Spring JdbcTemplate Tutorial
-Refer to https://www.javatpoint.com/spring-JdbcTemplate-tutorial
-https://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/jdbc.html
1. Spring JdbcTemplate is a powerful mechanism to connect to the database and execute SQL queries.

2. It internally uses JDBC api, but eliminates a lot of problems of JDBC API.

3. Problems of JDBC API
  1)We need to write a lot of code before and after executing the query, such as creating connection, statement, closing resultset, connection etc.
  2)We need to perform exception handling code on the database logic.
  3)We need to handle transaction.
  4)Repetition of all these codes from one to another database logic is a time consuming task.

4. Advantage of Spring JdbcTemplate
  1)Spring JdbcTemplate eliminates all the above mentioned problems of JDBC API.
  2)It provides you methods to write the queries directly, so it saves a lot of work and time.

5. Spring Jdbc Approaches
  -Spring framework provides following approaches for JDBC database access:
  1)JdbcTemplate
    -Is the classic Spring JDBC approach and the most popular. This "lowest level" approach and all others use a JdbcTemplate under the covers.
  2)NamedParameterJdbcTemplate
    -Wraps a JdbcTemplate to provide named parameters instead of the traditional JDBC "?" placeholders.
    -This approach provides better documentation and ease of use when you have multiple parameters for an SQL statement.
  3)SimpleJdbcTemplate
  4)SimpleJdbcInsert and SimpleJdbcCall
    -Optimize database metadata to limit the amount of necessary configuration.
    -This approach simplifies coding so that you only need to provide the name of the table or procedure and provide a map of parameters matching the column names.
    -This only works if the database provides adequate metadata.
    -If the database doesn’t provide this metadata, you will have to provide explicit configuration of the parameters.
  5)RDBMS Objects including MappingSqlQuery, SqlUpdate and StoredProcedure requires you to create reusable and thread-safe objects during initialization of your data access layer.
  6)This approach is modeled after JDO Query wherein you define your query string, declare parameters, and compile the query.
  7)Once you do that, execute methods can be called multiple times with various parameter values passed in.

6. JdbcTemplate class
  1)It is the central class in the Spring JDBC support classes.
  2)It takes care of creation and release of resources such as creating and closing of connection object etc.
  3)So it will not lead to any problem if you forget to close the connection.
  4)It handles the exception and provides the informative exception messages by the help of excepion classes defined in the org.springframework.dao package.

5)We can perform all the database operations by the help of JdbcTemplate class such as insertion, updation, deletion and retrieval of the data from the database.

7. The methods of spring JdbcTemplate class.

1)public int update(String query)
-Is used to insert, update and delete records.

2)public int update(String query,Object... args)
-Is used to insert, update and delete records using PreparedStatement using given arguments.

3)public void execute(String query)
-Is used to execute DDL query.

4)public T execute(String sql, PreparedStatementCallback action)
-Executes the query by using PreparedStatement callback.

5)public T query(String sql, ResultSetExtractor rse)
-Is used to fetch records using ResultSetExtractor.

6)public List query(String sql, RowMapper rse)
-Is used to fetch records using RowMapper.

8. Example of Spring JdbcTemplate

1)Create Table

```
create table employee(
  id number(10),
  name varchar2(100),
  salary number(10)
);
```

2)Employee.java

```
package com.example;

public class Employee {
  private int id;
  private String name;
  private float salary;
  //no-arg and parameterized constructors
  //getters and setters
}
```

3)EmployeeDao.java

```
package com.example;
import org.springframework.jdbc.core.JdbcTemplate;

public class EmployeeDao {
  private JdbcTemplate jdbcTemplate;

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

  public int saveEmployee(Employee e){
    String query="insert into employee values(
    '"+e.getId()+"','"+e.getName()+"','"+e.getSalary()+"')";
    return jdbcTemplate.update(query);
  }
```

```
96        public int updateEmployee(Employee e){
97          String query="update employee set
98          name='"+e.getName()+"',salary='"+e.getSalary()+"' where id='"+e.getId()+"' ";
99          return jdbcTemplate.update(query);
100       }
101       public int deleteEmployee(Employee e){
102         String query="delete from employee where id='"+e.getId()+"' ";
103         return jdbcTemplate.update(query);
104       }
105     }
106
107   4)applicationContext.xml
108     <?xml version="1.0" encoding="UTF-8"?>
109     <beans
110       xmlns="http://www.springframework.org/schema/beans"
111       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
112       xmlns:p="http://www.springframework.org/schema/p"
113       xsi:schemaLocation="http://www.springframework.org/schema/beans
114         http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
115
116       <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
117         <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
118         <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
119         <property name="username" value="scott" />
120         <property name="password" value="tiger" />
121       </bean>
122
123       <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
124         <property name="dataSource" ref="ds" />
125       </bean>
126
127       <bean id="edao" class="com.example.EmployeeDao">
128         <property name="jdbcTemplate" ref="jdbcTemplate" />
129       </bean>
130
131     </beans>
132
133   5)Test.java
134     package com.example;
135
136     import org.springframework.context.ApplicationContext;
137     import org.springframework.context.support.ClassPathXmlApplicationContext;
138     public class Test {
139
140       public static void main(String[] args) {
141         ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");
142
143         EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
144         int status=dao.saveEmployee(new Employee(102,"Amit",35000));
145         System.out.println(status);
146
147         /*int status=dao.updateEmployee(new Employee(102,"Sonoo",15000));
148           System.out.println(status);
149         */
```

```
150
151         /*Employee e=new Employee();
152         e.setId(102);
153         int status=dao.deleteEmployee(e);
154         System.out.println(status);*/
155       }
156     }
157
158
159 9. Example of PreparedStatement in Spring JdbcTemplate
160   1)We can execute parameterized query using Spring JdbcTemplate by the help of execute()
      method of JdbcTemplate class.
161   2)To use parameterized query, we pass the instance of PreparedStatementCallback in the
      execute method.
162   3)Syntax of execute method to use parameterized query
163     public T execute(String sql,PreparedStatementCallback<T>);
164
165   4)PreparedStatementCallback interface
166     -It processes the input parameters and output results.
167     -In such case, you don't need to care about single and double quotes.
168
169   5)Method of PreparedStatementCallback interface
170     -It has only one method doInPreparedStatement.
171     -Syntax of the method is given below:
172
173       public T doInPreparedStatement(PreparedStatement ps)throws SQLException,
          DataAccessException
174
175   6)Example of using PreparedStatement in Spring
176   7)Create Table
177     create table employee(
178       id number(10),
179       name varchar2(100),
180       salary number(10)
181     );
182
183   8)Employee.java
184     package com.example;
185
186     public class Employee {
187       private int id;
188       private String name;
189       private float salary;
190       //no-arg and parameterized constructors
191       //getters and setters
192     }
193
194   9)EmployeeDao.java
195     package com.example;
196     import java.sql.PreparedStatement;
197     import java.sql.SQLException;
198
199     import org.springframework.dao.DataAccessException;
200     import org.springframework.jdbc.core.JdbcTemplate;
```

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

public class EmployeeDao {
  private JdbcTemplate jdbcTemplate;

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

  public Boolean saveEmployeeByPreparedStatement(final Employee e){
    String query="insert into employee values(?,?,?)";
    return jdbcTemplate.execute(query,new PreparedStatementCallback<Boolean>(){
      @Override
      public Boolean doInPreparedStatement(PreparedStatement ps)
          throws SQLException, DataAccessException {

        ps.setInt(1,e.getId());
        ps.setString(2,e.getName());
        ps.setFloat(3,e.getSalary());

        return ps.execute();

      }
    });
  }
}
```

10)applicationContext.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"
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

  <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
    <property name="username" value="scott" />
    <property name="password" value="tiger" />
  </bean>

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

  <bean id="edao" class="com.example.EmployeeDao">
    <property name="jdbcTemplate" ref="jdbcTemplate" />
  </bean>
</beans>
```

11)Test.java

```
package com.example;
```

```
255
256     import org.springframework.context.ApplicationContext;
257     import org.springframework.context.support.ClassPathXmlApplicationContext;
258     public class Test {
259
260       public static void main(String[] args) {
261         ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");
262
263         EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
264         dao.saveEmployeeByPreparedStatement(new Employee(108,"Amit",35000));
265       }
266     }
267
268
269 10. ResultSetExtractor Example | Fetching Records by Spring JdbcTemplate
270   1)We can easily fetch the records from the database using query() method of JdbcTemplate
      class where we need to pass the instance of ResultSetExtractor.
271   2)Syntax of query method using ResultSetExtractor
272     public T query(String sql,ResultSetExtractor<T> rse)
273   3)ResultSetExtractor Interface
274     -ResultSetExtractor interface can be used to fetch records from the database.
275     -It accepts a ResultSet and returns the list.
276
277   4)Method of ResultSetExtractor interface
278     -It defines only one method extractData that accepts ResultSet instance as a parameter.
279     -Syntax of the method is given below:
280       public T extractData(ResultSet rs)throws SQLException,DataAccessException
281
282   5)Example of ResultSetExtractor Interface to show all the records of the table
283   6)Create Table
284     create table employee(
285       id number(10),
286       name varchar2(100),
287       salary number(10)
288     );
289
290   7)Employee.java
291     package com.example;
292
293     public class Employee {
294       private int id;
295       private String name;
296       private float salary;
297       //no-arg and parameterized constructors
298       //getters and setters
299       public String toString(){
300         return id+" "+name+" "+salary;
301       }
302     }
303
304   8)EmployeeDao.java
305     package com.example;
306     import java.sql.ResultSet;
307     import java.sql.SQLException;
```

```
308     import java.util.ArrayList;
309     import java.util.List;
310     import org.springframework.dao.DataAccessException;
311     import org.springframework.jdbc.core.JdbcTemplate;
312     import org.springframework.jdbc.core.ResultSetExtractor;
313
314     public class EmployeeDao {
315       private JdbcTemplate template;
316
317       public void setTemplate(JdbcTemplate template) {
318         this.template = template;
319       }
320
321       public List<Employee> getAllEmployees(){
322        return template.query("select * from employee",new
          ResultSetExtractor<List<Employee>>(){
323         @Override
324          public List<Employee> extractData(ResultSet rs) throws SQLException,
325              DataAccessException {
326
327            List<Employee> list=new ArrayList<Employee>();
328            while(rs.next()){
329              Employee e=new Employee();
330              e.setId(rs.getInt(1));
331              e.setName(rs.getString(2));
332              e.setSalary(rs.getInt(3));
333              list.add(e);
334            }
335            return list;
336         }
337         });
338        }
339       }
340     }
341
342   9)applicationContext.xml
343     <?xml version="1.0" encoding="UTF-8"?>
344     <beans
345       xmlns="http://www.springframework.org/schema/beans"
346       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
347       xmlns:p="http://www.springframework.org/schema/p"
348       xsi:schemaLocation="http://www.springframework.org/schema/beans
349         http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
350
351       <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
352         <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
353         <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
354         <property name="username" value="scott" />
355         <property name="password" value="tiger" />
356       </bean>
357
358       <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
359         <property name="dataSource" ref="ds" />
360       </bean>
```

```
361
362         <bean id="edao" class="com.example.EmployeeDao">
363           <property name="jdbcTemplate" ref="jdbcTemplate" />
364         </bean>
365
366       </beans>
367
368     10)Test.java
369       package com.example;
370
371       import java.util.List;
372
373       import org.springframework.context.ApplicationContext;
374       import org.springframework.context.support.ClassPathXmlApplicationContext;
375       public class Test {
376
377         public static void main(String[] args) {
378           ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");
379           EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
380           List<Employee> list=dao.getAllEmployees();
381
382           for(Employee e:list)
383             System.out.println(e);
384
385         }
386       }
387
388
389   11. RowMapper Example | Fetching records by Spring JdbcTemplate
390     1)Like ResultSetExtractor, we can use RowMapper interface to fetch the records from the
        database using query() method of JdbcTemplate class.
391     2)In the execute of we need to pass the instance of RowMapper now.
392     3)Syntax of query method using RowMapper
393       public T query(String sql,RowMapper<T> rm)
394
395     4)RowMapper Interface
396       -RowMapper interface allows to map a row of the relations with the instance of user-defined
        class.
397       -It iterates the ResultSet internally and adds it into the collection.
398       -So we don't need to write a lot of code to fetch the records as ResultSetExtractor.
399
400     5)Advantage of RowMapper over ResultSetExtractor
401       -RowMapper saves a lot of code becuase it internally adds the data of ResultSet into the
        collection.
402
403     6)Method of RowMapper interface
404       -It defines only one method mapRow that accepts ResultSet instance and int as the parameter
        list.
405       -Syntax of the method is given below:
406
407       public T mapRow(ResultSet rs, int rowNumber)throws SQLException
408
409     7)Example of RowMapper Interface to show all the records of the table
410     8)Create Table
```

```
411     create table employee(
412       id number(10),
413       name varchar2(100),
414       salary number(10)
415     );
416
417   9)Employee.java
418     package com.example;
419
420     public class Employee {
421       private int id;
422       private String name;
423       private float salary;
424       //no-arg and parameterized constructors
425       //getters and setters
426       public String toString(){
427         return id+" "+name+" "+salary;
428       }
429     }
430
431   8)EmployeeDao.java
432     package com.example;
433     import java.sql.ResultSet;
434     import java.sql.SQLException;
435     import java.util.ArrayList;
436     import java.util.List;
437     import org.springframework.dao.DataAccessException;
438     import org.springframework.jdbc.core.JdbcTemplate;
439     import org.springframework.jdbc.core.ResultSetExtractor;
440
441     public class EmployeeDao {
442       private JdbcTemplate template;
443
444       public void setTemplate(JdbcTemplate template) {
445         this.template = template;
446       }
447
448       public List<Employee> getAllEmployees(){
449         return template.query("select * from employee",new RowMapper<Employee>(){
450           @Override
451           public Employee mapRow(ResultSet rs, int rownumber) throws SQLException {
452             Employee e=new Employee();
453             e.setId(rs.getInt(1));
454             e.setName(rs.getString(2));
455             e.setSalary(rs.getInt(3));
456             return e;
457           }
458         });
459       }
460     }
461
462   9)applicationContext.xml
463     <?xml version="1.0" encoding="UTF-8"?>
464     <beans
```

```
        xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

        <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
          <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
          <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
          <property name="username" value="scott" />
          <property name="password" value="tiger" />
        </bean>

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

        <bean id="edao" class="com.example.EmployeeDao">
          <property name="jdbcTemplate" ref="jdbcTemplate"></property>
        </bean>

      </beans>

    10)Test.java
      package com.example;

      import java.util.List;

      import org.springframework.context.ApplicationContext;
      import org.springframework.context.support.ClassPathXmlApplicationContext;
      public class Test {

        public static void main(String[] args) {
          ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");
          EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
          List<Employee> list=dao.getAllEmployeesRowMapper();

          for(Employee e:list)
            System.out.println(e);

        }
      }


  12. Spring NamedParameterJdbcTemplate Example
    1)Spring provides another way to insert data by named parameter.
    2)In such way, we use names instead of ?(question mark).
    3)So it is better to remember the data for the column.
    4)Simple example of named parameter query
      insert into employee values (:id,:name,:salary)
    5)Method of NamedParameterJdbcTemplate class
      -In this example,we are going to call only the execute method of
      NamedParameterJdbcTemplate class.
      -Syntax of the method is as follows:
```

```
518
519     pubic T execute(String sql,Map map,PreparedStatementCallback psc)
520
521   6)Example of NamedParameterJdbcTemplate class
522   7)Create Table
523     create table employee(
524       id number(10),
525       name varchar2(100),
526       salary number(10)
527     );
528
529   8)Employee.java
530     package com.example;
531
532     public class Employee {
533       private int id;
534       private String name;
535       private float salary;
536       //no-arg and parameterized constructors
537       //getters and setters
538     }
539
540   9)EmployeeDao.java
541     package com.example;
542
543     import java.sql.PreparedStatement;
544     import java.sql.SQLException;
545     import org.springframework.dao.DataAccessException;
546     import org.springframework.jdbc.core.PreparedStatementCallback;
547     import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
548     import java.util.*;
549
550     public class EmpDao {
551       NamedParameterJdbcTemplate template;
552
553       public EmpDao(NamedParameterJdbcTemplate template) {
554         this.template = template;
555       }
556       public  void save (Emp e){
557         String query="insert into employee values (:id,:name,:salary)";
558
559         Map<String,Object> map=new HashMap<String,Object>();
560         map.put("id",e.getId());
561         map.put("name",e.getName());
562         map.put("salary",e.getSalary());
563
564         template.execute(query,map,new PreparedStatementCallback() {
565           @Override
566           public Object doInPreparedStatement(PreparedStatement ps)
567                 throws SQLException, DataAccessException {
568             return ps.executeUpdate();
569           }
570         });
571       }
```

```
572     }
573
574   10)applicationContext.xml
575     <?xml version="1.0" encoding="UTF-8"?>
576     <beans
577       xmlns="http://www.springframework.org/schema/beans"
578       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
579       xmlns:p="http://www.springframework.org/schema/p"
580       xsi:schemaLocation="http://www.springframework.org/schema/beans
581         http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
582
583       <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
584         <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
585         <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
586         <property name="username" value="scott" />
587         <property name="password" value="tiger" />
588       </bean>
589
590       <bean id="jtemplate"
          class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">
591         <constructor-arg ref="ds" />
592       </bean>
593
594       <bean id="edao" class="com.example.EmpDao">
595         <constructor-arg>
596           <ref bean="jtemplate"/>
597         </constructor-arg>
598       </bean>
599
600     </beans>
601
602   11)Test.java
603     package com.example;
604
605     import org.springframework.beans.factory.BeanFactory;
606     import org.springframework.beans.factory.xml.XmlBeanFactory;
607     import org.springframework.core.io.ClassPathResource;
608     import org.springframework.core.io.Resource;
609
610     public class SimpleTest {
611       public static void main(String[] args) {
612
613         Resource r=new ClassPathResource("applicationContext.xml");
614         BeanFactory factory=new XmlBeanFactory(r);
615
616         EmpDao dao=(EmpDao)factory.getBean("edao");
617         dao.save(new Emp(23,"sonoo",50000));
618
619       }
620     }
621
622
623 13. Spring SimpleJdbcTemplate Example
624   1)Spring 3 JDBC supports the java 5 feature var-args (variable argument) and autoboxing by
```

the help of SimpleJdbcTemplate class.
2)SimpleJdbcTemplate class wraps the JdbcTemplate class and provides the update method where we can pass arbitrary number of arguments.

3)Syntax of update method of SimpleJdbcTemplate class
  int update(String sql,Object... parameters)
4)We should pass the parameter values in the update method in the order they are defined in the parameterized query.
5)Example of SimpleJdbcTemplate class
6)Create Table
```
  create table employee(
    id number(10),
    name varchar2(100),
    salary number(10)
  );
```

7)Employee.java
```
  package com.example;

  public class Employee {
    private int id;
    private String name;
    private float salary;
    //no-arg and parameterized constructors
    //getters and setters
  }
```

8)EmployeeDao.java
```
  package com.example;

  import org.springframework.jdbc.core.simple.SimpleJdbcTemplate;
  public class EmpDao {
    SimpleJdbcTemplate template;

    public EmpDao(SimpleJdbcTemplate template) {
      this.template = template;
    }
    public int update (Emp e){
      String query="update employee set name=? where id=?";
      return template.update(query,e.getName(),e.getId());

      //String query="update employee set name=?,salary=? where id=?";
      //return template.update(query,e.getName(),e.getSalary(),e.getId());
    }
  }
```

9)applicationContext.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"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
```

```
676
677         <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
678           <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
679           <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
680           <property name="username" value="scott" />
681           <property name="password" value="tiger" />
682         </bean>
683
684         <bean id="jtemplate" class="org.springframework.jdbc.core.simple.SimpleJdbcTemplate">
685           <constructor-arg ref="ds" />
686         </bean>
687
688         <bean id="edao" class="com.example.EmpDao">
689           <constructor-arg>
690             <ref bean="jtemplate"/>
691           </constructor-arg>
692         </bean>
693
694       </beans>
695
696     10)Test.java
697       package com.example;
698
699       import org.springframework.beans.factory.BeanFactory;
700       import org.springframework.beans.factory.xml.XmlBeanFactory;
701       import org.springframework.core.io.ClassPathResource;
702       import org.springframework.core.io.Resource;
703
704       public class SimpleTest {
705         public static void main(String[] args) {
706
707           Resource r=new ClassPathResource("applicationContext.xml");
708           BeanFactory factory=new XmlBeanFactory(r);
709
710           EmpDao dao=(EmpDao)factory.getBean("edao");
711           int status=dao.update(new Emp(23,"Tarun",35000));
712           System.out.println(status);
713         }
714       }
```