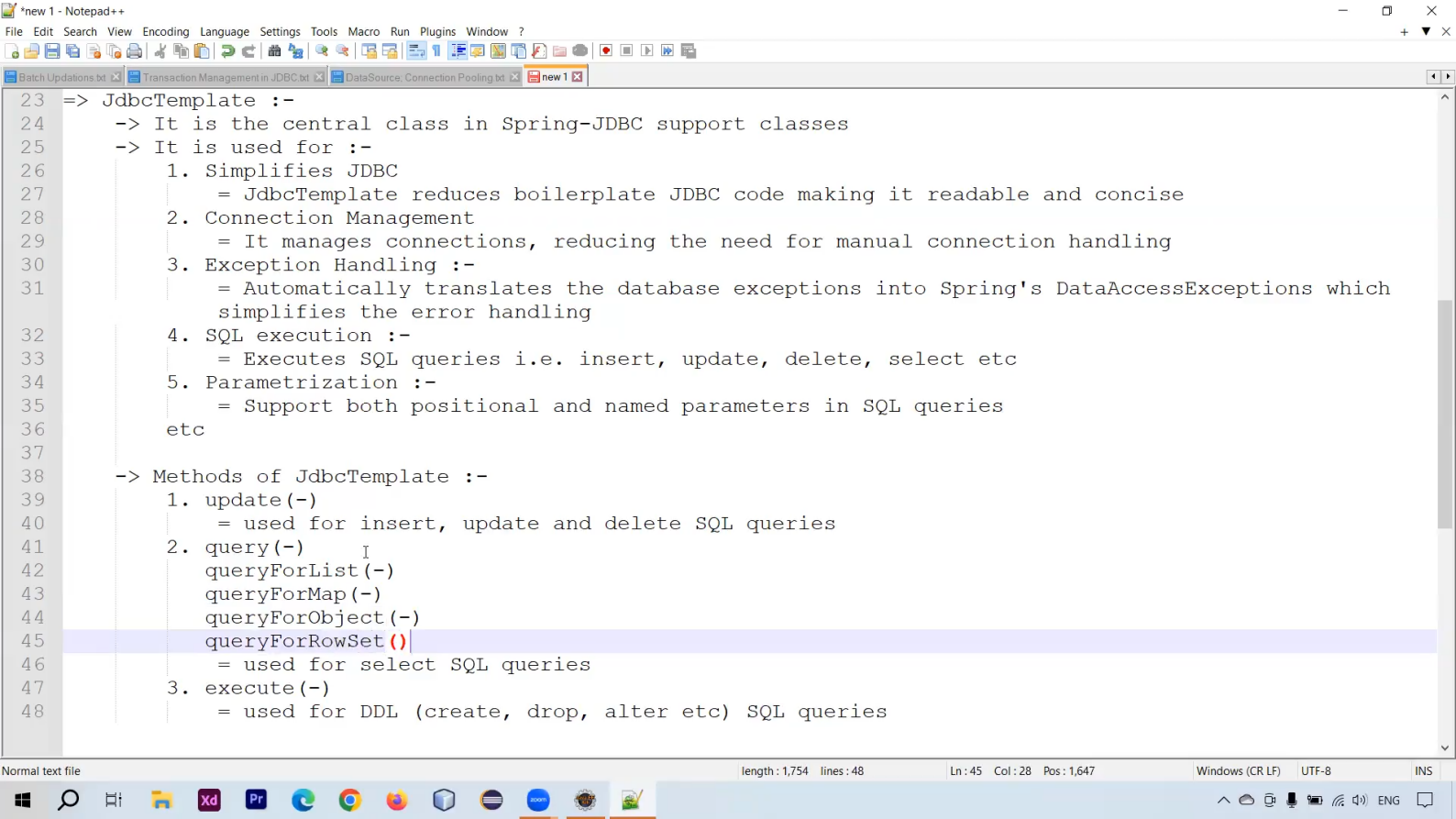
=> Spring JDBC Module :-
    -> Spring JDBC provides mechanism to connect the spring application with database and execute SQL queries
    -> NOTE : It internally uses JDBC API (but solves the problem of Plain-JDBC)

    -> Spring provides some classes which are as follows :-
        1. DriverManagerDataSource
        2. JdbcTemplate
        3. NamedParameterJdbcTemplate
           etc

=> DriverManagerDataSource :-
    -> It is an implemented class of DataSource interface which is present in "org.springframework.jdbc.datasource" package
    -> It is used for :-
        1. Database configurations
        2. Driver loading
        3. Connection creation
           etc

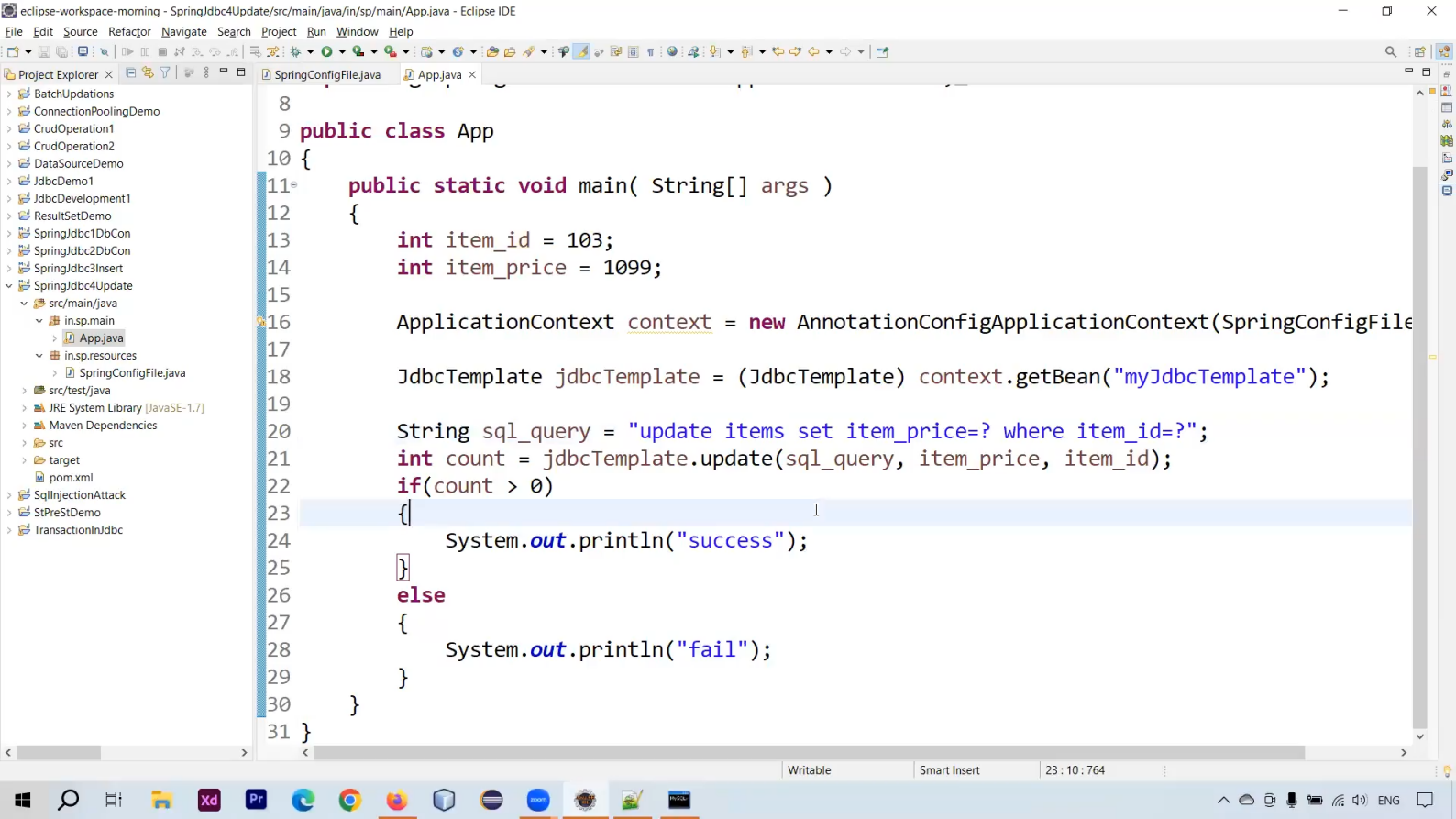

*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Batch Updations.txt
Transaction Management in JDBC.txt
DataSource: Connection Pooling.txt
new 1
=> JdbcTemplate :-
-> It is the central class in Spring-JDBC support classes
-> It is used for :-
1. Simplifies JDBC
= JdbcTemplate reduces boilerplate JDBC code making it readable and concise
2. Connection Management
= It manages connections, reducing the need for manual connection handling
3. Exception Handling :-
= Automatically translates the database exceptions into Spring's DataAccessExceptions which simplifies the error handling
4. SQL execution :-
= Executes SQL queries i.e. insert, update, delete, select etc
5. Parametrization :-
= Support both positional and named parameters in SQL queries
etc
-> Methods of JdbcTemplate :-
1. update(-)
= used for insert, update and delete SQL queries
2. query(-)
queryForList(-)
queryForMap(-)
queryForObject(-)
queryForRowSet()
= used for select SQL queries
3. execute(-)
= used for DDL (create, drop, alter etc) SQL queries
Normal text file
length : 1,754 lines : 48
Ln : 45 Col : 28 Pos : 1,647
Windows (CR LF)
UTF-8
INS
ENG

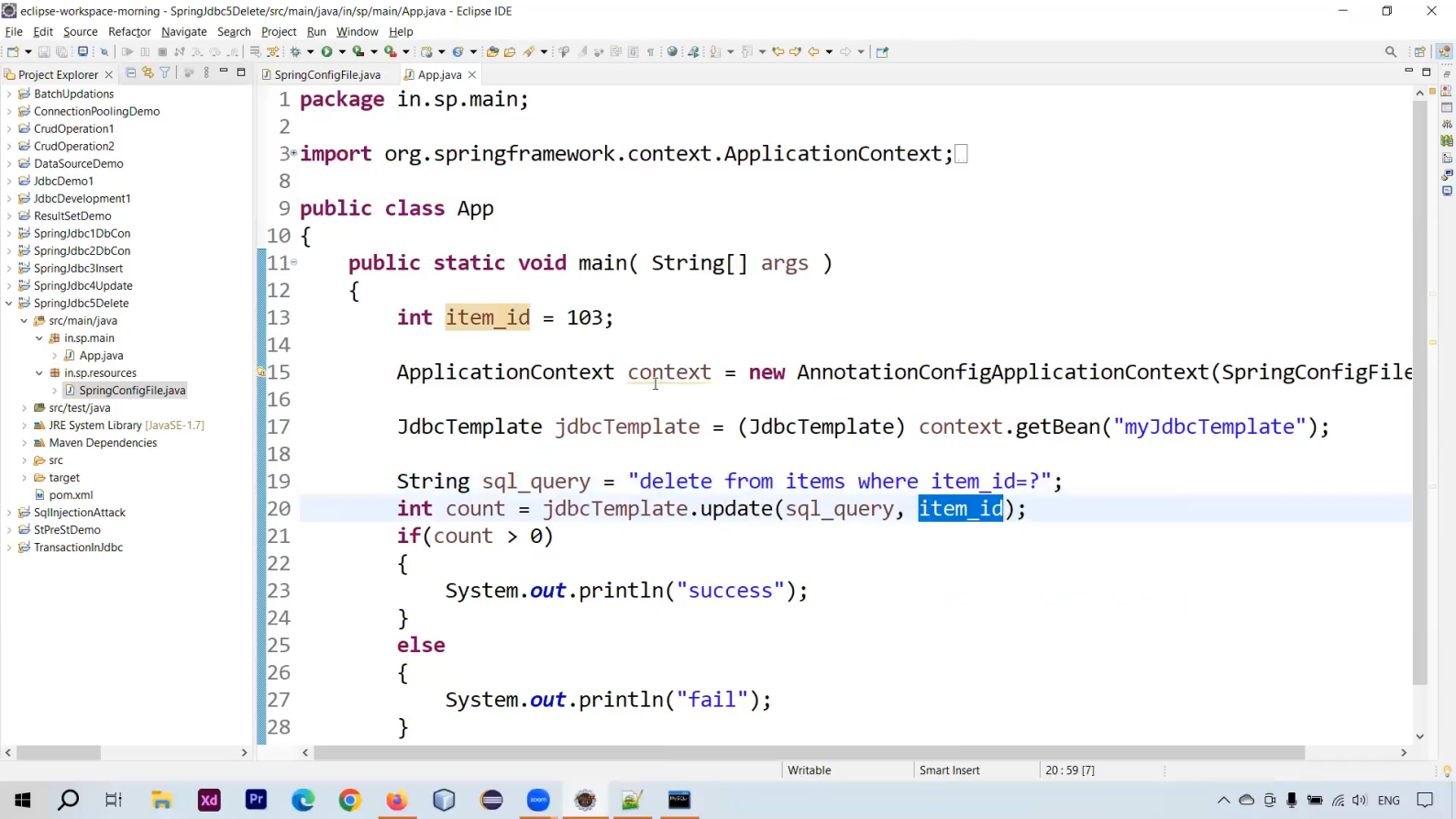
SpringConfigFile.java App.java

```
9  public class App
10 {
11     public static void main( String[] args )
12     {
13         int item_id = 101;
14         String item_name = "jeans";
15         int item_price = 899;
16
17         ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfigFile.class);
18
19         JdbcTemplate jdbcTemplate = (JdbcTemplate) context.getBean("myJdbcTemplate");
20
21         String sql_query = "insert into items values('"+item_id+"', '"+item_name+"', '"+item_price+"')";
22         int count = jdbcTemplate.update(sql_query);
23         if(count > 0)
24         {
25             System.out.println("success");
26         }
27         else
28         {
29             System.out.println("fail");
30         }
31     }
32 }
```

```
//------------------ way 2----------------------------
String sql_query = "insert into items values(?,?,?)";
int count = jdbcTemplate.update(sql_query, item_id, item_name, item_price);
if(count > 0)
{
    System.out.println("success");
}
else
{
    System.out.println("fail");
}
}
```

```
//------------------ way 3----------------------------
String sql_query = "insert into items values(?,?,?)";
int count = jdbcTemplate.update(sql_query, new Object[] {item_id, item_name, item_price});
if(count > 0)
{
    System.out.println("success");
}
else
{
    System.out.println("fail");
}
```


eclipse-workspace-morning - SpringJdbc4Update/src/main/java/in/sp/main/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
BatchUpdations
ConnectionPoolingDemo
CrudOperation1
CrudOperation2
DataSourceDemo
JdbcDemo1
JdbcDevelopment1
ResultSetDemo
SpringJdbc1DbCon
SpringJdbc2DbCon
SpringJdbc3Insert
SpringJdbc4Update
src/main/java
in.sp.main
App.java
in.sp.resources
SpringConfigFile.java
src/test/java
JRE System Library [JavaSE-1.7]
Maven Dependencies
src
target
pom.xml
SqlInjectionAttack
StPreStDemo
TransactionInJdbc
SpringConfigFile.java
App.java
8
9 public class App
10 {
11 public static void main(String[] args)
12 {
13 int item_id = 103;
14 int item_price = 1099;
15
16 ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfigFile
17
18 JdbcTemplate jdbcTemplate = (JdbcTemplate) context.getBean("myJdbcTemplate");
19
20 String sql_query = "update items set item_price=? where item_id=?";
21 int count = jdbcTemplate.update(sql_query, item_price, item_id);
22 if(count > 0)
23 {
24 System.out.println("success");
25 }
26 else
27 {
28 System.out.println("fail");
29 }
30 }
31 }
Writable
Smart Insert
23 : 10 : 764
ENG


eclipse-workspace-morning - SpringJdbc5Delete/src/main/java/in/sp/main/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
BatchUpdations
ConnectionPoolingDemo
CrudOperation1
CrudOperation2
DataSourceDemo
JdbcDemo1
JdbcDevelopment1
ResultSetDemo
SpringJdbc1DbCon
SpringJdbc2DbCon
SpringJdbc3Insert
SpringJdbc4Update
SpringJdbc5Delete
src/main/java
in.sp.main
App.java
in.sp.resources
SpringConfigFile.java
src/test/java
JRE System Library [JavaSE-1.7]
Maven Dependencies
src
target
pom.xml
SqlInjectionAttack
StPreStDemo
TransactionInJdbc
SpringConfigFile.java
App.java
1 package in.sp.main;
2
3 import org.springframework.context.ApplicationContext;
8
9 public class App
10 {
11 public static void main(String[] args)
12 {
13 int item_id = 103;
14
15 ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfigFile
16
17 JdbcTemplate jdbcTemplate = (JdbcTemplate) context.getBean("myJdbcTemplate");
18
19 String sql_query = "delete from items where item_id=?";
20 int count = jdbcTemplate.update(sql_query, item_id);
21 if(count > 0)
22 {
23 System.out.println("success");
24 }
25 else
26 {
27 System.out.println("fail");
28 }
Writable
Smart Insert
20 : 59 [7]
ENG

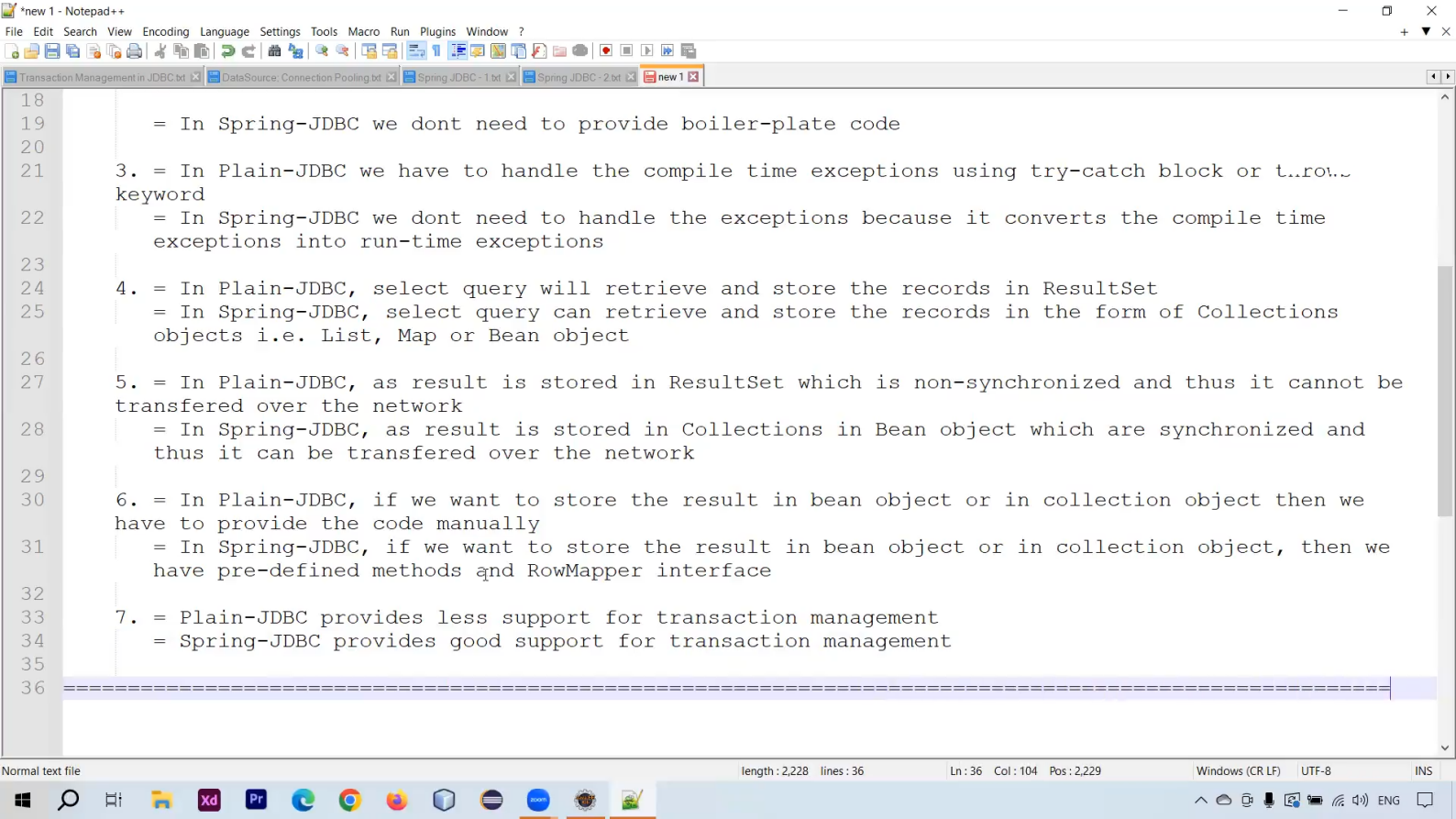
=> Difference between Plain-JDBC and Spring-JDBC :-

1. = In Plain-JDBC we have to get the connection object manually
   = In Spring-JDBC, connection object is automatically provided by JdbcTemplate class

2. = In Plain-JDBC we have following steps :-

   i. Load and register driver
      Class.forName("----");
   ii. Create connection object
      Connection con = DriverManager.getConnection("url", "username", "password");
   iii. Create Statement, PreparedStatement or CallableStatement object
      Statement st = con.createStatement();
   iv. Write and execute SQL query
      st.executeQuery("----sql query----");
   v. Close the resources
      st.close();
      con.close();
   -> NOTE : above 1,2,3 and 5th steps are always same but 4th step is variable. These common steps (1,2,3,5) are known as boiler-plate code which we have to write again and again

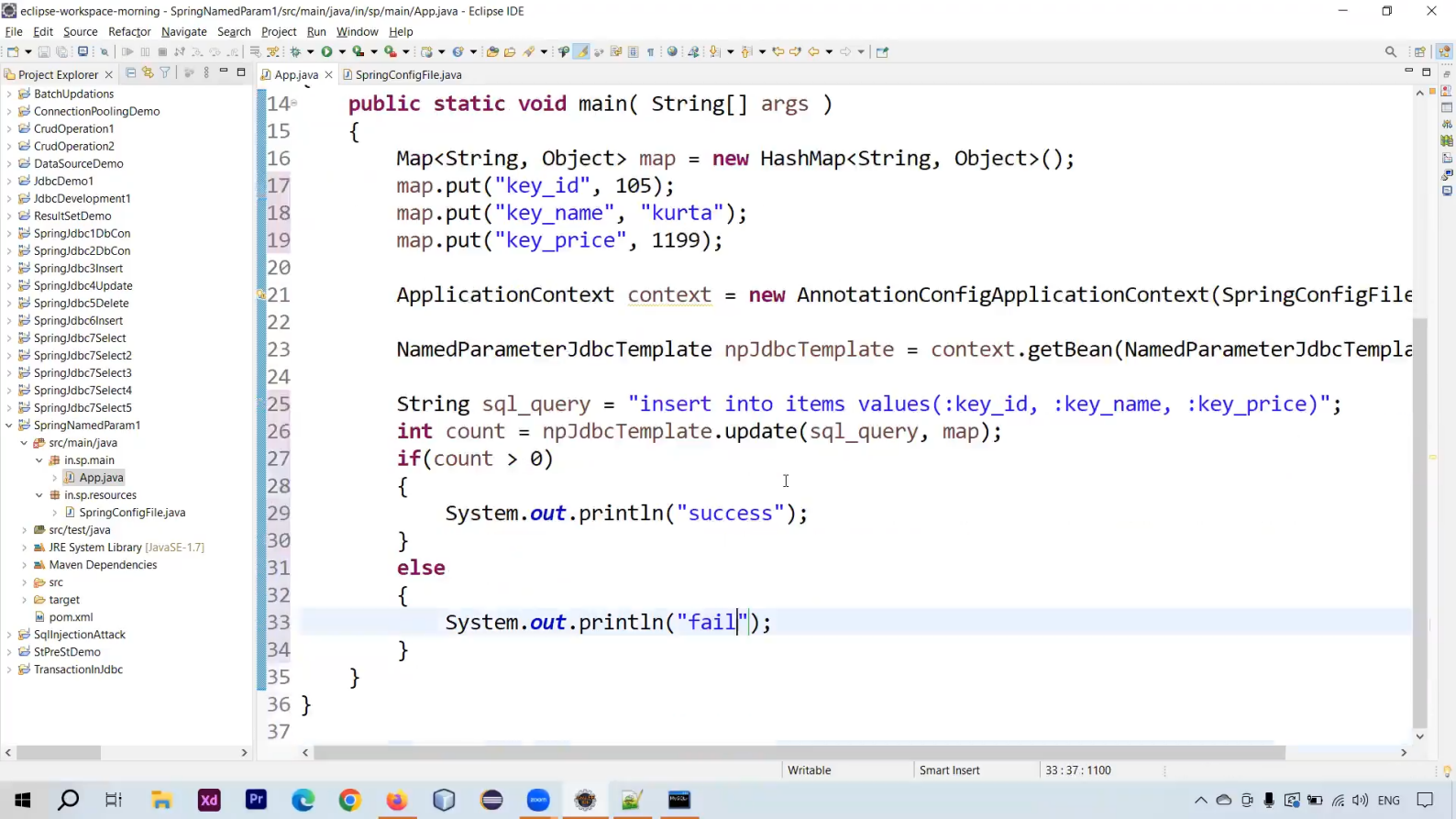
   = In Spring-JDBC we dont need to provide boiler-plate code

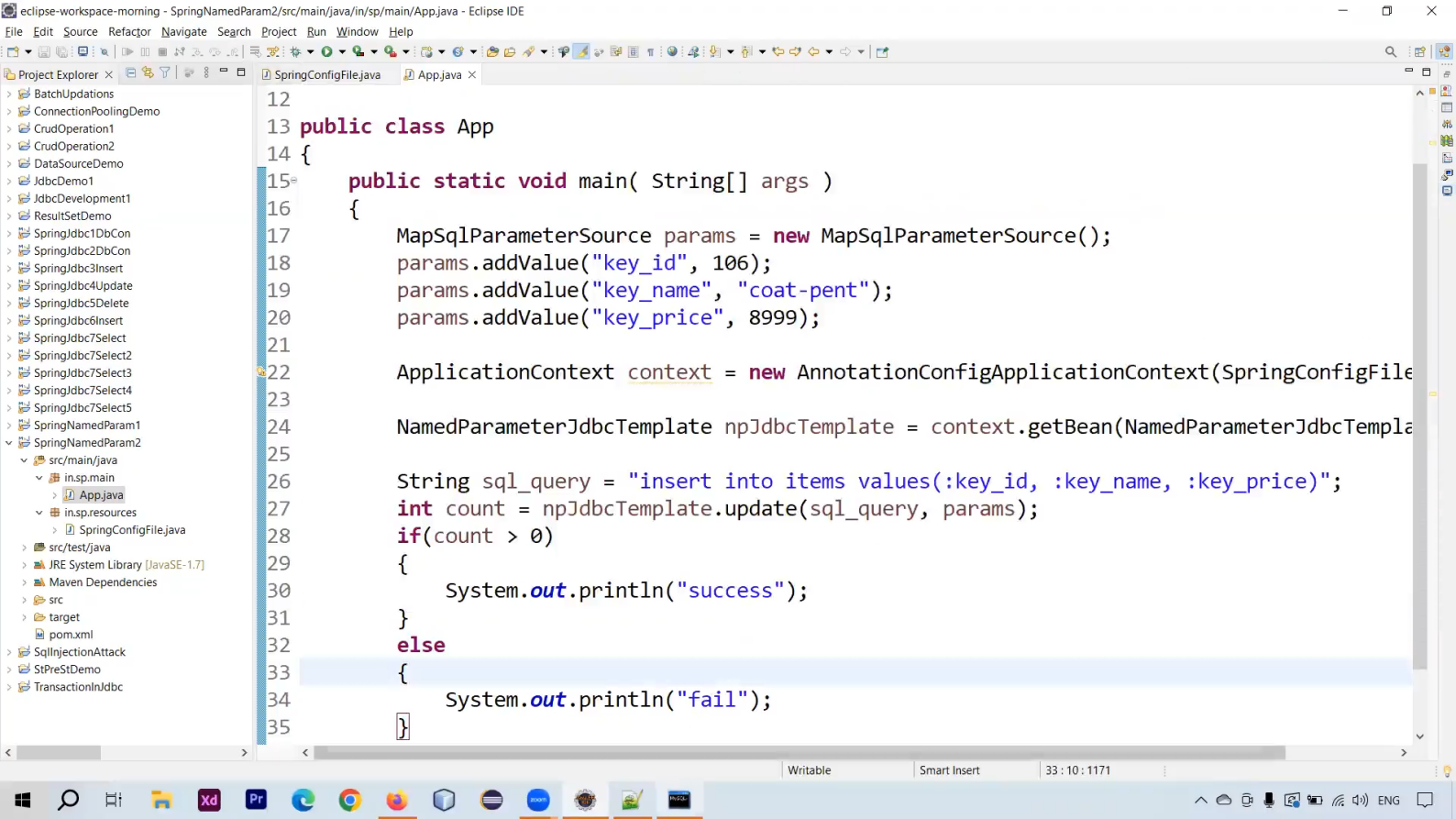

*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Transaction Management in JDBC.txt
DataSource: Connection Pooling.txt
Spring JDBC - 1.txt
Spring JDBC - 2.txt
new 1
= In Spring-JDBC we dont need to provide boiler-plate code
3. = In Plain-JDBC we have to handle the compile time exceptions using try-catch block or t...ro...
keyword
= In Spring-JDBC we dont need to handle the exceptions because it converts the compile time
exceptions into run-time exceptions
4. = In Plain-JDBC, select query will retrieve and store the records in ResultSet
= In Spring-JDBC, select query can retrieve and store the records in the form of Collections
objects i.e. List, Map or Bean object
5. = In Plain-JDBC, as result is stored in ResultSet which is non-synchronized and thus it cannot be
transfered over the network
= In Spring-JDBC, as result is stored in Collections in Bean object which are synchronized and
thus it can be transfered over the network
6. = In Plain-JDBC, if we want to store the result in bean object or in collection object then we
have to provide the code manually
= In Spring-JDBC, if we want to store the result in bean object or in collection object, then we
have pre-defined methods and RowMapper interface
7. = Plain-JDBC provides less support for transaction management
= Spring-JDBC provides good support for transaction management
==========
Normal text file
length : 2,228 lines : 36
Ln : 36 Col : 104 Pos : 2,229
Windows (CR LF)
UTF-8
INS
ENG

=> NamedParameterJdbcTemplate :-
-> It is the class which is provided by Spring framework as the part of Spring-JDBC module
-> It provides "higher-level abstraction" and more convenient way to work with SQL queries
= A "higher-level abstraction" means that it provides user-friendly way to interact wit.. th..
system as compared to lower-level abstraction

-> Why we need NamedParameterJdbcTemplate :-
= In JdbcTemplate, we provide "Positional Parameters" which is not user-friendly because if there are a lot of positional parameters then it will create confusion. To remove this confusion we use NamedParameterJdbcTemplate in which we use "Named Parameters"

-> We can provide "Named Parameters" by 2 ways :-
1. By using Map <interface>
= We can use any implemented class of Map interface for eg. HashMap etc
2. By using SqlParameterSource <interface>
= We can use below implemented classes :-
a. MapSqlParameterSource
b. BeanPropertySqlParameterSource


eclipse-workspace-morning - SpringNamedParam1/src/main/java/in/sp/main/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
BatchUpdations
ConnectionPoolingDemo
CrudOperation1
CrudOperation2
DataSourceDemo
JdbcDemo1
JdbcDevelopment1
ResultSetDemo
SpringJdbc1DbCon
SpringJdbc2DbCon
SpringJdbc3Insert
SpringJdbc4Update
SpringJdbc5Delete
SpringJdbc6Insert
SpringJdbc7Select
SpringJdbc7Select2
SpringJdbc7Select3
SpringJdbc7Select4
SpringJdbc7Select5
SpringNamedParam1
src/main/java
in.sp.main
App.java
in.sp.resources
SpringConfigFile.java
src/test/java
JRE System Library [JavaSE-1.7]
Maven Dependencies
src
target
pom.xml
SqlInjectionAttack
StPreStDemo
TransactionInJdbc
App.java
SpringConfigFile.java
14 public static void main(String[] args)
15 {
16 Map<String, Object> map = new HashMap<String, Object>();
17 map.put("key_id", 105);
18 map.put("key_name", "kurta");
19 map.put("key_price", 1199);
20
21 ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfigFile
22
23 NamedParameterJdbcTemplate npJdbcTemplate = context.getBean(NamedParameterJdbcTempla
24
25 String sql_query = "insert into items values(:key_id, :key_name, :key_price)";
26 int count = npJdbcTemplate.update(sql_query, map);
27 if(count > 0)
28 {
29 System.out.println("success");
30 }
31 else
32 {
33 System.out.println("fail");
34 }
35 }
36 }
37
Writable
Smart Insert
33 : 37 : 1100
ENG


eclipse-workspace-morning - SpringNamedParam2/src/main/java/in/sp/main/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
BatchUpdations
ConnectionPoolingDemo
CrudOperation1
CrudOperation2
DataSourceDemo
JdbcDemo1
JdbcDevelopment1
ResultSetDemo
SpringJdbc1DbCon
SpringJdbc2DbCon
SpringJdbc3Insert
SpringJdbc4Update
SpringJdbc5Delete
SpringJdbc6Insert
SpringJdbc7Select
SpringJdbc7Select2
SpringJdbc7Select3
SpringJdbc7Select4
SpringJdbc7Select5
SpringNamedParam1
SpringNamedParam2
src/main/java
in.sp.main
App.java
in.sp.resources
SpringConfigFile.java
src/test/java
JRE System Library [JavaSE-1.7]
Maven Dependencies
src
target
pom.xml
SqlInjectionAttack
StPreStDemo
TransactionInJdbc
SpringConfigFile.java
App.java
12
13 public class App
14 {
15 public static void main(String[] args)
16 {
17 MapSqlParameterSource params = new MapSqlParameterSource();
18 params.addValue("key_id", 106);
19 params.addValue("key_name", "coat-pent");
20 params.addValue("key_price", 8999);
21
22 ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfigFile
23
24 NamedParameterJdbcTemplate npJdbcTemplate = context.getBean(NamedParameterJdbcTempla
25
26 String sql_query = "insert into items values(:key_id, :key_name, :key_price)";
27 int count = npJdbcTemplate.update(sql_query, params);
28 if(count > 0)
29 {
30 System.out.println("success");
31 }
32 else
33 {
34 System.out.println("fail");
35 }
Writable
Smart Insert
33 : 10 : 1171
ENG

```
MapSqlParameterSource params = new MapSqlParameterSource();
params.addValue("key_id", 106);
params.addValue("key_name", "coat-pent");
params.addValue("key_price", 8999);

MapSqlParameterSource params = new MapSqlParameterSource();
params.addValue("key_id", 107)
        .addValue("key_name", "top")
        .addValue("key_price", 999);
```

eclipse-workspace-morning - SpringNamedParam3/src/main/java/in/sp/main/App.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

SpringConfigFile.java | App.java | Items.java

```
15      {
16          Items items = new Items();
17          items.setItemid(108);
18          items.setItemname("lehnga");
19          items.setItemprice(20000);
20
21          BeanPropertySqlParameterSource params = new BeanPropertySqlParameterSource(items);
22
23          ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfigFile.class);
24
25          NamedParameterJdbcTemplate npJdbcTemplate = context.getBean(NamedParameterJdbcTemplate.class);
26
27          String sql_query = "insert into items values(:itemid, :itemname, :itemprice)";
28          int count = npJdbcTemplate.update(sql_query, params);
29          if(count > 0)
30          {
31              System.out.println("success");
32          }
33          else
34          {
35              System.out.println("fail");
36          }
37      }
38  }
```

Writable | Smart Insert | 24 : 9 : 882

ENG

App.java SpringConfigFile.java

```java
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

import in.sp.resources.SpringConfigFile;

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfigFile.class);

        JdbcTemplate jdbcTemplate = context.getBean(JdbcTemplate.class);

        String query1 = "insert into items values(101, 'shirt', 500)";
        String query2 = "insert into items values(102, 't-shirt', 600)";
        String query3 = "insert into items values(103, 'jeans', 700)";

        int[] count = jdbcTemplate.batchUpdate(query1, query2, query3);
        for(int i : count)
        {
            System.out.println(i+" : success");
        }
    }
}
```

```
//------------------way 2-----------------------------------------------
String query1 = "insert into items values(101, 'shirt', 500)";
String query2 = "insert into items values(102, 't-shirt', 600)";
String query3 = "insert into items values(103, 'jeans', 700)";

String[] queries = {query1, query2, query3};

int[] count = jdbcTemplate.batchUpdate(queries);
for(int i : count)
{
    System.out.println(i+" : success");
}
```

```
=> Spring Batch Updates :-
    -> It is a batch of updates grouped together and sent to the database in one batch rather then
    sending them one by one
    -> Advantages :-
        = Application performance will be improved
        = Network traffic will be reduced
    -> Disadvantages :-
        = It can only be used for "insert, update and delete" SQL query, not for select SQL query
        = If any single SQL query gets an error then it will disturb the flow of program

    -> In Spring-JDBC, for batch update we have one method :-
        = batchUpdate(-)
```

```
public class App
{
    public static void main( String[] args )
    {
```

```
Items item1 = new Items();
item1.setItemid(101);
item1.setItemname("shirt");
item1.setItemprice(500);

Items item2 = new Items();
item2.setItemid(102);
item2.setItemname("t-shirt");
item2.setItemprice(700);

Items item3 = new Items();
item3.setItemid(103);
item3.setItemname("jeans");
item3.setItemprice(900);


final List<Items> items_list = new ArrayList<Items>();
items_list.add(item1);
items_list.add(item2);
items_list.add(item3);
```

item1: 101 shirt 500

item2: 102 t-shirt 700

item3:

items_list: [101 shirt 500] [102 t-shirt 700] [ ]

```
    }
}
```

```
ApplicationContext context = new ----------
JdbcTemplate jdbcTemplate = context.getBean(JdbcTemplate.class);

String sql_query = "insert into items values(?,?,?)";
jdbcTemplate.batchUpdate(sql_query, new BatchPreparedStatementSetter() {

    @Override
    public void setValues(PreparedStatement ps, int index) throws SQLException
    {
        Items item = items_list.get(index);

        ps.setInt(1, item.getItemid());
        ps.setString(2, item.getItemname());
        ps.setInt(3, item.getItemprice());
    }

    @Override
    public int getBatchSize()
    {
        return items_list.size();
    }
});
```

3

```java
@Bean
public DataSource createDataSourceObj()
{
    HikariConfig config = new HikariConfig();
    config.setJdbcUrl("jdbc:mysql://localhost:3306/shopping_app");
    config.setUsername("root");
    config.setPassword("root");
    config.setMaximumPoolSize(5);

    HikariDataSource hikariDataSource = new HikariDataSource(config);
    return hikariDataSource;
}
```

eclipse-workspace-morning - SpringConnectionPooling1/src/main/java/in/sp/main/App.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

App.java SpringConfigFile.java

```
12 public class App
13 {
14     public static void main( String[] args ) throws Exception
15     {
16         ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfigFile.class);
17
18         DataSource dataSource = context.getBean(DataSource.class);
19
20         Connection con1 = dataSource.getConnection();
21         Connection con2 = dataSource.getConnection();
22         Connection con3 = dataSource.getConnection();
23         Connection con4 = dataSource.getConnection();
24         Connection con5 = dataSource.getConnection();
25
26         con1.close();
27         con2.close();
28
29         Connection con6 = dataSource.getConnection();
30         Connection con7 = dataSource.getConnection();
31
32         System.out.println("success");
33     }
34 }
35
```

Writable Smart Insert 23 : 19 : 736

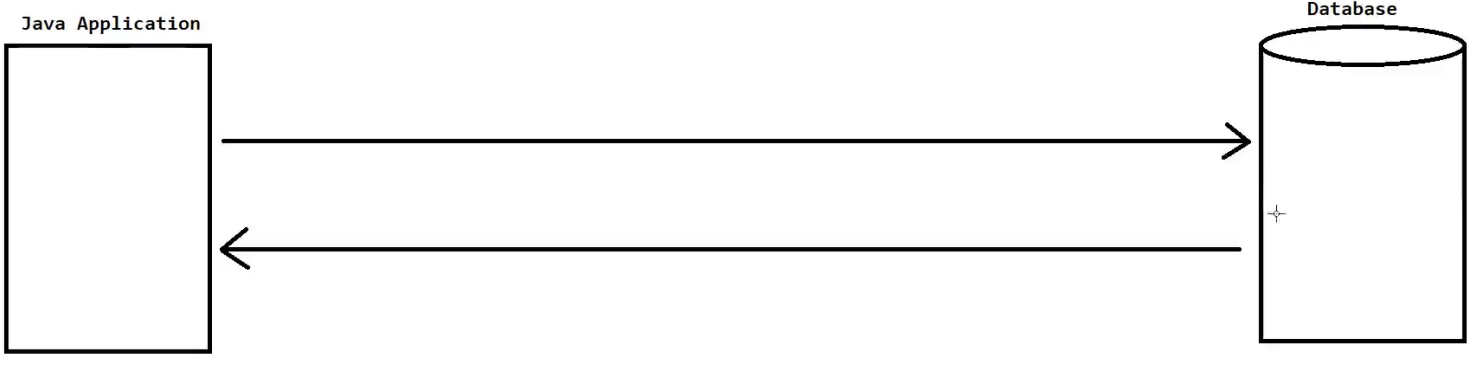
ENG

=> Design Patterns :-
    -> Design patterns are established best practices and reusable templates for solving common software design problem
        = "established best practices" means that these practices have been tried, tested and widely recognized as effective solutions
    -> They offer structured approaches to create relationssips and interactions between classes and objects, promoting efficient, maintanable and scalable code
    -> For eg.
        = Singleton Design Pattern
        = Factory Design Pattern
        = Abstract Factory Design Pattern
        = Prototype Design Pattern
        = DAO Design Pattern
        = MVC Design Pattern
        = Dependency Injection Design Pattern
          etc

    -> NOTE : There is one book for design patterns i.e. "Design Patterns : Elements of Reusable Object-Oriented Software"

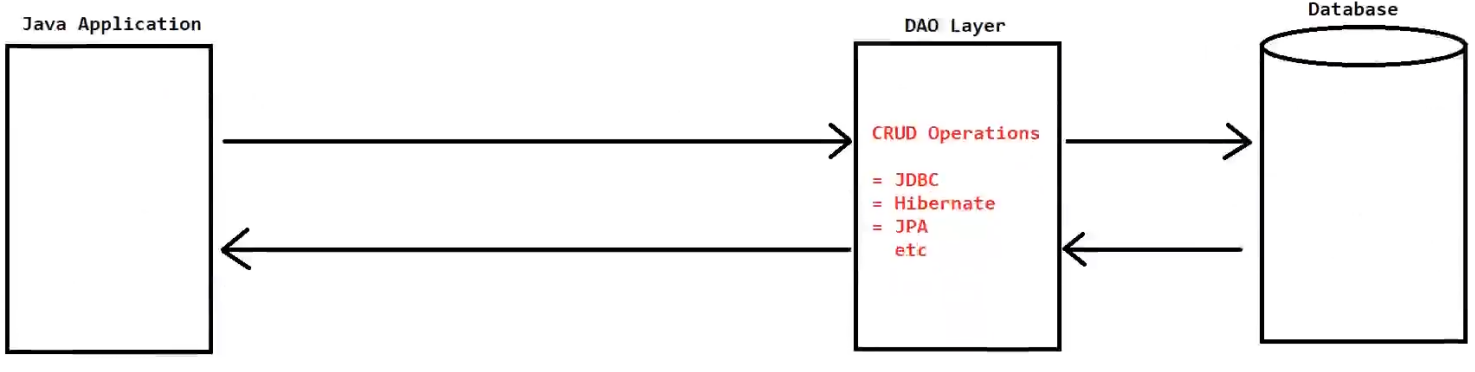
==========

This is Old Approach

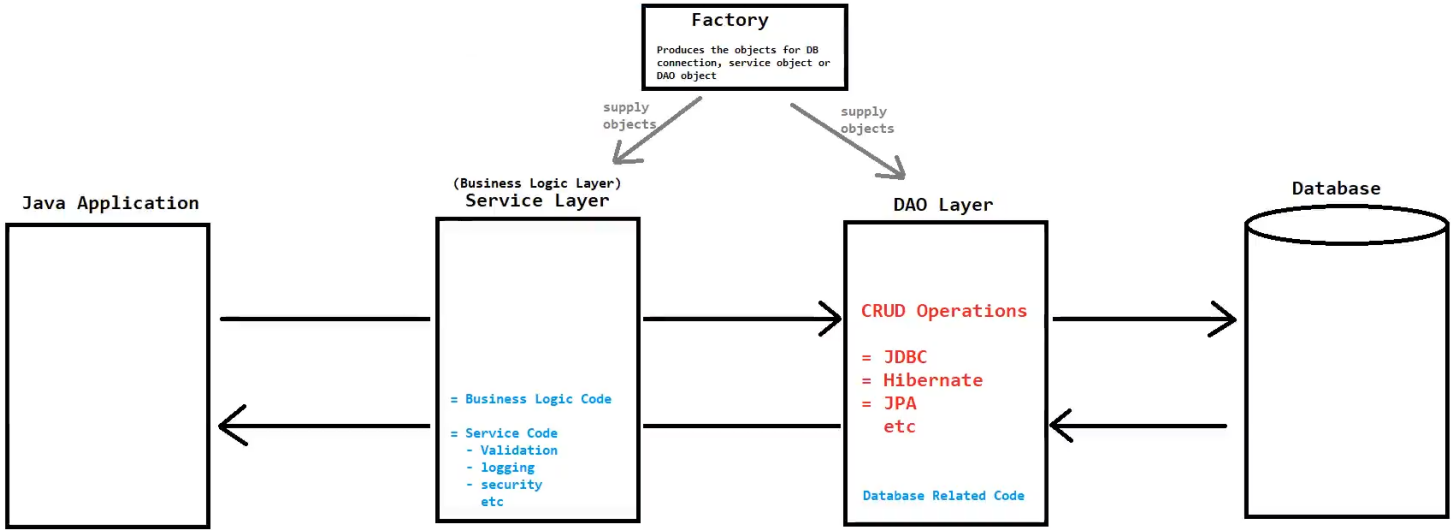


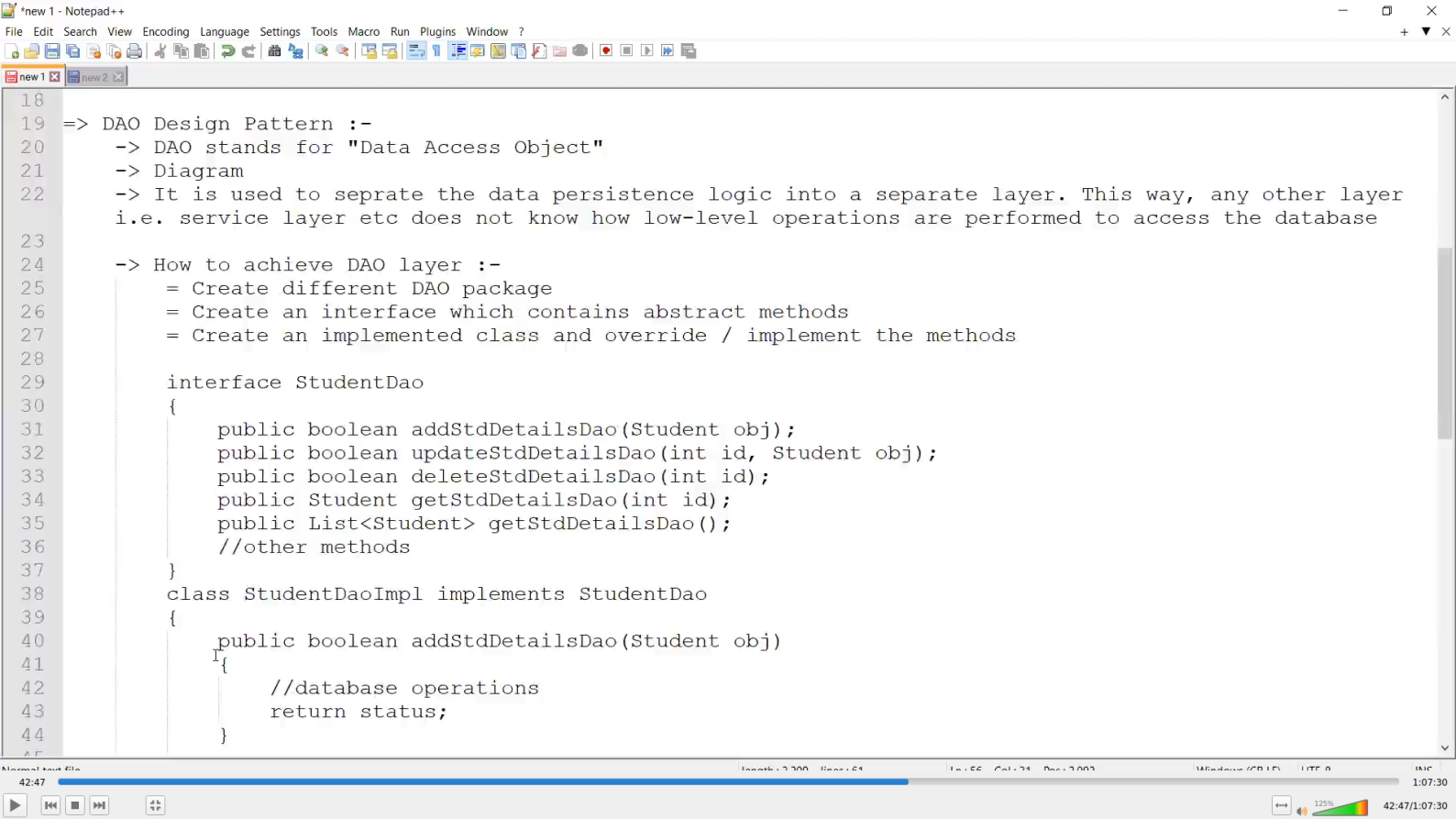
Simple DAO Design Pattern

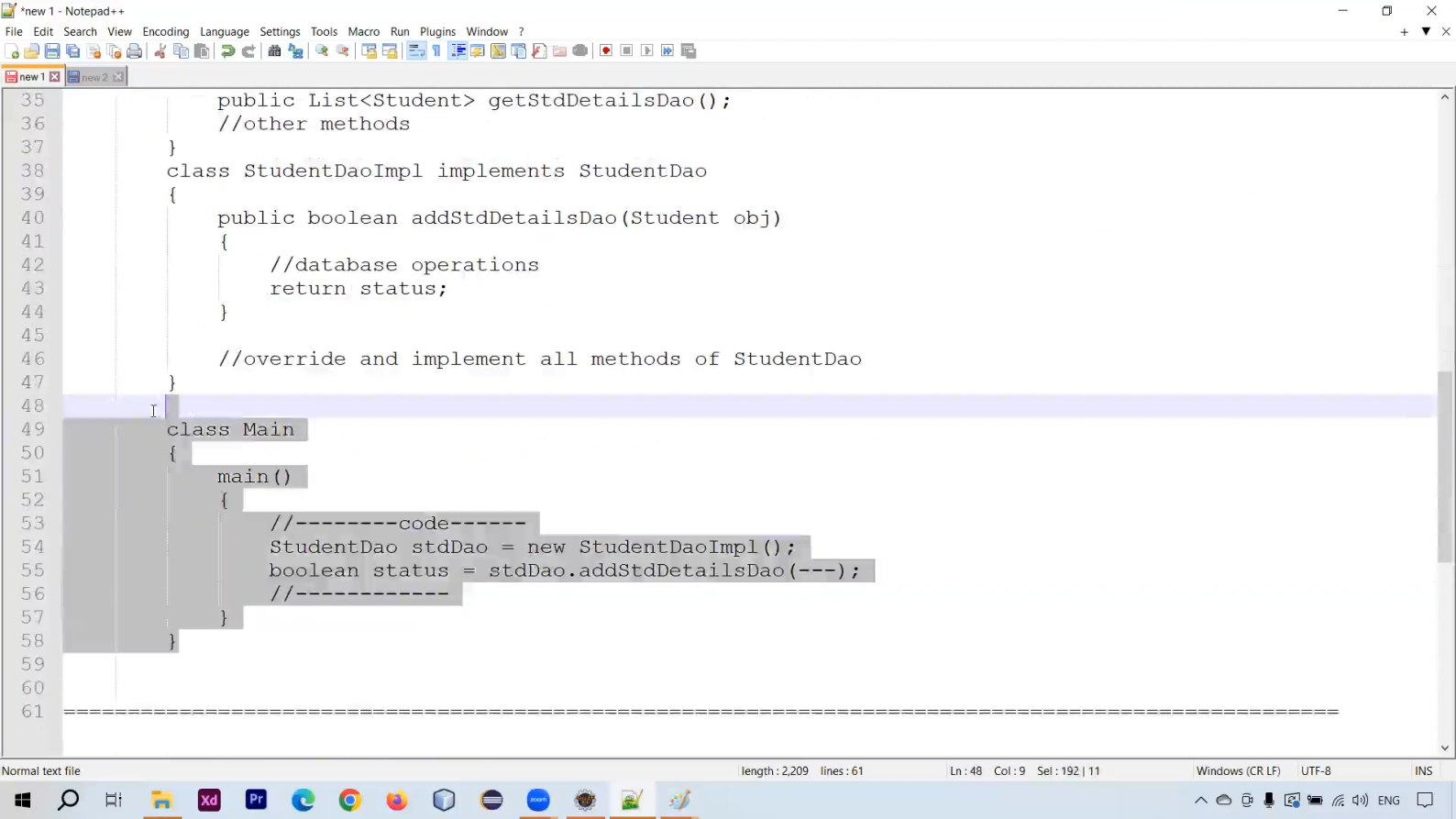
This is New Approach



This is Modular Design Pattern




*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 1
new 2
=> DAO Design Pattern :-
-> DAO stands for "Data Access Object"
-> Diagram
-> It is used to seprate the data persistence logic into a separate layer. This way, any other layer i.e. service layer etc does not know how low-level operations are performed to access the database
-> How to achieve DAO layer :-
= Create different DAO package
= Create an interface which contains abstract methods
= Create an implemented class and override / implement the methods
interface StudentDao
{
public boolean addStdDetailsDao(Student obj);
public boolean updateStdDetailsDao(int id, Student obj);
public boolean deleteStdDetailsDao(int id);
public Student getStdDetailsDao(int id);
public List<Student> getStdDetailsDao();
//other methods
}
class StudentDaoImpl implements StudentDao
{
public boolean addStdDetailsDao(Student obj)
{
//database operations
return status;
}
42:47
1:07:30
42:47/1:07:30


*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 1
new 2
public List<Student> getStdDetailsDao();
//other methods
}
class StudentDaoImpl implements StudentDao
{
public boolean addStdDetailsDao(Student obj)
{
//database operations
return status;
}
//override and implement all methods of StudentDao
}
class Main
{
main()
{
//--------code------
StudentDao stdDao = new StudentDaoImpl();
boolean status = stdDao.addStdDetailsDao(---);
//------------
}
}
Normal text file
length : 2,209 lines : 61
Ln : 48 Col : 9 Sel : 192 | 11
Windows (CR LF)
UTF-8
INS
ENG

Untitled - Paint

File Home View

Cut Copy Paste Clipboard

Crop Resize Rotate Select Image

Brushes Tools

Outline Fill Shapes

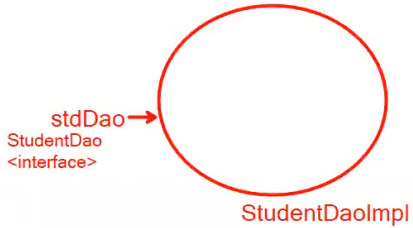
Size

Color 1 Color 2 Edit colors Edit with Paint 3D Colors

```
interface StudentDao
{
    public boolean addStdDetailsDao(Student obj);
    public boolean updateStdDetailsDao(int id, Student obj);
    public boolean deleteStdDetailsDao(int id);
    public Student getStdDetailsDao(int id);
    public List<Student> getStdDetailsDao();
    //other methods
}
class StudentDaoImpl implements StudentDao
{
    public boolean addStdDetailsDao(Student obj)
    {
        //database operations
        return status;
    }

    //override and implement all methods of StudentDao
}
```

```
class Main
{
    main()
    {
        //--------code------
        StudentDao stdDao = new StudentDaoImpl();
        boolean status = stdDao addStdDetailsDao(---);
        //------------
    }
}
```



1709, 652px 1858 × 810px 100%

ENG

=> Spring DAO :-
    -> Spring DAO is a concept (design pattern) to work with database technologies i.e. JDBC, Hibernate, JPA etc in an easy way
    -> We can achieve DAO design pattern in spring by 3 ways :-
        1. By straight-forward and traiditional way
        2. By extending DAO support classes
        3. By using Annotations

=> By straight-forward and traiditional way :-
    -> This approach involves the creationof custom classes to perform data access operations in a traiditional way without relying on spring pre-defined classes or annotations

---

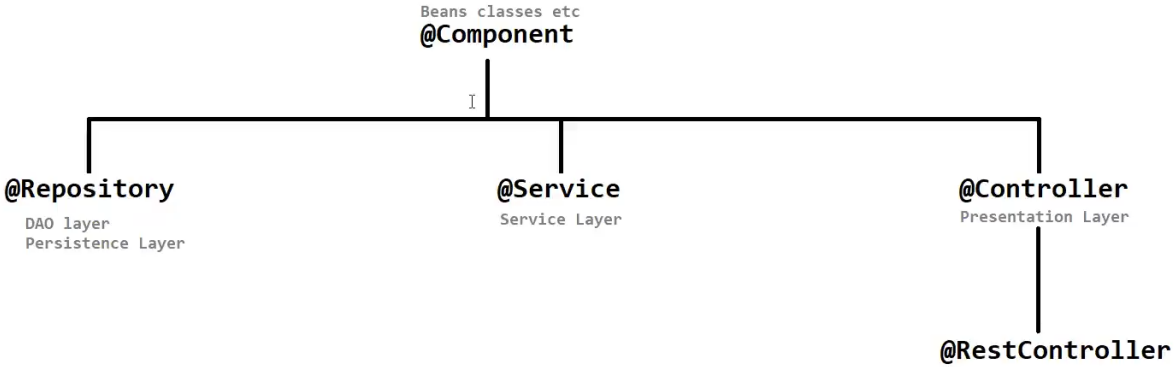
=> By extending DAO support classes :-

-> Spring has provided some DAO support classes (XxxDaoSupport) to simplify certain aspects of DAO development

-> Spring has provided many DAO support classes and 2 main classes are :-

1. JdbcDaoSupport
   = public final void setDataSource(DataSource dataSource){ - }
   = public final void setJdbcTemplate(JdbcTemplate jdbcTemplate) { - }
   = public final JdbcTemplate getJdbcTemplate() { - }
   etc
2. NamedParameterJdbcDaoSupport

-> NOTE : This is not a standalone approach because we still need to extend and customize the classes to create our specific DAO's


Beans classes etc
@Component
@Repository
DAO layer
Persistence Layer
@Service
Service Layer
@Controller
Presentation Layer
@RestController

=> By using Annotations :-
    -> Spring provides some annotations which can simplify the configuration and make our DAO's more concise and declative
    -> Main annotations are :-
        1. @Repository
        2. @Transactional

        3. @Service
        4. @Autowired

=> @Repository :-
    -> It is used with DAO layer
    -> It is a specialization of @Component

=> @Transactional :-
    -> It is used for transaction management

=> @Service :-
    -> It is used with service layer
    -> it is a specialization of @Component

(iv) SpringConfig.java → It will annotated by @Configuration
@ComponentScan (basePackages = "mee")

ye iske andar ka sara class
class bean automatically bana
dega.

It has only two work.

① Create the Bean of DataSource

② Create the Bean of JdbcTemplate

And jitne bhi class hai sara ka Object spring
handle kr lega.

(v) Main-class → same hoga
but Student ka Object bana
kr Set kr deya
usme value

& Baki sara cheez same rahega.

(III) Spring config.

```
@Bean
public DataSource mydatasource() {
    DriverManagerDataSource datasource = .....
    { connection for database
    return datasource;
}
```

for connection

```
@Bean.
public JDBC Template jdbcTemplate() {
    jdbc Temp jbcte = new ...;
    jdbctemplate.set( my Datasource() );
    return jdbc ;
}
```

then jdbc data daal payga Sql mein.

Same @ Student DAO , @ Student , @ Student Servi.

(IV) Service class. → private StudentDAO StdDao;
↳ setter method for StudentDAO;

```
public boolean addStudentService( Student student ) {
    boolean Status = StudentDAO.addStdDetail( Student);
    return Status;
}
```

(V) Main :- App.jav

```
boolean Status = Service.addStudentServi (Student);
if( Status ) {
    S.O.p("Succesn");
}
else
    S.O.p( "Fail");
```

Flow → App.java to Servi.ja to StdDao
Status Status

Annotation pat (Spring) — DAO pattern

Working

(1) → we dont need to create the bean by our self it is done by spring

→ in Annotation we use @Service, @Repository, @Autowired

@componentscan inbuilt rehta hai.

→ (1) bean → as usual

(II) Student DAO Imp → we annotate it with @Repository.

↳ @Autowired

public JdbcTemplate jdbcTemplate;

Then spring will handle the object and inject into it.

public void setJdbcTemplate (JdbcTemplate jdbcTemplate) { ... }

(III) Spring Service Imp → we annotate it with @Service

@Autowired

public private StudentDAO studentDAO;

public void setStudentDAO (StudentDAO studentDAO) { ... }

Simple Spring DAO Pattern

Working of DAO pattern

① → bean → Student → spring will handle.
- ↳ name
- ↳ age
- ↳ Marks

② interface → StudentDao → Method declaration
↳ public boolean addStdDetails (Student student);

It takes student object as param to get Name, age and marks.

implementation class → StudentImp →

```
private JdbcTemplate jdbcTemplate;
```
↳ Setter method of this

```
boolean status = false;
boolean addStdDetails (Student student){
    String sql_query = " . . . . ";
    int count = jdbc template. update ( Sql_Qur , Student.getNam
                                        . . . . );
    if ( count > 0) {
        Status = true;
    }
    else
        status = false;
    }
    return Stata
}
```

DAO Pattern

Using JDBC.

Working of DAO pattern

(I) Bean class.
- name
- age
- marks.

→ object → spring will handle. Variable handle ker liya.

(II) → DAO (Interface) → Method Rahega sirf

boolean addStudDetails (Student student);

taki wo set ker paye value.

DAO (Implementation class) → It will Return the Status.

```
boolean addStudDetails (Student stud) {
    (* object null jayega)
    try {
        Connection con = DbConnection.getConnection();
        Prepared Statement ....
        ps.setString(1, student.getName());   // beans se liya object. / variable null jay
        int count = prepare stat.executeUpdate();
        if (count > 0) {
            status = true
        }
        else
            status = false
        return status;   // last mein status
```

(III) DB Connection class → It builds the connection b/w the database.

↳ It return the connection object.

```
public static Connection con;
public static Connection getConnection() {
    try {
```

Connection to database

}
catch (exception e) {

e.printStackTrace();

}

return con; ye Connection ka object return kr raha hai.

(iv) Main class.

```
Student std2 = new Student();
Std.setName("Payush");
Std.setAge(21);
Std.setMarks(95);

StudentDAO stddao = new StudentImpDAO();
interface                    class.

int Status = stddao.addStudentDetails(Student);

if(Status)<0.
    S.O.P("Success");
else
    S.O.P("Fail");
```

}