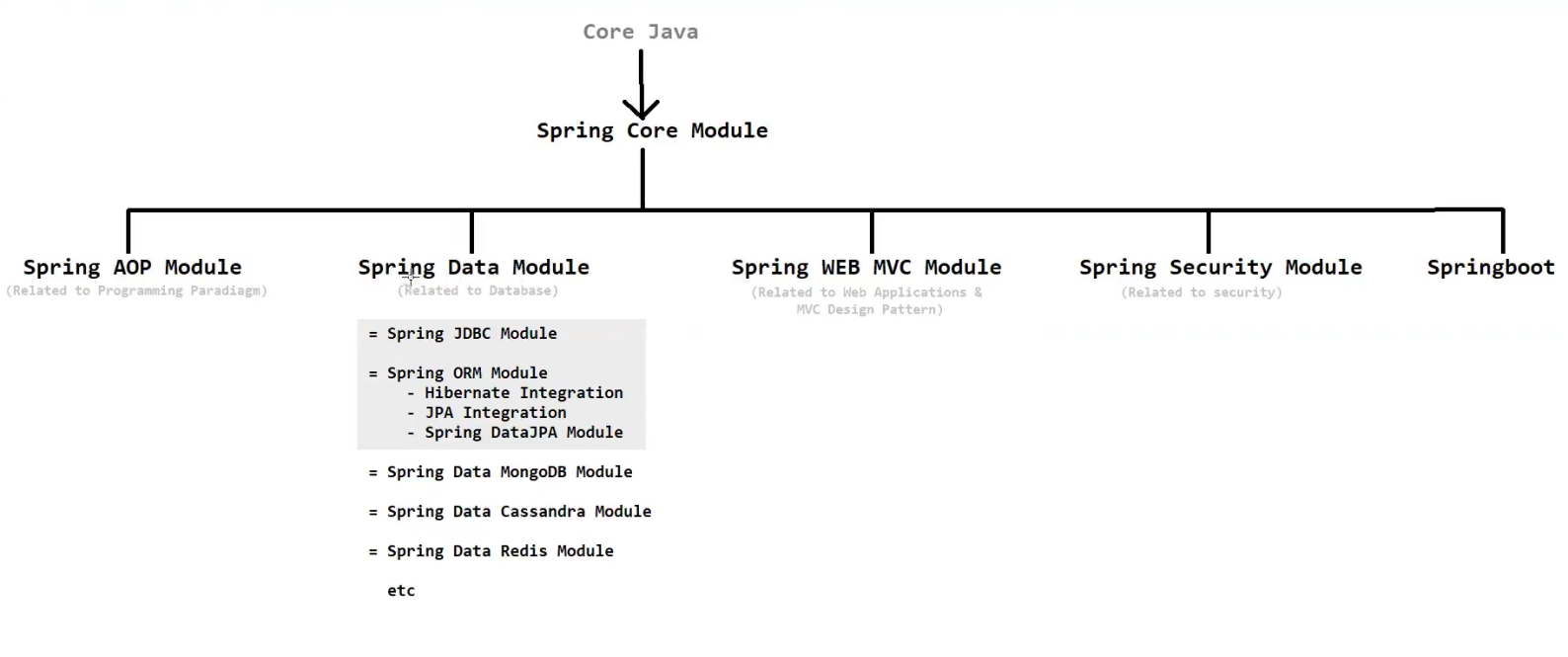

Core Java
Spring Core Module
Spring AOP Module
(Related to Programming Paradiagm)
Spring Data Module
(Related to Database)
= Spring JDBC Module
= Spring ORM Module
- Hibernate Integration
- JPA Integration
- Spring DataJPA Module
= Spring Data MongoDB Module
= Spring Data Cassandra Module
= Spring Data Redis Module
etc
Spring WEB MVC Module
(Related to Web Applications & MVC Design Pattern)
Spring Security Module
(Related to security)
Springboot

=> Database :-
-> A database is an organized collection of structured information or data typically stored electronically in computer system
-> A database is usually controlled by "Database Management System (DBMS)"
-> Few examples of DBMS are :-
- MySQL
- Microsoft SQL Server
- PostgreSQL
- SQLite
- MongoDB
- Cassandra
- DB2
etc

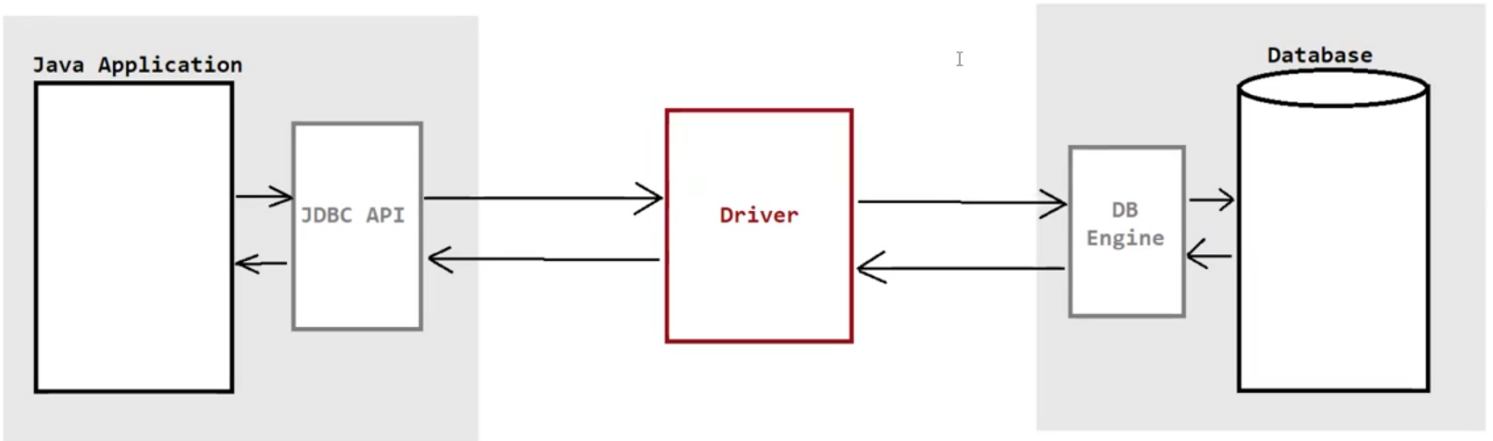
-> To store the data in database we have to use one query langauge i.e. SQL, Oracle etc

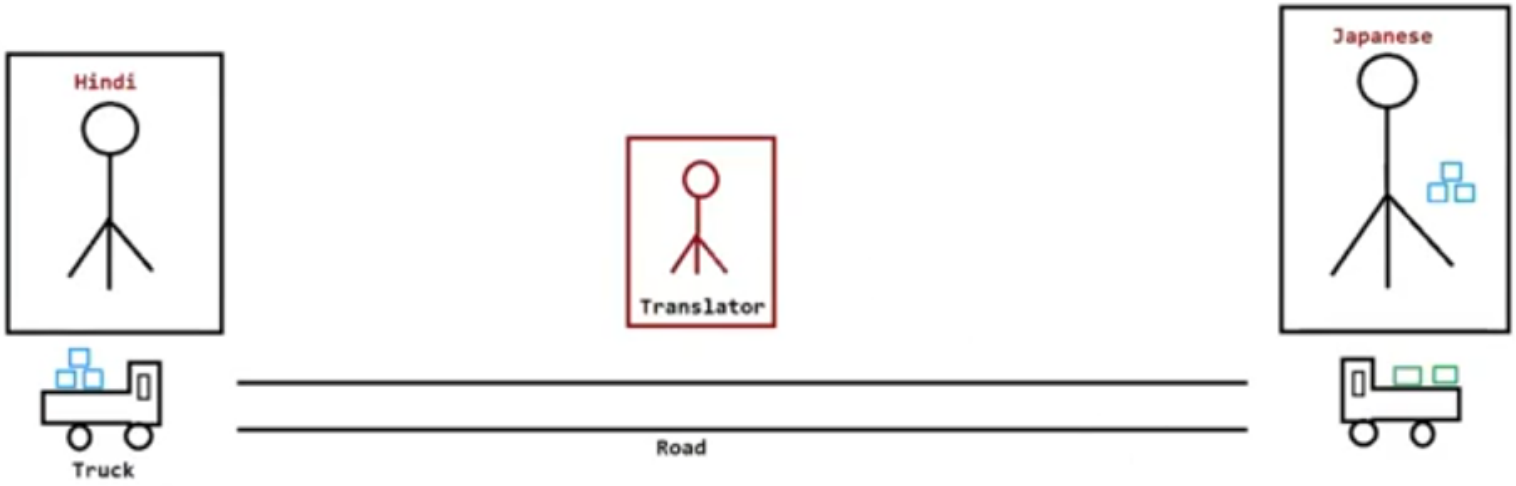
==========

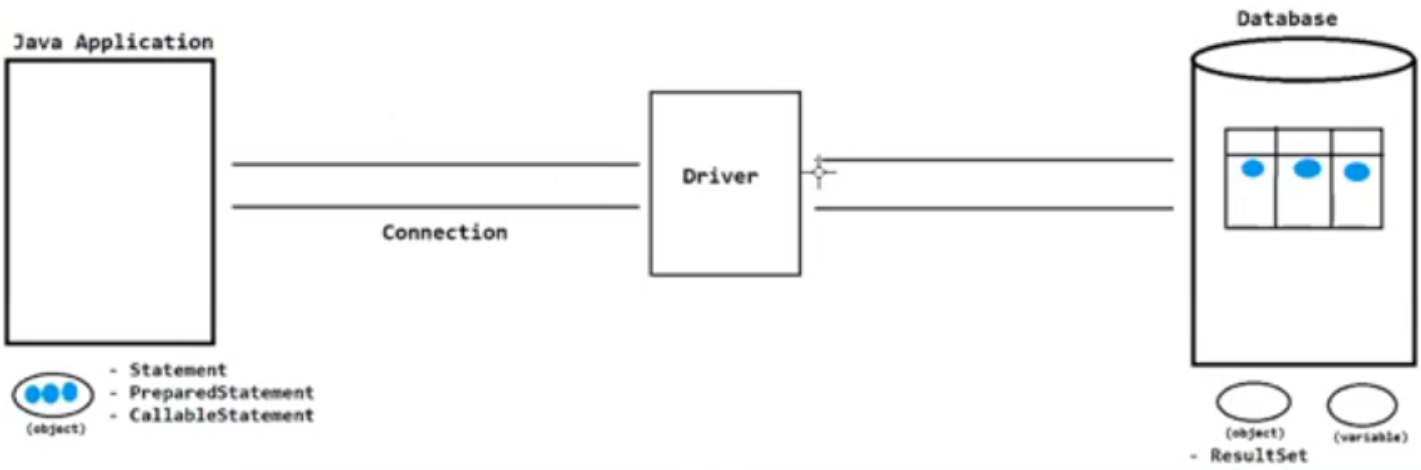
JDBC API

"java.sql" package		"javax.sql" package	
Interfaces	Classes	Interfaces	Classes
= Driver	= DriverManager	= DataSource	= RowSetEvent
= Connection	= Date	= RowSet	= ConnectionEvent
= Statement	= Time	= RowSetListener	etc
= PreparedStatement	= TimeStamp	= ConnectionEventListener	
= CallableStatement	= Types	etc	
= ResultSet	etc		
etc			

Working of JDBC




Hindi
Translator
Japanese
Truck
Road

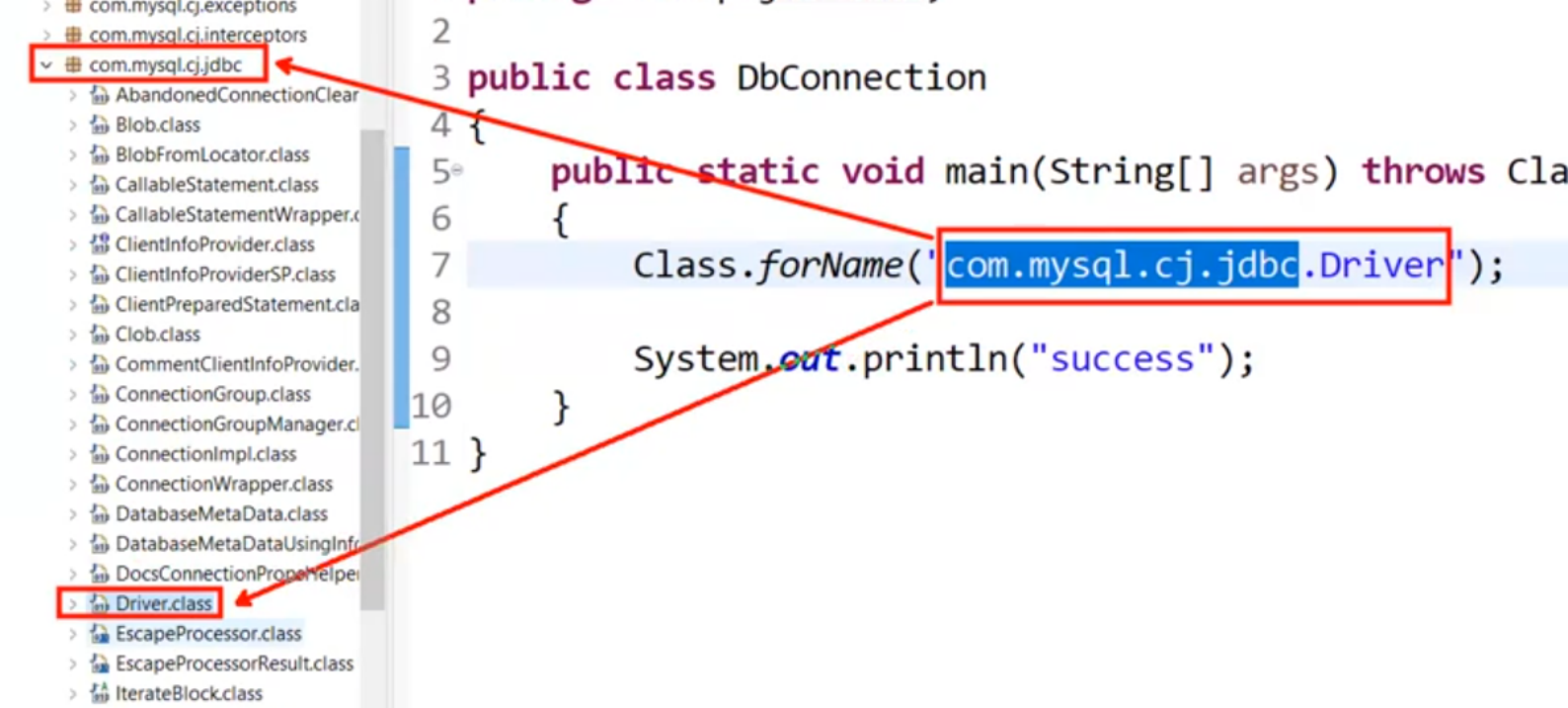

Java Application
Connection
Driver
Database
(object)
- Statement
- PreparedStatement
- CallableStatement
(object)
(variable)
- ResultSet

=> JDBC :-
-> JDBC stands for Java Database Connectivity
-> JDBC is a technology which is used to interact the java application with database

-> JDBC is also an API (pre-defined interfaces, classes and packages)
-> NOTE : JDBC is an abstraction which is provided by Sun-Microsystems and implemented by database vendors (and provide them in the form of jar files)

-> Working of JDBC :-
= Diagram

-> JDBC Components :-
= Driver (Translator) :
- It is used to convert the java calls to database specific calls and database specific calls to java calls
= Connection (Road) :
- It is used to create connection between java application and database
= Statement / PreparedStatement / CallableStatement (Truck) :
- It is used to send the SQL Queries (with data) from java application to database and get the result
= ResultSet (Box) :
- It is used to store the output from database


com.mysql.cj.exceptions
com.mysql.cj.interceptors
com.mysql.cj.jdbc
AbandonedConnectionClear
Blob.class
BlobFromLocator.class
CallableStatement.class
CallableStatementWrapper.
ClientInfoProvider.class
ClientInfoProviderSP.class
ClientPreparedStatement.cla
Clob.class
CommentClientInfoProvider.
ConnectionGroup.class
ConnectionGroupManager.cl
ConnectionImpl.class
ConnectionWrapper.class
DatabaseMetaData.class
DatabaseMetaDataUsingInfo
DocsConnectionPropeHelper
Driver.class
EscapeProcessor.class
EscapeProcessorResult.class
IterateBlock.class
2
3 public class DbConnection
4 {
5 public static void main(String[] args) throws Cla
6 {
7 Class.forName("com.mysql.cj.jdbc.Driver");
8
9 System.out.println("success");
10 }
11 }

-> Steps to create database connection :-

1. Load and register driver
   = Class.forName("Driver ClassName");

2. Establish the connection between java application and database
   = DriverManager.getConnection("url", "username", "password");

==========

=> Steps to execute SQL query using JDBC :-

1. Load and register driver
   -> Class.forName("Driver ClassName");

2. Establish the connection between java application and database
   -> Connection con = DriverManager.getConnection("url", "username", "password");

3. Create Statement / PreparedStatement / CallableStatement object
   -> Statement st = con.createStatement();
   -> PreparedStatement ps = con.prepareStatement();
   -> CallableStatement cs = con.prepareCall();

4. Send and execute SQL query
   -> int count = ps.executeUpdate();
   -> ResultSet rs = ps.executeQuery();

5. Process the result

6. Close all the resources
   -> ps.close();
   -> con.close();

==========

eclipse-workspace-morning - CrudOperation1/src/in/sp/main/InsertDemo1.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

InsertDemo1.java

```
10    public static void main(String[] args) throws ClassNotFoundException, SQLException
11    {
12        Class.forName("com.mysql.cj.jdbc.Driver");
13        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/shopping_app", "root", "root");
14
15        PreparedStatement ps = con.prepareStatement("INSERT INTO users VALUES(?,?,?,?,?)");
16        ps.setString(1, "ravi");
17        ps.setString(2, "ravi@gmail.com");
18        ps.setString(3, "ravi123");
19        ps.setString(4, "male");
20        ps.setString(5, "delhi");
21
22        int count = ps.executeUpdate();
23        if(count > 0)
24        {
25            System.out.println("user inserted successfully");
26        }
27        else
28        {
29            System.out.println("user not registered due to some error");
30        }
31
32        ps.close();
33        con.close();
```

Writable | Smart Insert | 33 : 21 : 907

ENG

eclipse-workspace-morning - CrudOperation1/src/in/sp/main/UpdateDemo.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

InsertDemo.java UpdateDemo.java

```
5  import java.sql.PreparedStatement;
6
7  public class UpdateDemo
8
9      public static void main(String[] args) throws Exception
10     {
11         Class.forName("com.mysql.cj.jdbc.Driver");
12         Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/shopping_app", "root", "root");
13
14         PreparedStatement ps = con.prepareStatement("UPDATE users SET city=? WHERE email=?");
15         ps.setString(1, "mumbai");
16         ps.setString(2, "ravi@gmail.com");
17
18         int count = ps.executeUpdate();
19         if(count > 0)
20         {
21             System.out.println("user updated successfully");
22         }
23         else
24         {
25             System.out.println("user not updated due to some error");
26         }
27     }
28
```

Writable Smart Insert 25 : 70 : 721

ENG

eclipse-workspace-morning - CrudOperation1/src/in/sp/main/DeleteDemo.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

InsertDemo.java UpdateDemo.java DeleteDemo.java

```
4  import java.sql.DriverManager;
5  import java.sql.PreparedStatement;
6
7  public class DeleteDemo
8  {
9      public static void main(String[] args) throws Exception
10     {
11         Class.forName("com.mysql.cj.jdbc.Driver");
12         Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/shopping_app", "root", "root");
13
14         PreparedStatement ps = con.prepareStatement("DELETE FROM users WHERE name=?");
15         ps.setString(1, "kamal");
16
17         int count = ps.executeUpdate();
18         if(count > 0)
19         {
20             System.out.println("user updated successfully");
21         }
22         else
23         {
24             System.out.println("user not updated due to some error");
25         }
26
27         ps.close();
```

Writable Smart Insert 15 : 34 : 480

ENG

eclipse-workspace-morning - CrudOperation2/src/in/sp/main/SelectDemo.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

SelectDemo.java

```
7
8  public class SelectDemo
9  {
10     public static void main(String[] args) throws Exception
11     {
12         Class.forName("com.mysql.cj.jdbc.Driver");
13         Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/shopping_app", "root", "root");
14
15         PreparedStatement ps = con.prepareStatement("select * from users");
16         ResultSet rs = ps.executeQuery();
17
18         while(rs.next())
19         {
20             String myname = rs.getString("name");
21             String myemail = rs.getString("email");
22             String mypass = rs.getString("password");
23             String mygender = rs.getString("gender");
24             String mycity = rs.getString("city");
25
26             System.out.println(myname+" - "+myemail+" - "+mypass+" - "+mygender+" - "+mycity+"\n");
27         }
28     }
29 }
30
```

Writable | Smart Insert | 26 : 97 : 848

ENG

```
PreparedStatement ps = con.prepareStatement("select * from users");
ResultSet rs = ps.executeQuery();

while(rs.next())
{
      String myname = rs.getString("name");
      String myemail = rs.getString("email");
      String mypass = rs.getString("password");
      String mygender = rs.getString("gender");
      String mycity = rs.getString("city");

      System.out.println(myname+" - "+myemail+" - "+mypass+" - "+mygender+" - "+mycity+"\n");
}
```

"rs" object

name	email	password	gender	city
deepak	deepak@gmail.com	deepak123	male	chandigarh
priya	priya@gmail.com	priya123	female	pune
ravi	ravi@gmail.com	ravi123	male	mumbai

resultset cursor

MySQL

```
mysql> select * from users;
+--------+-----------------+----------+--------+-----------+
| name   | email           | password | gender | city      |
+--------+-----------------+----------+--------+-----------+
| deepak | deepak@gmail.com | deepak123 | male   | chandigarh |
| priya  | priya@gmail.com  | priya123  | female | pune       |
| ravi   | ravi@gmail.com   | ravi123   | male   | mumbai     |
+--------+-----------------+----------+--------+-----------+
```

```
=> ResultSet :-
    -> It is an interface which contains the data and manages the cursor to access the data
    -> Method of ResultSet :-
        1. Navigation Methods :-
            = next()
            = previous()
            = beforeFirst()
            = afterLast()
            = first()
            = last()
            = abosolute(-)
            = relative(-)
              etc

        2. Getter Methods :-
            = getString(-)
            = getInt(-)
            = getXXX(-)
              etc
```

```
//it will move the resultset cursor one by one to each row
while(rs.next())
{
    String myname = rs.getString("name");
    String myemail = rs.getString("email");
    String mypass = rs.getString("password");
    String mygender = rs.getString("gender");
    String mycity = rs.getString("city");

    System.out.println(myname+" - "+myemail+" - "+mypass+" - "+mygender+" - "+mycity
}
```

```
//it will move the resultset cursor to the next row
rs.next();

String myname = rs.getString("name");
String myemail = rs.getString("email");
String mypass = rs.getString("password");
String mygender = rs.getString("gender");
String mycity = rs.getString("city");

System.out.println(myname+" - "+myemail+" - "+mypass+" - "+mygender+" - "+mycity+"\r

rs.close();
```

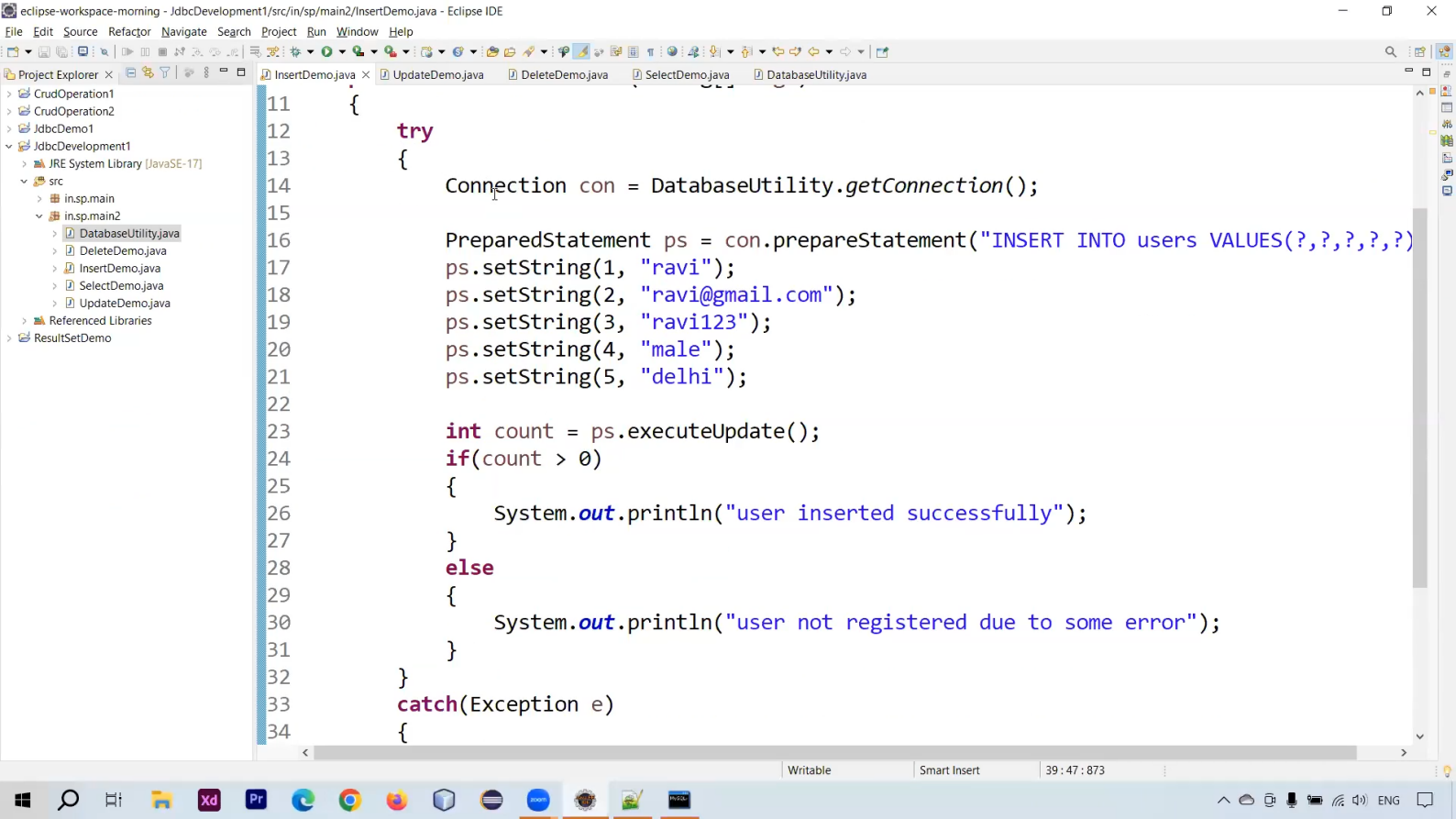
```
//it will move the resultset cursor to first row and then second row
rs.next();
rs.next();

String myname = rs.getString("name");
String myemail = rs.getString("email");
String mypass = rs.getString("password");
String mygender = rs.getString("gender");
String mycity = rs.getString("city");

System.out.println(myname+" - "+myemail+" - "+mypass+" - "+mygender+" - "+mycity+"\r
```

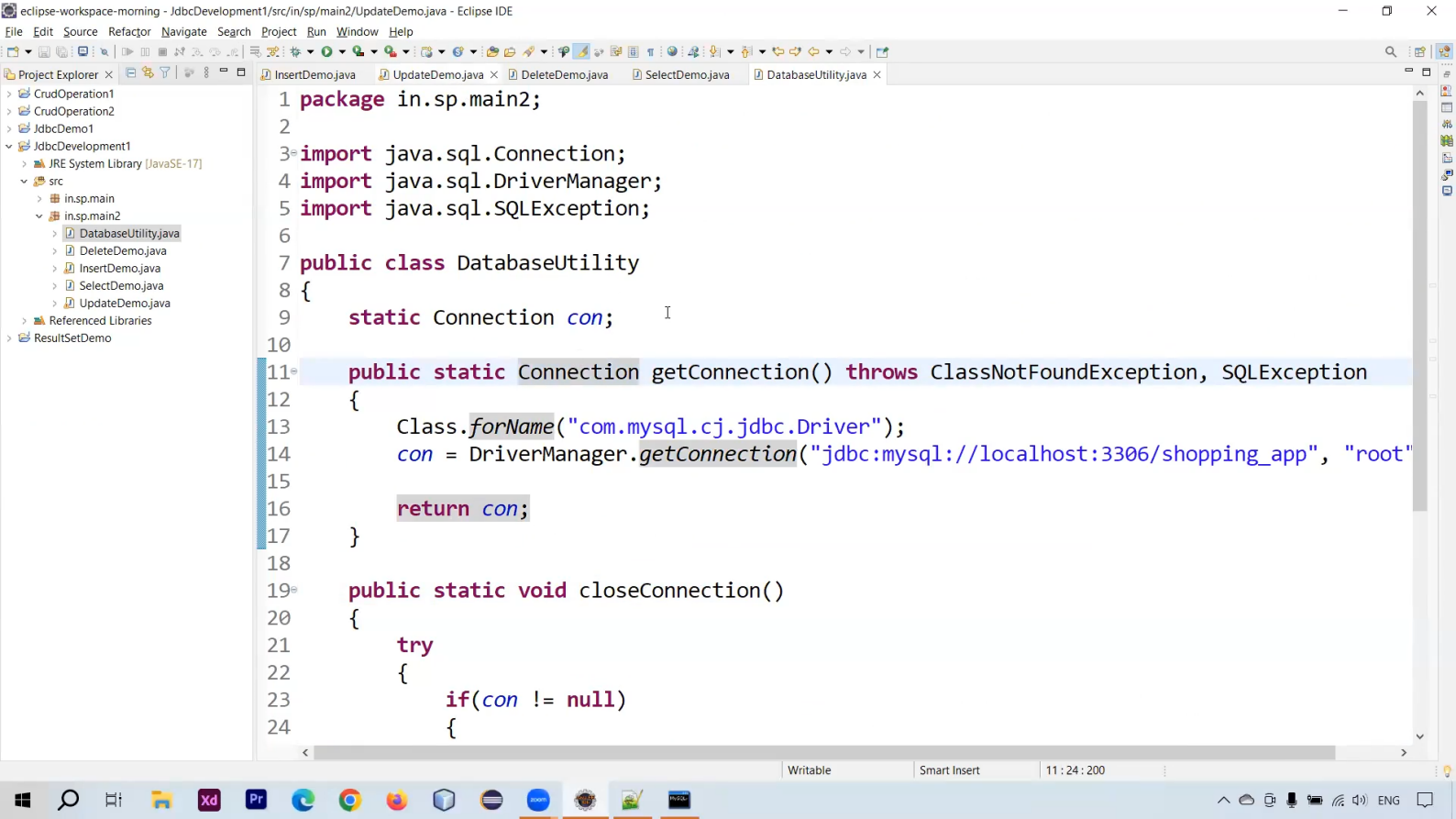
```
//it will move the resultset cursor to the first row
rs.first();

String myname = rs.getString("name");
String myemail = rs.getString("email");
String mypass = rs.getString("password");
String mygender = rs.getString("gender");
String mycity = rs.getString("city");

System.out.println(myname+" - "+myemail+" - "+mypass+" - "+mygender+" - "+mycity+"\r

rs.close();
```

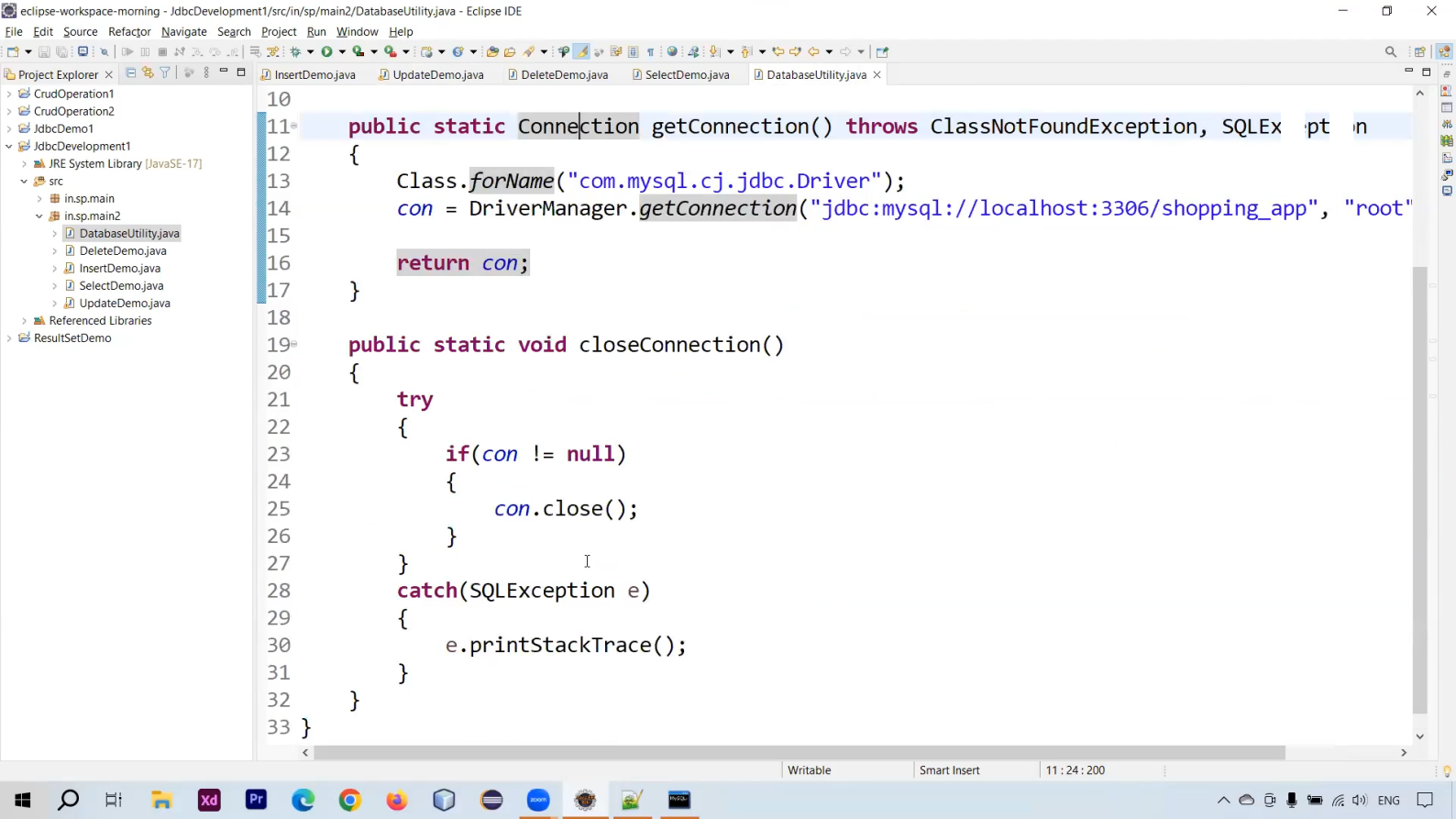
```
//it will move the resultset cursor to the last row
rs.last();

String myname = rs.getString("name");
String myemail = rs.getString("email");
String mypass = rs.getString("password");
String mygender = rs.getString("gender");
String mycity = rs.getString("city");

System.out.println(myname+" - "+myemail+" - "+mypass+" - "+mygender+" - "+mycity+"\r

rs.close();
```

```
//it will point the resultset cursor after the last row
rs.afterLast();

//it will fetch the data in backword direction
while(rs.previous())
{
    String myname = rs.getString("name");
    String myemail = rs.getString("email");
    String mypass = rs.getString("password");
    String mygender = rs.getString("gender");
    String mycity = rs.getString("city");

    System.out.println(myname+" - "+myemail+" - "+mypass+" - "+mygender+" - "+mycity
```

```
-> Types of ResultSet cursor :-
    1. Forward-Only ResultSet
        = This type of resultset allows the traversal of data only in forward direction
        = It's the most memory-efficient type
        = Syntax :-
            ResultSet.TYPE_FORWARD_ONLY
    2. Scrollable ResultSet
        = This type of resultset allows the traversal of data in forward and backword direction
        = These are of two types :-
            - ResultSet.TYPE_SCROLL_INSENSITIVE
            - ResultSet.TYPE_SCROLL_SENSITIVE

    = How to use ?
        - PreparedStatement ps = con.prepareStatement("select query", ResultSet.TYPE_FORWARD_ONLY);
```


eclipse-workspace-morning - JdbcDevelopment1/src/in/sp/main2/InsertDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
CrudOperation1
CrudOperation2
JdbcDemo1
JdbcDevelopment1
JRE System Library [JavaSE-17]
src
in.sp.main
in.sp.main2
DatabaseUtility.java
DeleteDemo.java
InsertDemo.java
SelectDemo.java
UpdateDemo.java
Referenced Libraries
ResultSetDemo
InsertDemo.java
UpdateDemo.java
DeleteDemo.java
SelectDemo.java
DatabaseUtility.java
{
try
{
Connection con = DatabaseUtility.getConnection();
PreparedStatement ps = con.prepareStatement("INSERT INTO users VALUES(?,?,?,?,?)
ps.setString(1, "ravi");
ps.setString(2, "ravi@gmail.com");
ps.setString(3, "ravi123");
ps.setString(4, "male");
ps.setString(5, "delhi");
int count = ps.executeUpdate();
if(count > 0)
{
System.out.println("user inserted successfully");
}
else
{
System.out.println("user not registered due to some error");
}
}
catch(Exception e)
{
Writable
Smart Insert
39 : 47 : 873
ENG


eclipse-workspace-morning - JdbcDevelopment1/src/in/sp/main2/UpdateDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
CrudOperation1
CrudOperation2
JdbcDemo1
JdbcDevelopment1
JRE System Library [JavaSE-17]
src
in.sp.main
in.sp.main2
DatabaseUtility.java
DeleteDemo.java
InsertDemo.java
SelectDemo.java
UpdateDemo.java
Referenced Libraries
ResultSetDemo
InsertDemo.java
UpdateDemo.java
DeleteDemo.java
SelectDemo.java
DatabaseUtility.java
1 package in.sp.main2;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class DatabaseUtility
8 {
9 static Connection con;
10
11 public static Connection getConnection() throws ClassNotFoundException, SQLException
12 {
13 Class.forName("com.mysql.cj.jdbc.Driver");
14 con = DriverManager.getConnection("jdbc:mysql://localhost:3306/shopping_app", "root"
15
16 return con;
17 }
18
19 public static void closeConnection()
20 {
21 try
22 {
23 if(con != null)
24 {
Writable
Smart Insert
11 : 24 : 200
ENG

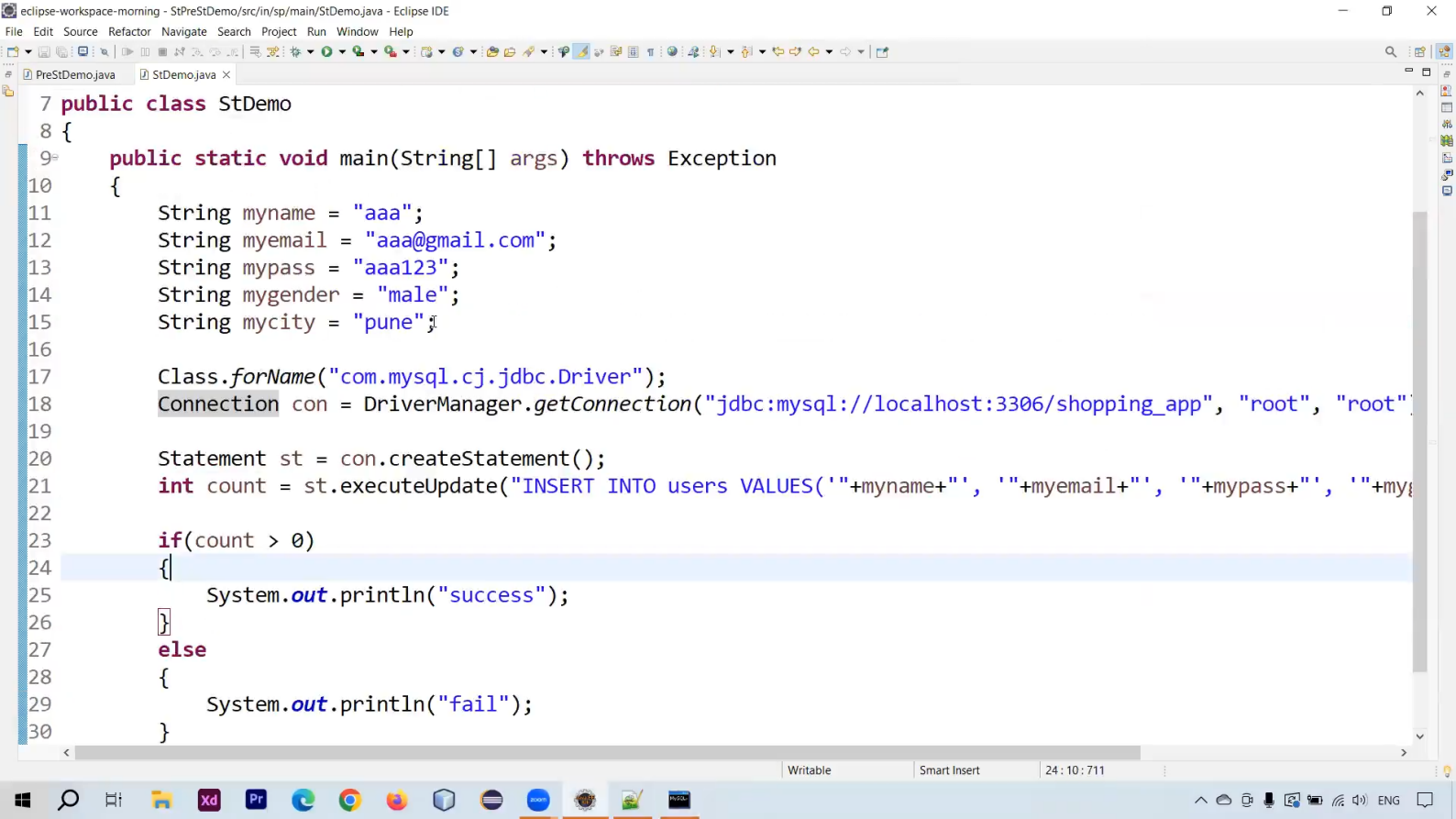

eclipse-workspace-morning - JdbcDevelopment1/src/in/sp/main2/DatabaseUtility.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
CrudOperation1
CrudOperation2
JdbcDemo1
JdbcDevelopment1
JRE System Library [JavaSE-17]
src
in.sp.main
in.sp.main2
DatabaseUtility.java
DeleteDemo.java
InsertDemo.java
SelectDemo.java
UpdateDemo.java
Referenced Libraries
ResultSetDemo
InsertDemo.java
UpdateDemo.java
DeleteDemo.java
SelectDemo.java
DatabaseUtility.java
10
11 public static Connection getConnection() throws ClassNotFoundException, SQLEx pt n
12 {
13 Class.forName("com.mysql.cj.jdbc.Driver");
14 con = DriverManager.getConnection("jdbc:mysql://localhost:3306/shopping_app", "root"
15
16 return con;
17 }
18
19 public static void closeConnection()
20 {
21 try
22 {
23 if(con != null)
24 {
25 con.close();
26 }
27 }
28 catch(SQLException e)
29 {
30 e.printStackTrace();
31 }
32 }
33 }
Writable
Smart Insert
11 : 24 : 200
ENG

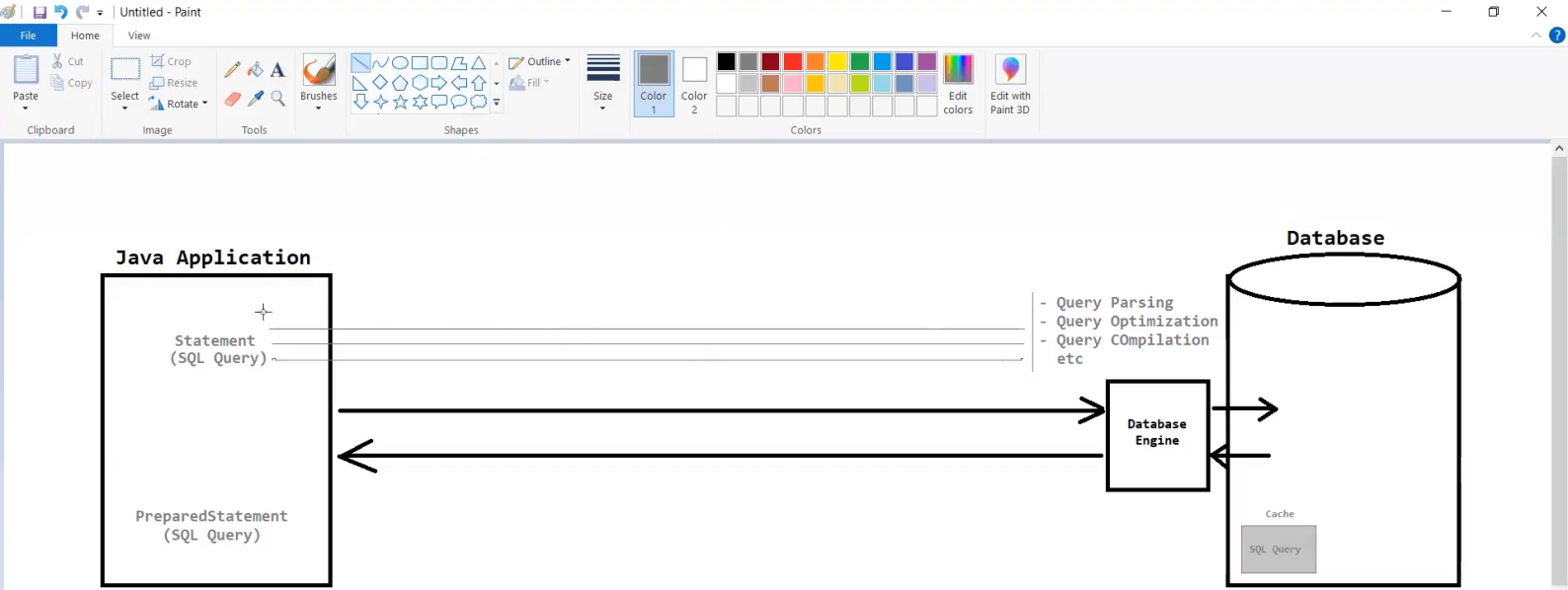
=> How to use JDBC in real world projects development :-

= NOTE : JDBC is less used for enterprise applications because there are a lot of drawbacks of JDBC and to remove these drawbacks we use frameworks i.e. hibernate, JPA, Data JPA etc

1. Use try-catch-finally block
2. Use try-with-resources
3. Use static method approach

= NOTE : Above all approches are not efficient. For database connections we use "connection pooling" concept


eclipse-workspace-morning - StPreStDemo/src/in/sp/main/StDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
PreStDemo.java
StDemo.java
7 public class StDemo
8 {
9 public static void main(String[] args) throws Exception
10 {
11 String myname = "aaa";
12 String myemail = "aaa@gmail.com";
13 String mypass = "aaa123";
14 String mygender = "male";
15 String mycity = "pune";
16
17 Class.forName("com.mysql.cj.jdbc.Driver");
18 Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/shopping_app", "root", "root");
19
20 Statement st = con.createStatement();
21 int count = st.executeUpdate("INSERT INTO users VALUES('"+myname+"', '"+myemail+"', '"+mypass+"', '"+myg
22
23 if(count > 0)
24 {
25 System.out.println("success");
26 }
27 else
28 {
29 System.out.println("fail");
30 }
Writable
Smart Insert
24 : 10 : 711
ENG


Untitled - Paint
File
Home
View
Cut
Copy
Paste
Crop
Resize
Rotate
Select
Brushes
Outline
Fill
Size
Color 1
Color 2
Edit colors
Edit with Paint 3D
Clipboard
Image
Tools
Shapes
Colors
Java Application
Statement
(SQL Query)
PreparedStatement
(SQL Query)
- Query Parsing
- Query Optimization
- Query COmpilation
etc
Database
Database Engine
Cache
SQL Query
317, 206px
918 × 1px
2756 × 1160px
100%
ENG

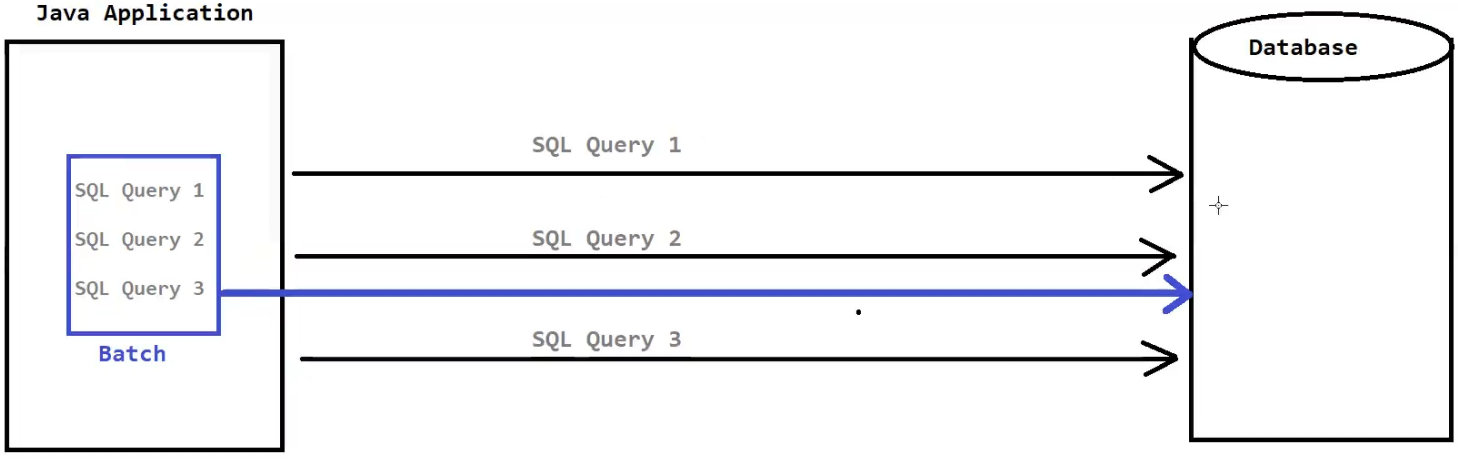
=> Statement :
    -> Statement objects are used to execute simple SQL queries without any parameters
    -> Statement are best suited for static queries that do not involve any user inputs
    -> Statement performance is low as compared to PreparedStatement
    -> Statement are less secured

=> PreparedStatement :
    -> PreparedStatement objects are used to execute parametrized SQL queries
    -> PreparedStatement are best suited for dynamic queries which involves user inputs
    -> PreparedStatement performance is fast as compared to Statement
    -> PreparedStatement are more secured

```
11      public static void main(String[] args) throws Exception
12      {
13          Scanner sc = new Scanner(System.in);
14
15          System.out.println("Enter Email Id :");
16          String myemail = sc.nextLine();
17
18          System.out.println("Enter Password :");
19          String mypass = sc.nextLine();
20
21          Class.forName("com.mysql.cj.jdbc.Driver");
22          Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/shopping_app", "root", "root"
23
24          Statement st = con.createStatement();
25          ResultSet rs = st.executeQuery("SELECT * FROM users WHERE email='"+myemail+"' AND password='"+mypass+"'"
26
27          if(rs.next())
28          {
29              System.out.println("Welcome : "+rs.getString("name"));
30          }
31          else
32          {
33              System.out.println("Email id and password didnt matched");
34          }
```

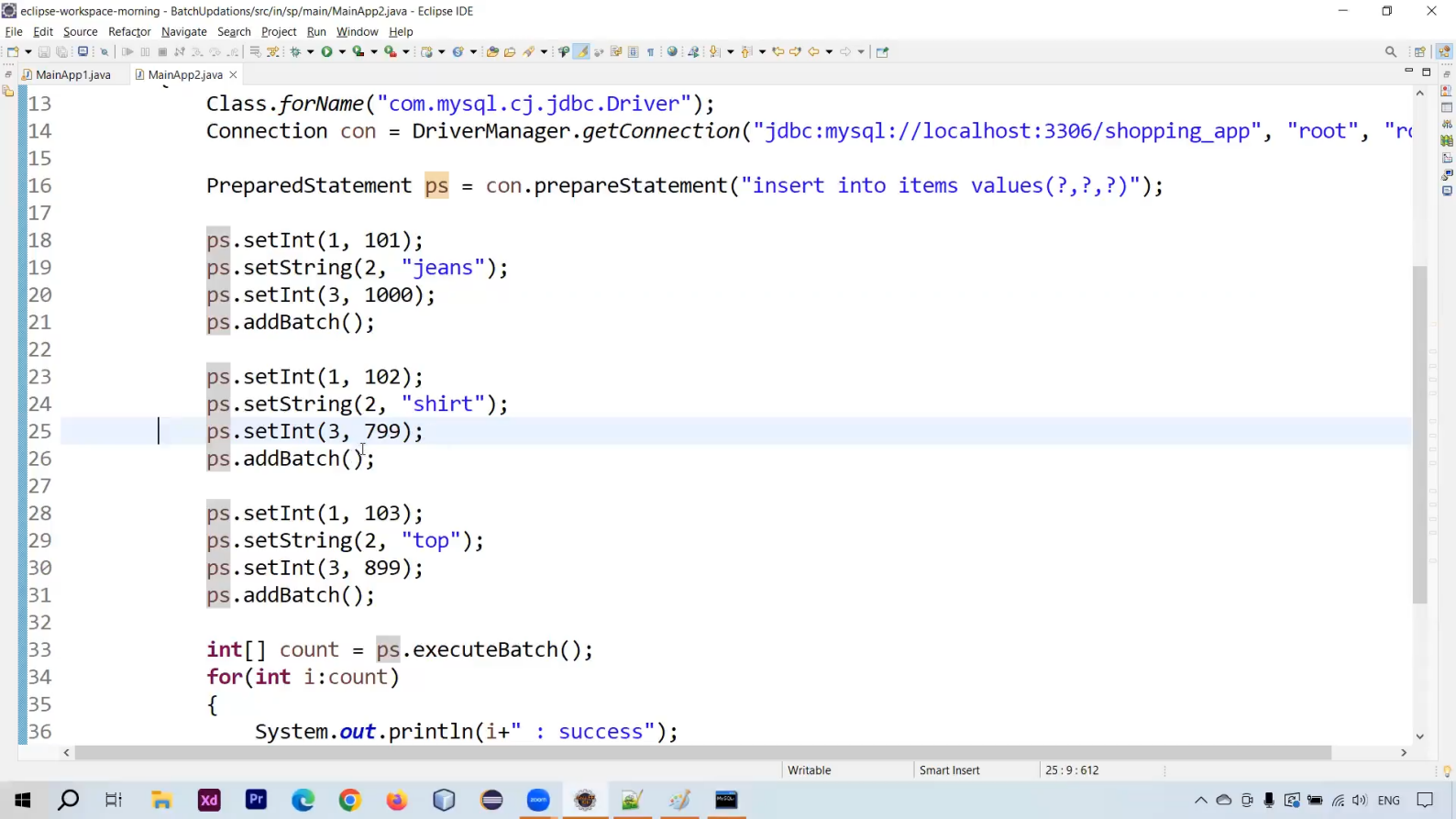
=> SQL Injection Attack :-
    -> SQL Injection is a type of cybersecurity attack that targets the database and is used to manipulate or gain unauthorized access to the data stored within the database
    -> The root cause of SQL injection is mixing of "SQL query" and "data"
    -> It occurs only in Statement interface

    -> How PreparedStatement protects from SQL Injection Attack ?
        = Becuase "SQL Query" and "data" are sent separately to the database server


Java Application
SQL Query 1
SQL Query 2
SQL Query 3
Batch
SQL Query 1
SQL Query 2
SQL Query 3
Database

```
=> Batch Updations :-
    -> 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 :-


    -> How to achieve batch updations :-
        = For batch updations we have 3 methods :-
            1. addBatch(String query)
            2. executeBatch() -> int[]
            3. clearBatch()
        = NOTE : These methods are present in Statement and PreparedStatement interface
```

```
{
    public static void main(String[] args)
    {
        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/shopping_app", "root", "r

            Statement st = con.createStatement();

            st.addBatch("insert into items values(101, 'jeans', 1000)");
            st.addBatch("insert into items values(102, 'shirt', 699)");
            st.addBatch("insert into items values(103, 'top', 899)");

            int[] count = st.executeBatch();
            for(int i : count)
            {
                System.out.println(1+" : success");
            }
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
```


eclipse-workspace-morning - BatchUpdations/src/in/sp/main/MainApp2.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
MainApp1.java
MainApp2.java
Class.forName("com.mysql.cj.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/shopping_app", "root", "r
PreparedStatement ps = con.prepareStatement("insert into items values(?,?,?)");
ps.setInt(1, 101);
ps.setString(2, "jeans");
ps.setInt(3, 1000);
ps.addBatch();
ps.setInt(1, 102);
ps.setString(2, "shirt");
ps.setInt(3, 799);
ps.addBatch();
ps.setInt(1, 103);
ps.setString(2, "top");
ps.setInt(3, 899);
ps.addBatch();
int[] count = ps.executeBatch();
for(int i:count)
{
System.out.println(i+" : success");
Writable
Smart Insert
25 : 9 : 612
ENG

```
=> Batch Updations :-
    -> It is a batch of updates grouped together and sent to the database in one batch rather t  en
    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 not be executed
    -> How to achieve batch updations :-
        = For batch updations we have 3 methods :-
            1. addBatch(String query)
            2. executeBatch() -> int[]
            3. clearBatch()
        = NOTE : These methods are present in Statement and PreparedStatement interface

================================================================================================================
```

=> What is transaction :-
-> A group of several SQL operations as a single unit where SQL operations are executed on ne principal of either all or none
-> Properties of transaction :-
= It follows the ACID property :-
- A (Atomicity) : Either perform all operations or none
- C (Consistency) : The databse state must be stable
- I (Isolation) : All the transactions must be executed independently, one transaction must not give effect to another transaction
- D (Durability) : The changes of successful transaction occurs even if the system/database failure occurs

---

=> Transaction in MySQL Database :
-> SELECT @@AUTOCOMMIT; : to check the autocommit status. If its 1 then it will store the data permanenetly and if value is 0 then it will store the data temporary

-> SET AUTOCOMMIT=0; : It will change the autocommit value to 0 (temporary)

-> COMMIT; : It will store the temporary data permanenetly

-> ROLLBACK; - It will rollback (remove) the temporary data

=> How to implement transaction management in JDBC :-

-> For transaction management, we have 3 methods :-

1. setAutoCommit(boolean)
2. commit()
3. rollback()

-> NOTE : These methods are present in Connection interface

-> Syntax :-

-> Syntax :-

```
Connection con = ------

try
{
    con.setAutoCommit(false);

    //SQL operations 1
    //SQL operations 2
    //SQL operations 3
    //SQL operations 4

    if(condition -> true)
    {
        con.commit();
    }
    else
    {
        con.rollback();
    }
}
catch(Exception e)
{
    con.rollback();
    e.printStackTrace();
}
```

Java Application

From Account No :

12345

To Account No :

67890

Amount :

2000

Transfer

```
= UPDATE balace SET balance=old_balance-2000 WHERE acc_no='12345';

= UPDATE balace SET balance=old_balance+2000 WHERE acc_no='67890';

= INSERT INTO transaction_details VALUES(?,?,?,?,?);
   -12345  -2000  -debit  -internetbanking  -11/11/2023  -10:10:20

= INSERT INTO transaction_details VALUES(?,?,?,?,?);
   -67890  -2000  -credit  -internetbanking  -11/11/2023  -10:10:20
```

Database

balance

id	acc_no	balance
1	12345	5000 3000
2	67890	8000

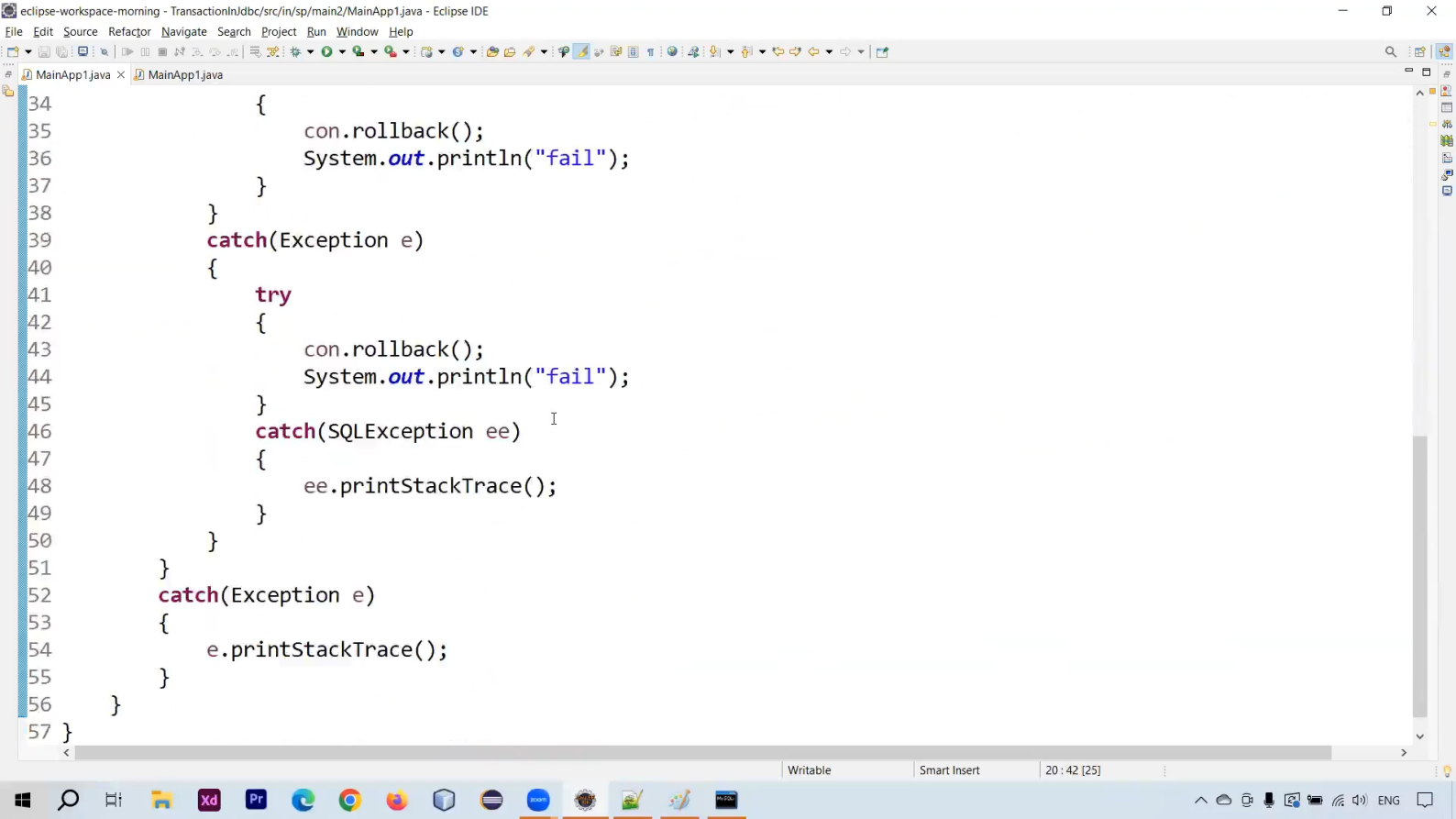
users

id	name	email	password	gender	acc_no
1	aaa	aaa@gmail.com	aaa123	male	12345
2	bbb	bbb@gmail.com	bbb123	female	67890

transaction_details

id	acc_no	amount	credit/debit	mode	date1	time1
1	12345	5000	credit	cash	----	----
2	67890	8000	credit	cash	----	----

```
13          try
14          {
15              Class.forName("com.mysql.cj.jdbc.Driver");
16              Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/shopping_app", "root", "r
17
18              try
19              {
20                  con.setAutoCommit(false);
21
22                  Statement st1 = con.createStatement();
23
24                  int count1 = st1.executeUpdate("insert into items values(101, 'jeans', 999)");
25                  int count2 = st1.executeUpdate("insert into items values(102, 'shirt', 'aaa')");
26                  int count3 = st1.executeUpdate("insert into items values(103, 'top', 799)");
27
28                  if(count1>0 && count2>0 && count3>0)
29                  {
30                      con.commit();
31                      System.out.println("success");
32                  }
33                  else
34                  {
35                      con.rollback();
36                      System.out.println("fail");
```

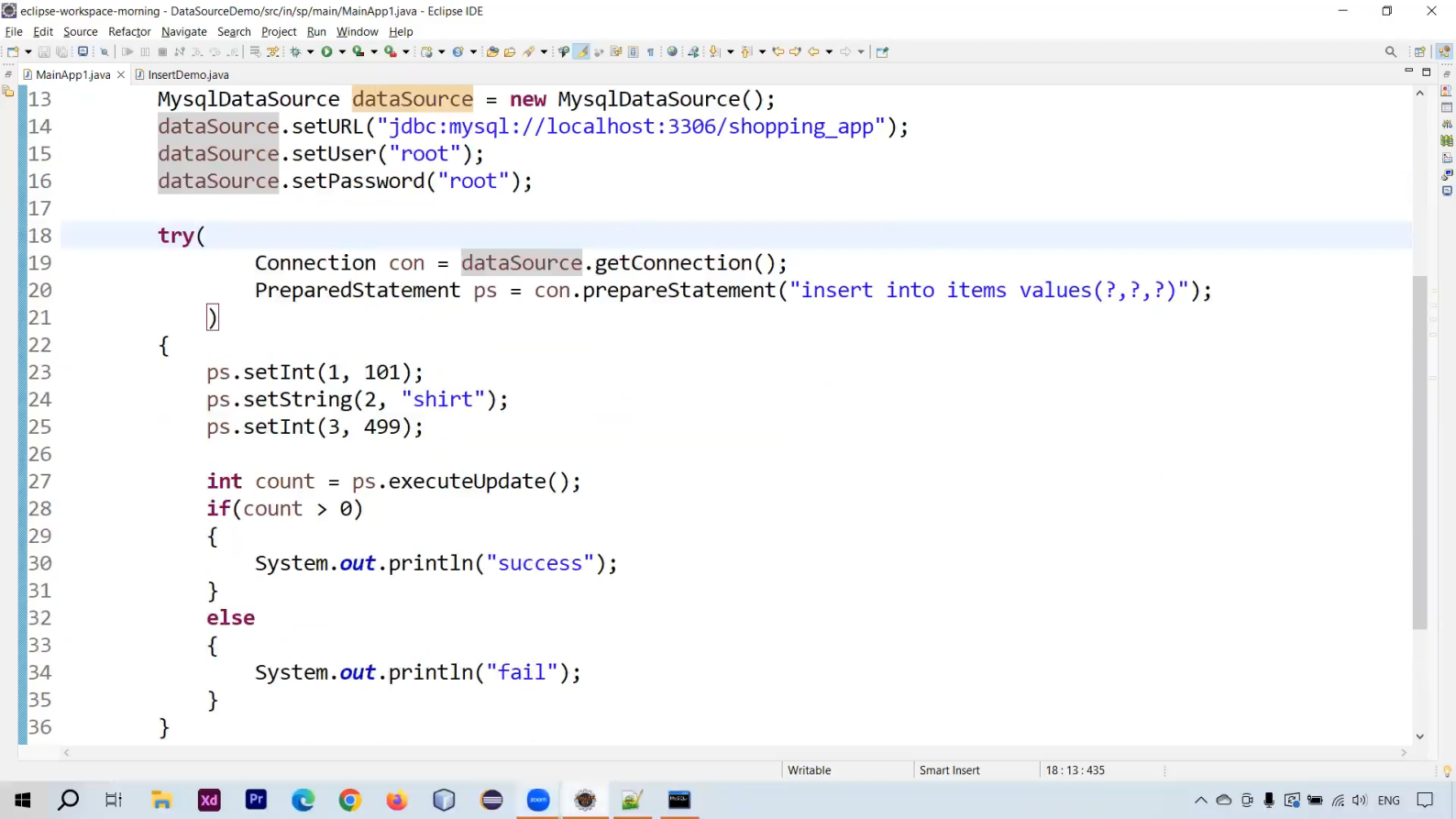

eclipse-workspace-morning - TransactionInJdbc/src/in/sp/main2/MainApp1.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
MainApp1.java
MainApp1.java
{
con.rollback();
System.out.println("fail");
}
}
catch(Exception e)
{
try
{
con.rollback();
System.out.println("fail");
}
catch(SQLException ee)
{
ee.printStackTrace();
}
}
}
catch(Exception e)
{
e.printStackTrace();
}
}
}
Writable
Smart Insert
20 : 42 [25]
ENG

=> We can get connection object by 2 ways :-
    1. DriverManager (class) - java.sql package
    2. DataSource

=> DriverManager :-
    -> It is a class which is present in "java.sql" package
    -> How to get connection object using DriverManager :-
        >> Connection con = DriverManager.getConnection("URL", "USERNAME", "PASSWORD");

    -> Drawbacks of DriverManager :-
        1. DriverManager takes a lot of time to open database connection in a network which will slow down the application performace
        2. Whenever there is increase in the number of clients, the performace of the application will be decreased
        3. The connection object created using DriverManager is not reusable, thus whenever we need connection object, it will again take time to create

---


eclipse-workspace-morning - DataSourceDemo/src/in/sp/main/MainApp1.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
MainApp1.java
InsertDemo.java
13 MysqlDataSource dataSource = new MysqlDataSource();
14 dataSource.setURL("jdbc:mysql://localhost:3306/shopping_app");
15 dataSource.setUser("root");
16 dataSource.setPassword("root");
17
18 try(
19 Connection con = dataSource.getConnection();
20 PreparedStatement ps = con.prepareStatement("insert into items values(?,?,?)");
21)
22 {
23 ps.setInt(1, 101);
24 ps.setString(2, "shirt");
25 ps.setInt(3, 499);
26
27 int count = ps.executeUpdate();
28 if(count > 0)
29 {
30 System.out.println("success");
31 }
32 else
33 {
34 System.out.println("fail");
35 }
36 }
Writable
Smart Insert
18 : 13 : 435
ENG

=> DataSource :-
    -> It is an interface which is present in "javax.sql" package
    -> It defines a standardized way to obtain database connection
    -> Some basic implementation of DataSource are :-
        1. MysqlDataSource : used for MySQL database
        2. OracleDataSource : used for Oracle database
        3. JdbcDataSource : used for H2 database (open-source, in-memory, lightweight database written in java)
            etc
        >> NOTE : Above provided implemented classes does not have in-built connection pooling feature

    -> Why we should use DataSource instead of DriverManager :-
        1. Easy configuration
        2. Easy to switch across different databases
        3. Automatic driver loading
        4. Cleaner and more readable code
        >> NOTE : "Connection Pooling" is the most important advantage of DataSource

=> Connection Pooling :-
-> Connection Pooling is a feature or technique that involves managing and reusing existing database connection objects in connection pool, instead of creating new connections from scracth every time an application interacts with the database
-> This improves the application performace and resource utilization by minimizing the time and resource needed to establish the database connection

-> Real World Example :
= Cooking in Restaurents

-> Different ways to provide connection pooling :-
1. Using third-party connection pooling libraries
= HikariCP
- Known for its high performance and lightweight nature, making it suitable for modern application
- "HikariDataSource" is an implemented class for DataSource
= Apache Commons DBCP
- A widely used connection pooling library with configurable options for connection management
- "BasicDataSource" is an implemented class for DataSource
= C3P0
- It offers features like connection testing and customization options
- "ComboPooledDataSource" is an implemented class for DataSource
= BoneCP
- It is also lightweight connection pooling library designed for speed and efficiency
- "BoneCPDataSource" is an implemented class for DataSource
etc

2. Application server-provided connection pooling
   = Many application servers i.e. Apache Tomcat, WildFly etc come with built-in connection pooling capabilities
3. Spring framework
   = Spring framework also provides its own connection pooling feature support through its DataSource abstraction
   = Spring framework also provides facility to integrate third party connection pooling libraries
   etc

-> NOTE : JDBC does not provide in-built connection pooling feature but we can integrate third-party connection pooling libraries with JDBC

```
HikariConfig config = new HikariConfig();
config.setJdbcUrl("jdbc:mysql://localhost:3306/shopping_app");
config.setUsername("root");
config.setPassword("root");
config.setMaximumPoolSize(10);

HikariDataSource dataSource = new HikariDataSource(config);

try(
        Connection con = dataSource.getConnection();
        PreparedStatement ps = con.prepareStatement("insert into items values(?,?,?)
    )
{
    ps.setInt(1, 101);
    ps.setString(2, "jeans");
    ps.setInt(3, 599);
```

```
try(
        Connection con = dataSource.getConnection();
        PreparedStatement ps = con.prepareStatement("insert into items values(?,?,?)
    )
{
    ps.setInt(1, 101);
    ps.setString(2, "jeans");
    ps.setInt(3, 599);

    int count = ps.executeUpdate();
    if(count > 0)
    {
        System.out.println("success");
    }
    else
    {
        System.out.println("fail");
    }
}
catch(SQLException e)
{
    e.printStackTrace();
}
```

```java
// register HikariCP pool as an MBean
MBeanServer mBeanServer = ManagementFactory.getPlatformMBeanServer();
ObjectName poolObjectName = new ObjectName("com.zaxxer.hikari:type=Pool("+dataSc
mBeanServer.registerMBean(dataSource.getHikariPoolMXBean(), poolObjectName);

// access the pool statistics using JMX attributes
int activeConnection = (int) mBeanServer.getAttribute(poolObjectName, "ActiveCor
int idleConnection = (int) mBeanServer.getAttribute(poolObjectName, "IdleConnect
int totalConnections = activeConnection + idleConnection;

System.out.println("Total Connection : "+totalConnections);
System.out.println("Active Connection : "+activeConnection);
System.out.println("Idle Connection : "+idleConnection);
```