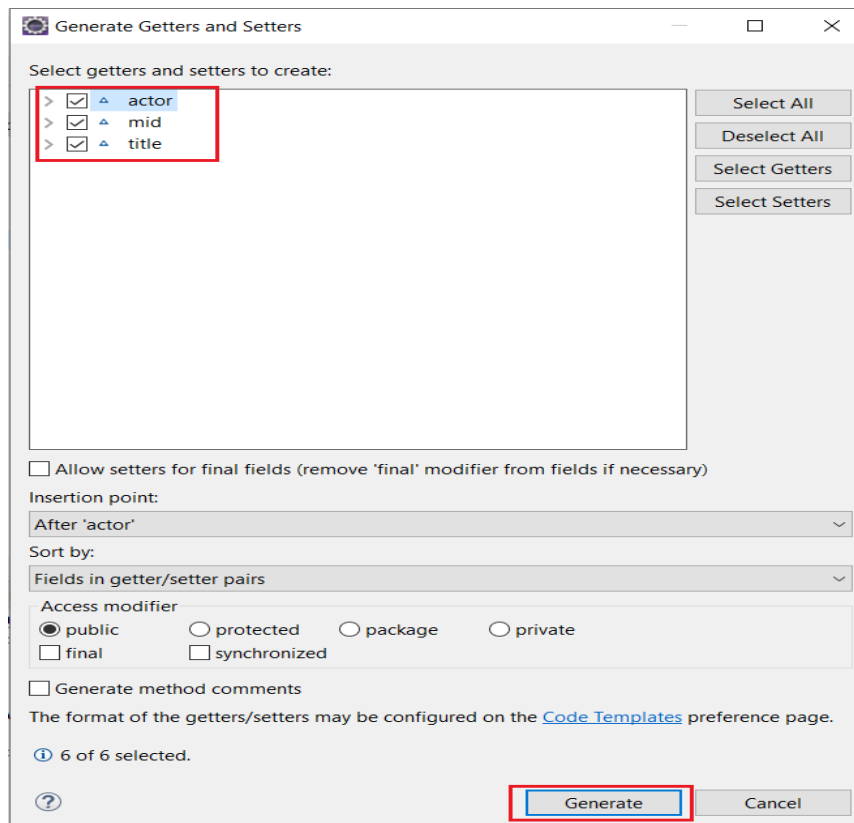


Problem Statement 9.1 : Write a program to insert, update and delete records from the given table.

Solution:

How to generate getter and setter methods

Right click on file-> source-> Generate getters and setters methods.



Filename-Movie1.java

```
public class Movie1 {
    int mid;
    String title;
    String actor;
    public Movie1(int mid, String title, String actor) {
        super();
        this.mid = mid;
        this.title = title;
        this.actor = actor;
    }
    public Movie1() {
        super();
    }
    public int getMid() {
        return mid;
    }
}
```

```

    public void setMid(int mid) {
        this.mid = mid;
    }

    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getActor() {
        return actor;
    }
    public void setActor(String actor) {
        this.actor = actor;
    }
}

```

Filename-MovieDAO.java

```
import org.springframework.jdbc.core.JdbcTemplate;
```

```

public class MovieDAO {
    JdbcTemplate jdbcTemplate;

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
    public int insMovie(Movie1 m1)
    {
        String insSql="insert into movies
values("+m1.getMid()+",""+m1.getTitle()+",""+m1.getActor()+")";

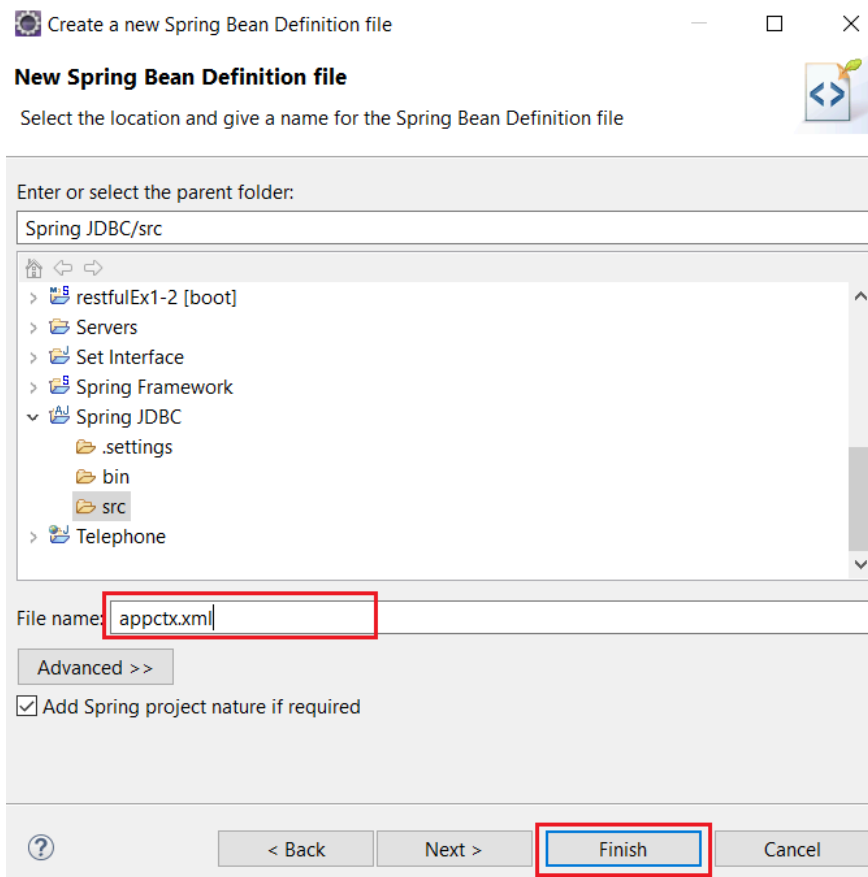
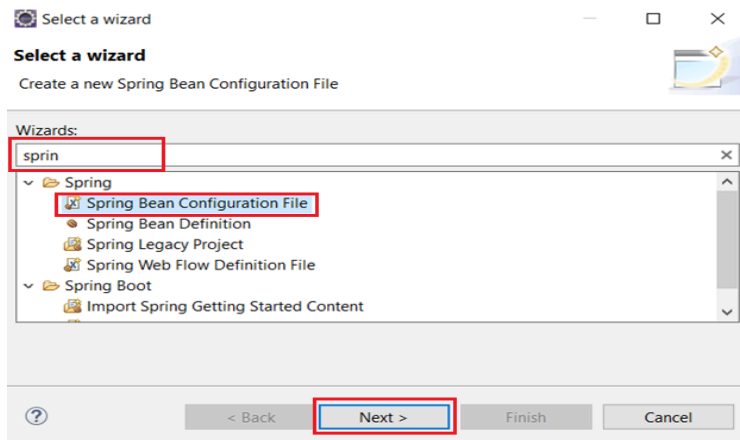
        return jdbcTemplate.update(insSql);
    }

    public int updateMovie(Movie1 m1){
        String query="update movies set
title="+m1.getTitle()+",actor="+m1.getActor()+" where mid="+m1.getMid()+" ";
        return jdbcTemplate.update(query);
    }

    public int deleteMovie(Movie1 m1){
        String query="delete from movies where mid="+m1.getMid()+" ";
        return jdbcTemplate.update(query);
    } }

```

Create Xml file



Create Xml file

Filename-appctx.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="org.postgresql.Driver" />
        <property name="url" value="jdbc:postgresql://localhost:5434/postgres" />
        <property name="username" value="postgres" />
        <property name="password" value="ravital23" />
    </bean>

    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="ds"></property>
    </bean>

    <bean id="mymovie" class="MovieDAO">
        <property name="jdbcTemplate" ref="jdbcTemplate"></property>
    </bean>
</beans>
```

Create Main java File

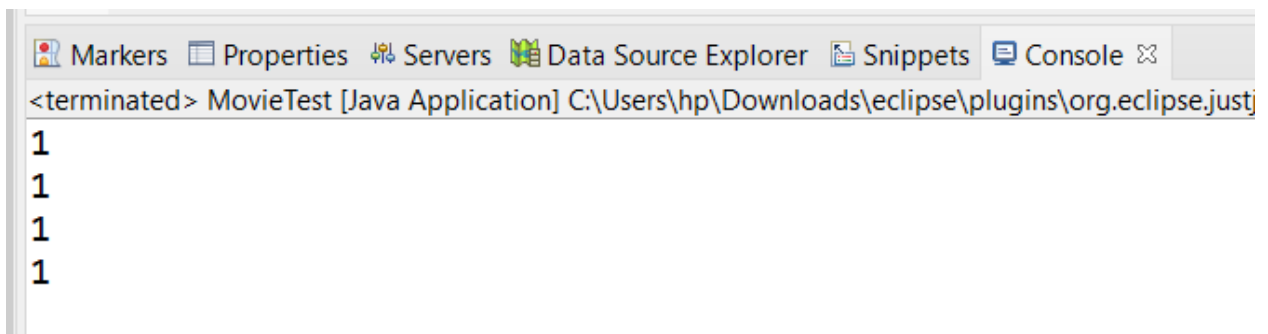
Filename-MovieTest.java

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class MovieTest {
    private static ApplicationContext appCon;
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        appCon = new ClassPathXmlApplicationContext("appctx.xml");
        MovieDAO m1 = (MovieDAO) appCon.getBean("mymovie");
        // insert query
        Movie1 t1 = new Movie1(4, "17 Again", "Zac");
        System.out.println(m1.insMovie(t1));

        Movie1 t = new Movie1(5, "Interstellar", "Christopher");
        System.out.println(m1.insMovie(t));
        // update query
        int status = m1.updateMovie(new Movie1(1, "18 Again", "Zac"));
        System.out.println(status);
        // delete
```

```
Movie1 t2=new Movie1();
t2.setMid(3);
int s=m1.deleteMovie(t2);
System.out.println(s);

} }
```



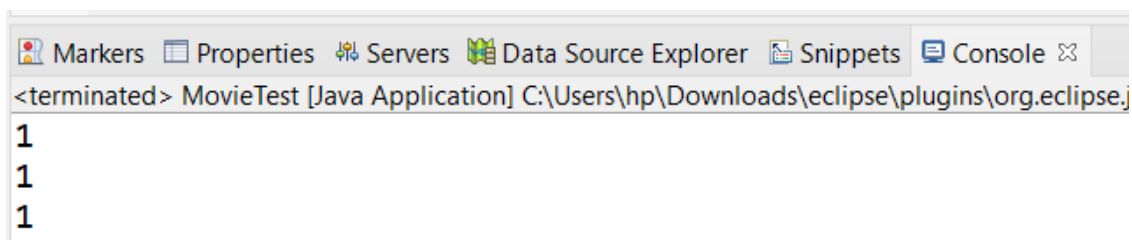
```
<terminated> MovieTest [Java Application] C:\Users\hp\Downloads\eclipse\plugins\org.eclipse.just
1
1
1
1
```

First we insert 3 records





// insert query

```
Movie1 t1 = new Movie1(1, "17 Again", "Zac");
System.out.println(m1.insMovie(t1));





Movie1 t2 = new Movie1(2, "23 Again", "Zac");
System.out.println(m1.insMovie(t));
Movie1 t3 = new Movie1(3, "Interstellar", "Christopher");
System.out.println(m1.insMovie(t));
```



```
<terminated> MovieTest [Java Application] C:\Users\hp\Downloads\eclipse\plugins\org.eclipse.j
1
1
1
```





	Data Output	Explain	Messages	Notifications
	 mid [PK] integer 		title character varying (50) 	actor character varying (50) 
1		1	17 Again	Zac
2		2	23 Again	Zac
3		3	Interstellar	Christopher





Update:
We update row 1

	Data Output	Explain	Messages	Notifications
	 mid [PK] integer 		title character varying (50) 	actor character varying (50) 
1		1	18 Again	Zac
2		2	23 Again	Zac
3		4	17 Again	Zac
4		5	Interstellar	Christopher

Delete:

We deleted row no 3 So, After deleted row

	Data Output	Explain	Messages	Notifications
	 mid [PK] integer 		title character varying (50) 	actor character varying (50) 
1		1	17 Again	Zac
2		2	23 Again	Zac
3		3	Interstellar	Christopher

	Data Output	Explain	Messages	Notifications
	 mid [PK] integer 	title character varying (50) 	actor character varying (50) 	
1		1	18 Again	Zac
2		2	23 Again	Zac
3		4	17 Again	Zac
4		5	Interstellar	Christopher

Database:

Create Movies Table :

CREATE TABLE `movies` (


`mid` int,

`title` varchar(50),

`actor` varchar(50),

PRIMARY KEY (`mid`)

);


 postgres/postgres@PostgreSQL 13 ▾

Query Editor Query History

```

1 CREATE TABLE movies (
2   mid int,
3   title varchar(50),
4   actor varchar(50),
5   PRIMARY KEY (mid)
6 );
7

```

 Tanmay

Data Output Explain Messages Notifications

CREATE TABLE

Query returned successfully in 1 secs 350 msec.

Problem Statement 9.2 : Write a program to demonstrate PreparedStatement in Spring JdbcTemplate.

Solution :

Filename- Movie1.java

```
public class Movie1 {
    int mid;
    String title;
    String actor;
    public Movie1(int mid, String title, String actor) {
        super();
        this.mid = mid;
        this.title = title;
        this.actor = actor;
    }
    public Movie1() {
        super();
    }
    public int getMid() {
        return mid;
    }
    public void setMid(int mid) {
        this.mid = mid;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getActor() {
        return actor;
    }
    public void setActor(String actor) {
        this.actor = actor;
    }
}
```

Filename- MovieDAO1.java

```
import java.sql.PreparedStatement;
import java.sql.SQLException;
```

```
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
```

```

import org.springframework.jdbc.core.PreparedStatementCallback;

public class MovieDAO1 {
    JdbcTemplate jdbcTemplate;

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public Boolean saveMovieByPreparedStatement(final Movie1 e){
        String query="insert into movies values(?,?,?)";
        return jdbcTemplate.execute(query,new
            PreparedStatementCallback<Boolean>(){
                @Override
                public Boolean doInPreparedStatement(PreparedStatement ps)
                    throws SQLException, DataAccessException {
                    ps.setInt(1,e.getMid());
                    ps.setString(2,e.getTitle());
                    ps.setString(3,e.getActor());
                    return ps.execute();
                }
            });
    }
}

```

Filename- appctx.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="org.postgresql.Driver" />
        <property name="url" value="jdbc:postgresql://localhost:5434/postgres" />
        <property name="username" value="postgres" />
        <property name="password" value="ravita123" />
    </bean>
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="ds"></property>
    </bean>
    <bean id="mymovie" class="MovieDAO1">
        <property name="jdbcTemplate" ref="jdbcTemplate"></property>
    </bean>
</beans>

```

Filename- MovieTest1.java

```
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;
```

```
public class MovieTest1 {  
    private static ApplicationContext appCon;  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        appCon = new ClassPathXmlApplicationContext("appctx1.xml");  
        MovieDAO1 m1=(MovieDAO1)appCon.getBean("mymovie");  
        m1.saveMovieByPreparedStatement(new  
        Movie1(3,"Inception","Cobb"));  
    }  
}
```

Output-

Data Output				Explain	Messages	Notifications
	mid [PK] integer		title character varying (50)		actor character varying (50)	
1		1	18 Again		Zac	
2		2	23 Again		Zac	
3		3	Inception		Cobb	
4		4	17 Again		Zac	
5		5	Interstellar		Christopher	

Problem Statement 9.3 : Write a program in Spring JDBC to demonstrate ResultSetExtractor Interface.

Solution :

Filename- Movie2.java

```
public class Movie2 {
    int mid;
    String title;
    String actor;
    public int getMid() {
        return mid;
    }
    public void setMid(int mid) {
        this.mid = mid;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getActor() {
        return actor;
    }
    public void setActor(String actor) {
        this.actor = actor;
    }
    public String toString(){
        return mid+" "+title+" "+actor;
    }
}
```

Filename- MovieDAO2.java

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.ResultSetExtractor;
public class MovieDAO2 {
    JdbcTemplate jdbcTemplate;

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
    public List<Movie2> getAllMovie(){
        return jdbcTemplate.query("select * from movies",new
```

```

ResultSetExtractor<List<Movie2>>() {
    @Override
    public List<Movie2> extractData(ResultSet rs) throws SQLException,
        DataAccessException {

        List<Movie2> list=new ArrayList<Movie2>();
        while(rs.next()){
            Movie2 e=new Movie2();
            e.setMid(rs.getInt(1));
            e.setTitle(rs.getString(2));
            e.setActor(rs.getString(3));
            list.add(e);
        }
        return list;
    }
});
}
}

```

Filename- appctx2.java

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="org.postgresql.Driver" />
        <property name="url" value="jdbc:postgresql://localhost:5434/postgres" />
        <property name="username" value="postgres" />
        <property name="password" value="ravital23" />
    </bean>

    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="ds"></property>
    </bean>

    <bean id="mymovie" class="MovieDAO2">
        <property name="jdbcTemplate" ref="jdbcTemplate"></property>
    </bean>
</beans>

```

Filename- MovieTest2.java

```

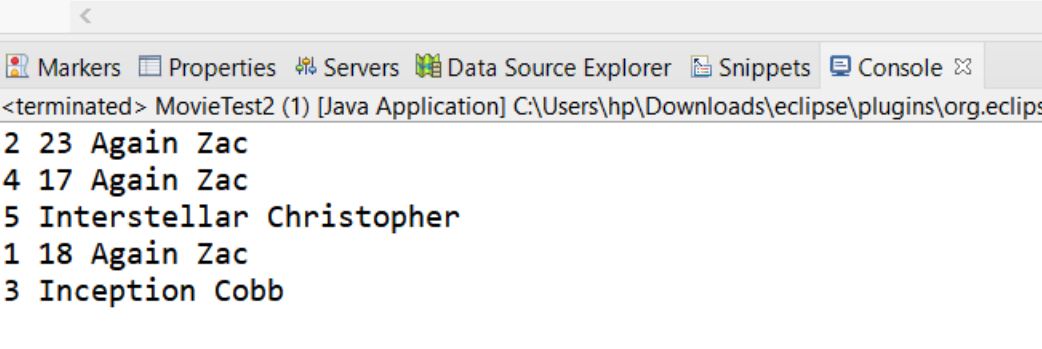
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class MovieTest2 {
    private static ApplicationContext appCon;
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        appCon = new ClassPathXmlApplicationContext("appctx2.xml");
    }
}

```

```
MovieDAO2 m1=(MovieDAO2)appCon.getBean("mymovie");  
List<Movie2> list=m1.getAllMovie();
```

```
for(Movie2 e:list)  
    System.out.println(e);  
}
```

OUTPUT-

A screenshot of the Eclipse IDE's console window. The window title is "<terminated> MovieTest2 (1) [Java Application] C:\Users\hp\Downloads\eclipse\plugins\org.eclips...". The console shows five lines of output, each with an index, a number, and a name: "2 23 Again Zac", "4 17 Again Zac", "5 Interstellar Christopher", "1 18 Again Zac", and "3 Inception Cobb".

```
<terminated> MovieTest2 (1) [Java Application] C:\Users\hp\Downloads\eclipse\plugins\org.eclips  
2 23 Again Zac  
4 17 Again Zac  
5 Interstellar Christopher  
1 18 Again Zac  
3 Inception Cobb
```

Problem Statement 9.4 : Write a program to demonstrate RowMapper interface to fetch the records from the database.

Solution :

Filename-Movie.java

```
public class Movie3 {
    int mid;
    String title;
    String actor;
    public Movie3(int mid, String title, String actor) {
        super();
        this.mid = mid;
        this.title = title;
        this.actor = actor;
    }

    public Movie3() {
        super();
        // TODO Auto-generated constructor stub
    }
    public int getMid() {
        return mid;
    }
    public void setMid(int mid) {
        this.mid = mid;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getActor() {
        return actor;
    }
    public void setActor(String actor) {
        this.actor = actor;
    }
}
```

Filename- MovieDAO3.java

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
```

```

public class MovieDAO3 {
    JdbcTemplate jdbcTemplate;

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public List<Movie2> getAllEmployeesRowMapper() {
        return jdbcTemplate.query("select * from movies",new
RowMapper<Movie2>(){
@Override
public Movie2 mapRow(ResultSet rs, int rownumber) throws SQLException {
    Movie2 e=new Movie2();
    e.setMid(rs.getInt(1));
    e.setTitle(rs.getString(2));
    e.setActor(rs.getString(3));
    return e;
}
});
}
}

```

Filename- appctx3.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="org.postgresql.Driver" />
        <property name="url" value="jdbc:postgresql://localhost:5434/postgres" />
        <property name="username" value="postgres" />
        <property name="password" value="ravital23" />
    </bean>

    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="ds"></property>
    </bean>

    <bean id="mymovie" class="MovieDAO3">
        <property name="jdbcTemplate" ref="jdbcTemplate"></property>
    </bean>
</beans>

```

Filename- MovieTest3.java

```

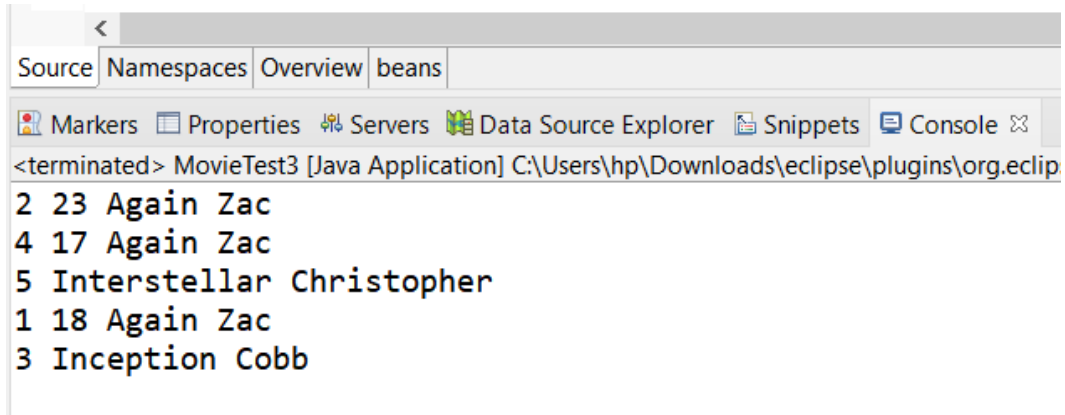
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class MovieTest3 {
    private static ApplicationContext appCon;
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        appCon = new ClassPathXmlApplicationContext("appctx3.xml");
    }
}

```

```
MovieDAO3 m1=(MovieDAO3)appCon.getBean("mymovie");
List<Movie2> list=m1.getAllEmployeesRowMapper();

    for(Movie2 e:list)
        System.out.println(e);
}}
```

OUTPUT-



The screenshot shows the Eclipse IDE's console window. The title bar indicates the application is 'MovieTest3 [Java Application]' located at 'C:\Users\hp\Downloads\eclipse\plugins\org.eclip'. The console output consists of five lines, each representing a movie record with an ID, a count, a title, and an actor's name:

```
<terminated> MovieTest3 [Java Application] C:\Users\hp\Downloads\eclipse\plugins\org.eclip
2 23 Again Zac
4 17 Again Zac
5 Interstellar Christopher
1 18 Again Zac
3 Inception Cobb
```