Statements types

Click to Continue

Prepare Statement are precompiled query, used in scenarios where the same query needs to be executed for multiple values. Since they are precompiled they are more efficient.

Java.sql.PreparedStatement

Click to Continue

PreparedStatement interface extends **Statement** interface.

- ***PreparedStatements*** are precompiled Statement.
- Using PreparedStatement a same SQL can be executed many times
- Since it is pre-compiled it has faster performance since it skips the compilation phase.

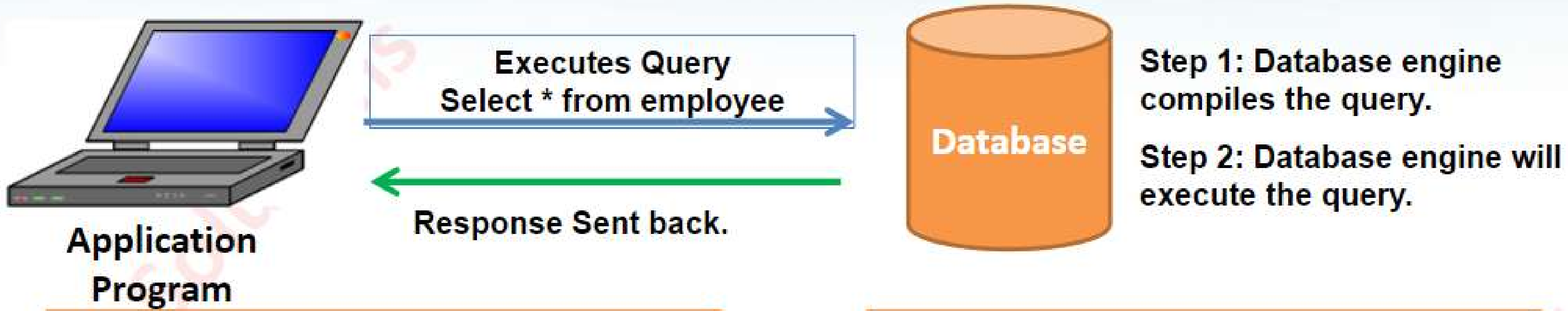
Illustration:

```
        String query = "insert into student values (?,?,?)";

  PreparedStatement  prepareStatement = connection.prepareStatement(query);
```

You will learn to set values in the coming slides.

How prepared statement works?

Click to Continue



Executes Query
Select * from employee
Response Sent back.
Application Program
Database
Step 1: Database engine compiles the query.
Step 2: Database engine will execute the query.

If the above query is executed using Statement, both the steps 1 & 2 will be repeated again and again.

If we execute the query using prepared statement, the query will be compiled once and only step 2 will be executed for repeated execution.

NOTE: ***Precompilation*** is a costly process, consumes more time. So prepared statement compiles the query only once and reuses it for subsequent execution. So it is faster than normal statements.

Analyzing PreparedStatement

Click to Continue

Illustration:

1. String query = “insert into student values (?,?,?)”;
2. **PreparedStatement** prepareStatement = connection.**prepareStatement**(query);

- **“?”** here refers to the parameters to be passed to the query.
- Post creation of the statement the values can be set and executed.
- The same prepared statement can be executed multiple number of times for different values.

Assume we need to insert two student records in the Student table we will have statement **1** and **2** executed only once. But execution will happen two times.

Statement vs PreparedStatement

Click to Continue

Statement	PreparedStatement
Used for executing simple SQL queries .	This is used for dynamic SQL queries with values being changed during run time.
The SQL query is parsed and compiled before it is executed by database engine.	It is precompiled only once, the subsequent executions are only executed.
Statement execution is slow.	PreparedStatement execution is fast.

Execute PreparedStatement

Click to Continue

Illustration:

```
// Load driver and create connection
String query = “insert into student values (?,?)”;
PreparedStatement  ps= connection.prepareStatement(query);
ps.setInt(1, 100);
ps.setString(2,”Raj”);
int n = prepareStatement.executeUpdate();
```

Create Prepared statement object using the query.

Bind variables for IN parameters.

Query executed and number of rows affected is returned

PreparedStatement Example

Click to Continue

Load and register driver.

Establish connection

Set Parameter values

Create a prepared statement using the given SQL

Close the statement, connection in finally block.

Execute query

```
package com.jdbc.demo;
import java.sql.*;
import java.util.Scanner;
public class JDBCDemo2 {
public static Connection connection;
public static PreparedStatement pStatement;
public static void main(String[] args)
    {
    Scanner scanner=new Scanner(System.in);
    System.out.println("Enter Student Id");
    int studentId=scanner.nextInt();
    System.out.println("Enter Student Name");
    String studentName=scanner.next();
    try{
       Driver driver=new oracle.jdbc.driver.OracleDriver();
       DriverManager.registerDriver(driver);
       String url = "jdbc:oracle:thin:@localhost:1521:orcl";
       connection = DriverManager.getConnection(url,"scott","tiger");
       String query="insert into Student(Student_Id,Student_Name) values(?,?)";
       pStatement = connection.prepareStatement(query);
       pStatement.setInt(1,studentId);
       pStatement.setString(2,studentName);
       int studentCount=pStatement.executeUpdate();
       if(studentCount>0){
           System.out.println("Student Details Sucessfully Added");}}
       catch(SQLException sqlException)  {sqlException.printStackTrace();}
       finally{
        try{pStatement.close();
            connection.close(); }
         catch(SQLException sqlException){
            sqlException.printStackTrace();} }  }
   }
```

Try it out – Prepared statemetn

Click to Continue

Let us reuse the same case study we used in the Staement Demo. Refer the video for the setup

Create a table BANKING_ACCOUNT with the following columns account id, name and date_of_opening.

Insert the below records into the table

account_id	Account_name	Date_of_opening
100	John	14/07/2012
22	Binu	11/09/2012
34	Raj	21/06/2012

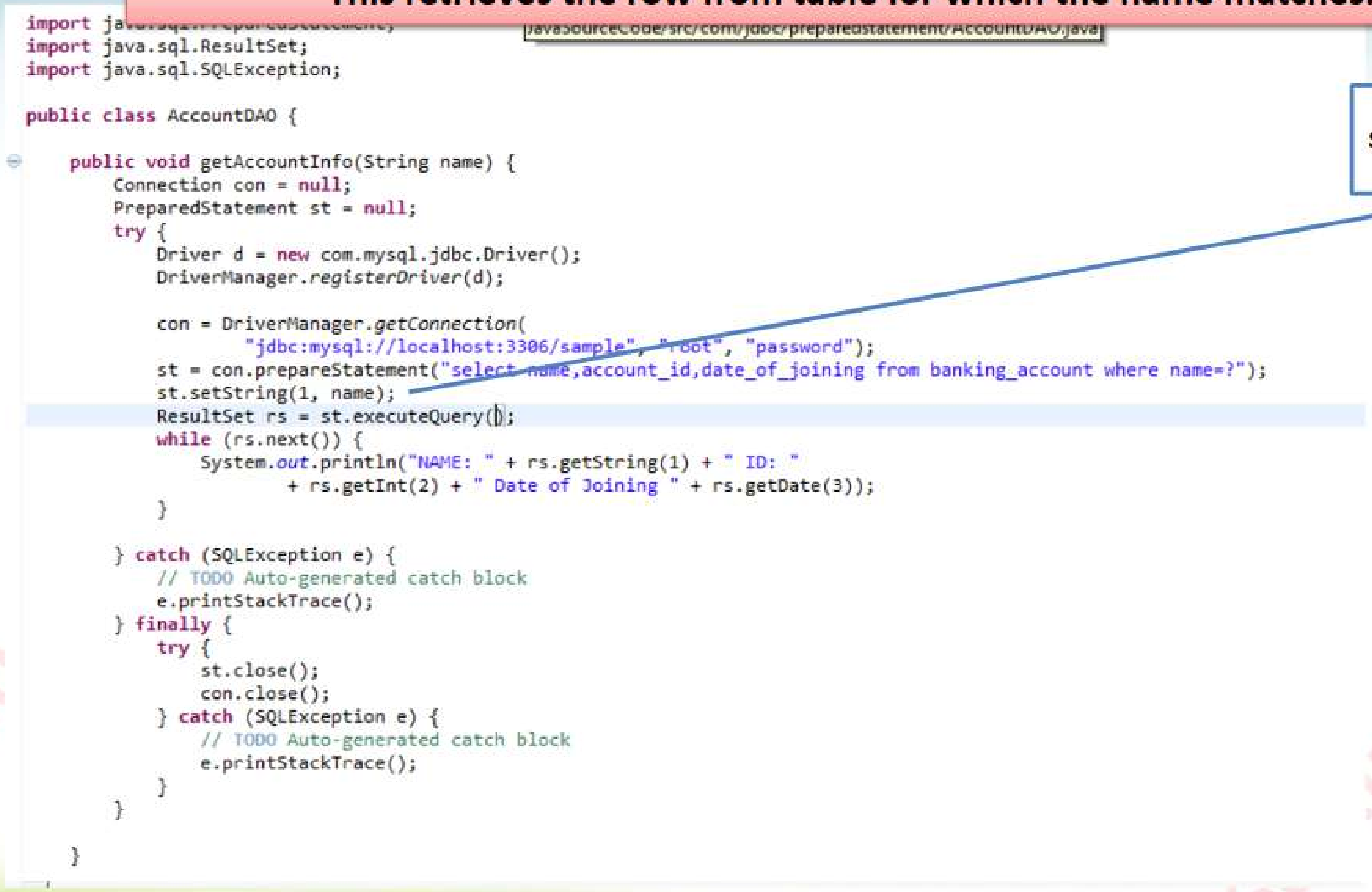
1. Develop a class **AccountDAO with method getAccountDetails(String name)** which retrieves the records from the Banking_Account table whose name is the name passed as method argument and displays each record in the format "Name: John, Account: 100, DOJ: 14/07/2012".
2. Develop a method storeAccountDetails (AccountVO vo) this will accept a AccountVO which has accountId, name and date of opening as member variables. This should insert the account details in the banking_account table.

2. Develop a main class AccountMain which invokes the **getAccountDetails(String name)** method.

Try it out - JDBC Prepared Statement

Click to Continue

This retrieves the row from table for which the name matches.



```
javaSourceCode/src/com/jdbc/preparedstatement/AccountDAO.java
```

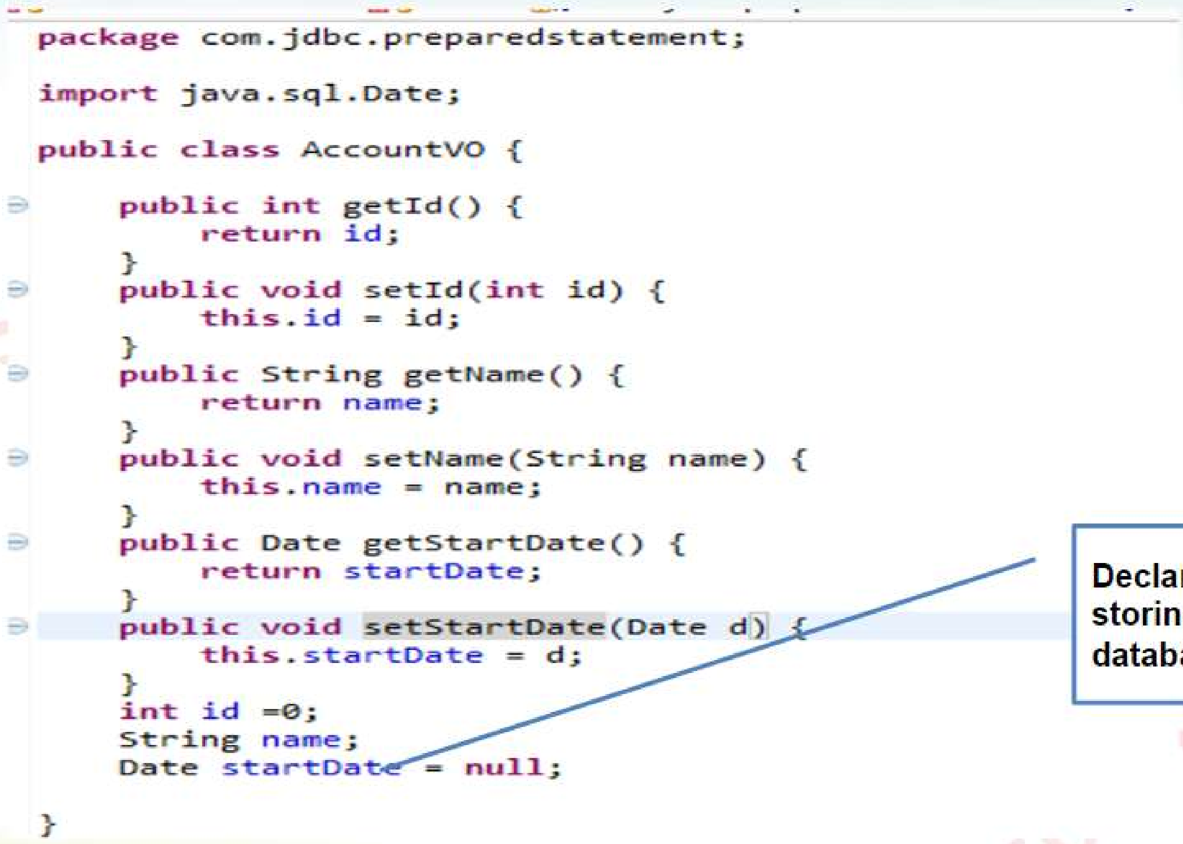
```
import ja
import java.sql.ResultSet;
import java.sql.SQLException;

public class AccountDAO {

    public void getAccountInfo(String name) {
        Connection con = null;
        PreparedStatement st = null;
        try {
            Driver d = new com.mysql.jdbc.Driver();
            DriverManager.registerDriver(d);

            con = DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/sample", "root", "password");
            st = con.prepareStatement("select name,account_id,date_of_joining from banking_account where name=?");
            st.setString(1, name);
            ResultSet rs = st.executeQuery();
            while (rs.next()) {
                System.out.println("NAME: " + rs.getString(1) + " ID: "
                        + rs.getInt(2) + " Date of Joining " + rs.getDate(3));
            }

        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                st.close();
                con.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

    }
```

Setting the value

Try it out - JDBC Prepared Statement

Click to Continue

Create a value object to pass the values of the tables.



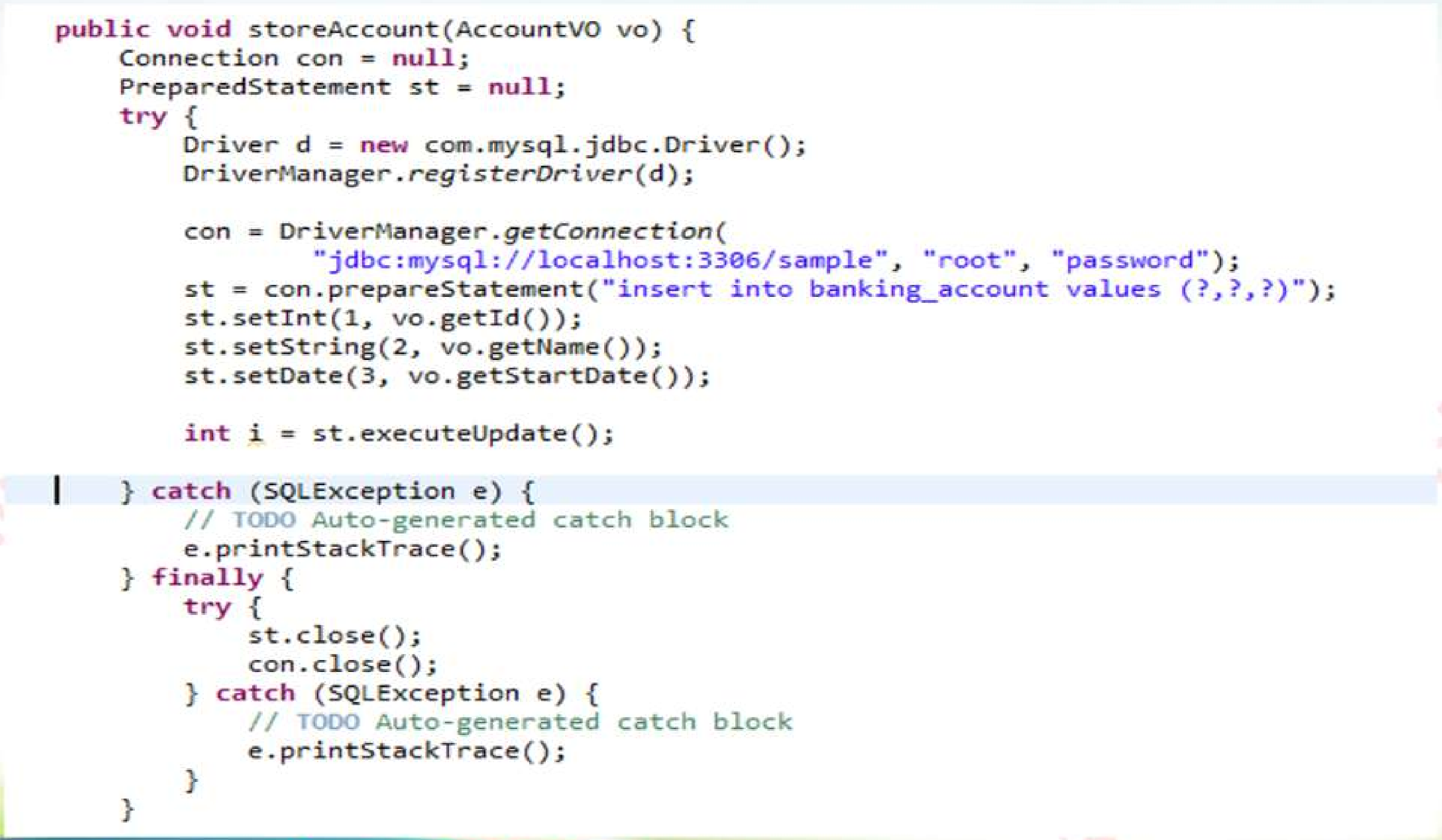
```
package com.jdbc.preparedstatement;

import java.sql.Date;

public class AccountVO {

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Date getStartDate() {
        return startDate;
    }
    public void setStartDate(Date d) {
        this.startDate = d;
    }
    int id =0;
    String name;
    Date startDate = null;

}
```

Declares a java.sql.Date for storing date values in database.

Try it out – StoreAccount

Click to Continue

This inserts a row into the table.
Add this method in the AccountDAO.

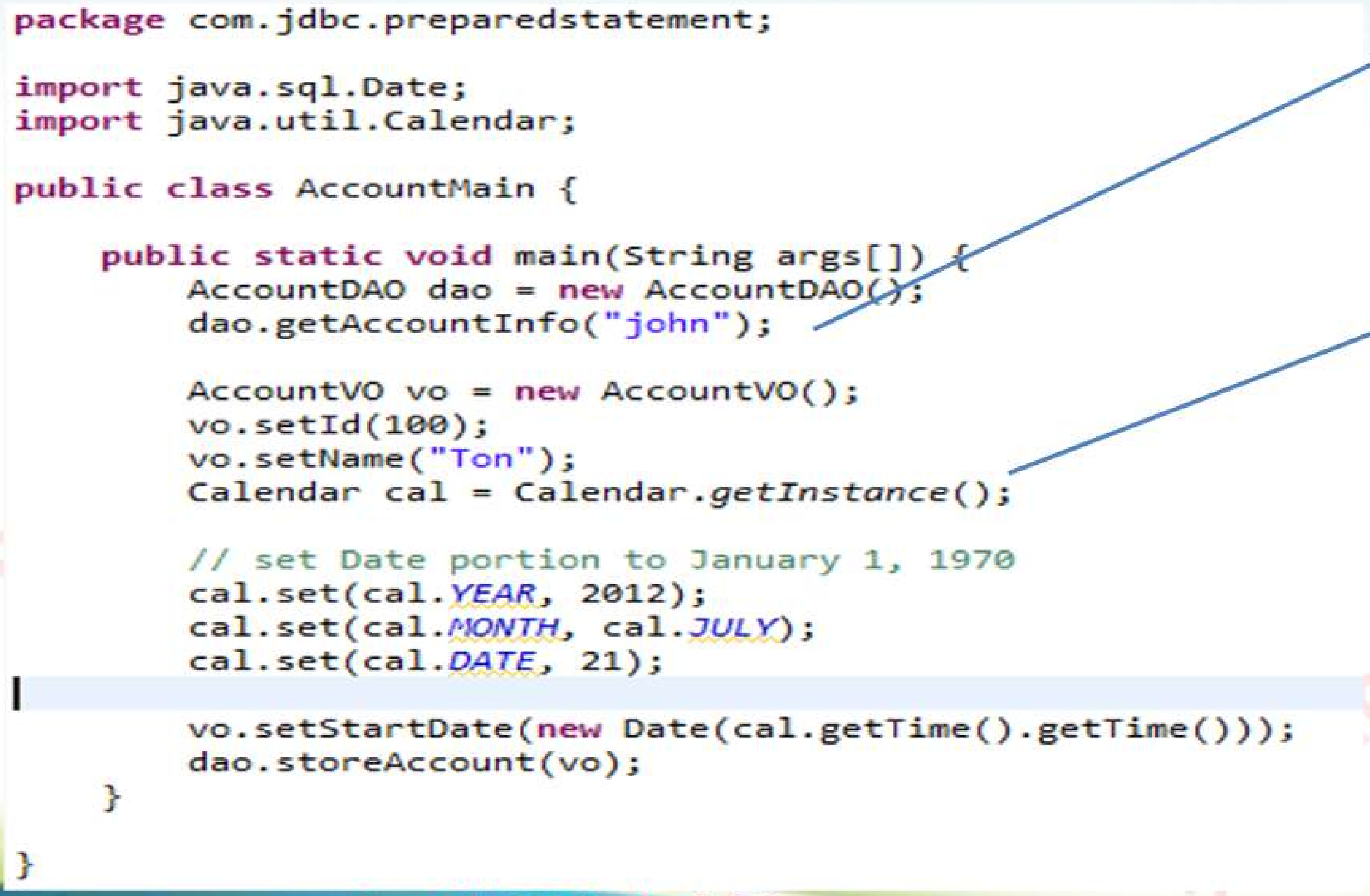


```
public void storeAccount(AccountVO vo) {
    Connection con = null;
    PreparedStatement st = null;
    try {
        Driver d = new com.mysql.jdbc.Driver();
        DriverManager.registerDriver(d);

        con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/sample", "root", "password");
        st = con.prepareStatement("insert into banking_account values (?,?,?)");
        st.setInt(1, vo.getId());
        st.setString(2, vo.getName());
        st.setDate(3, vo.getStartDate());

        int i = st.executeUpdate();

    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } finally {
        try {
            st.close();
            con.close();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

Try it out – Main Method

Click to Continue

Main method which retrieves the data of John and inserts s new record.



```java
package com.jdbc.preparedstatement;

import java.sql.Date;
import java.util.Calendar;

public class AccountMain {

    public static void main(String args[]) {
        AccountDAO dao = new AccountDAO();
        dao.getAccountInfo("john");

        AccountVO vo = new AccountVO();
        vo.setId(100);
        vo.setName("Ton");
        Calendar cal = Calendar.getInstance();

        // set Date portion to January 1, 1970
        cal.set(cal.YEAR, 2012);
        cal.set(cal.MONTH, cal.JULY);
        cal.set(cal.DATE, 21);

        vo.setStartDate(new Date(cal.getTime().getTime()));
        dao.storeAccount(vo);
    }

}
```

This fetches the John record from database and prints it.

This is how we update date inside a date column in a table.