SIMPLE DATABASE QUERYING WITH JAVA

RICHARD C. CUPAL, LPT

(Sepiroth X)

TABLE OF CONTENTS

-

INTRODUCTION

Welcome to "Simple Database Querying with Java," a comprehensive manual designed to empower students with the essential skills needed to excel in database management and query execution using the Java programming language. As an educator with a passion for guiding students through their academic journey, I have crafted this book to address a critical need in modern education. Many students often find themselves grappling with the complexities of database systems, struggling to harness the power of Java to retrieve, manipulate, and manage data effectively. This book is a result of my commitment to providing a clear and concise resource that simplifies the art of database querying, making it accessible to learners of all levels.

The primary motivation behind writing this manual is to serve as a trusted guide for my students and anyone else embarking on the exciting journey of database management and Java programming. With this book, my aim is not only to demystify the intricacies of database querying but also to provide invaluable insights and practical examples that can significantly enhance the skill set of students, particularly those undertaking capstone projects. By mastering the techniques and best practices outlined in this manual, students will not only be better prepared to navigate their capstone projects but also equipped to build robust, real-world applications that require database integration—a skill set that is invaluable in today's technology-driven world. Let's embark on this learning journey together and unlock the potential of Java in the realm of database querying.

DISCLAIMER

THIS BOOK IS STILL A WORK IN PROGRESS.

Stay tuned for the completed version of this book in the future!

SUPPORT MY WORK!

Follow my socials:

Facebook: https://facebook.com/tsadiqz3

Instagram: https://instagram.com/sepiroth.x/

Twitter: https://twitter.com/sepirothx000

Github: https://github.com/sepiroth-x

Patreon: https://www.patreon.com/vajrayogiii

Gcash: +639150388448

Email: richard.cupal@ifamsocial.com , chardy.tsadiq@gmail.com

---

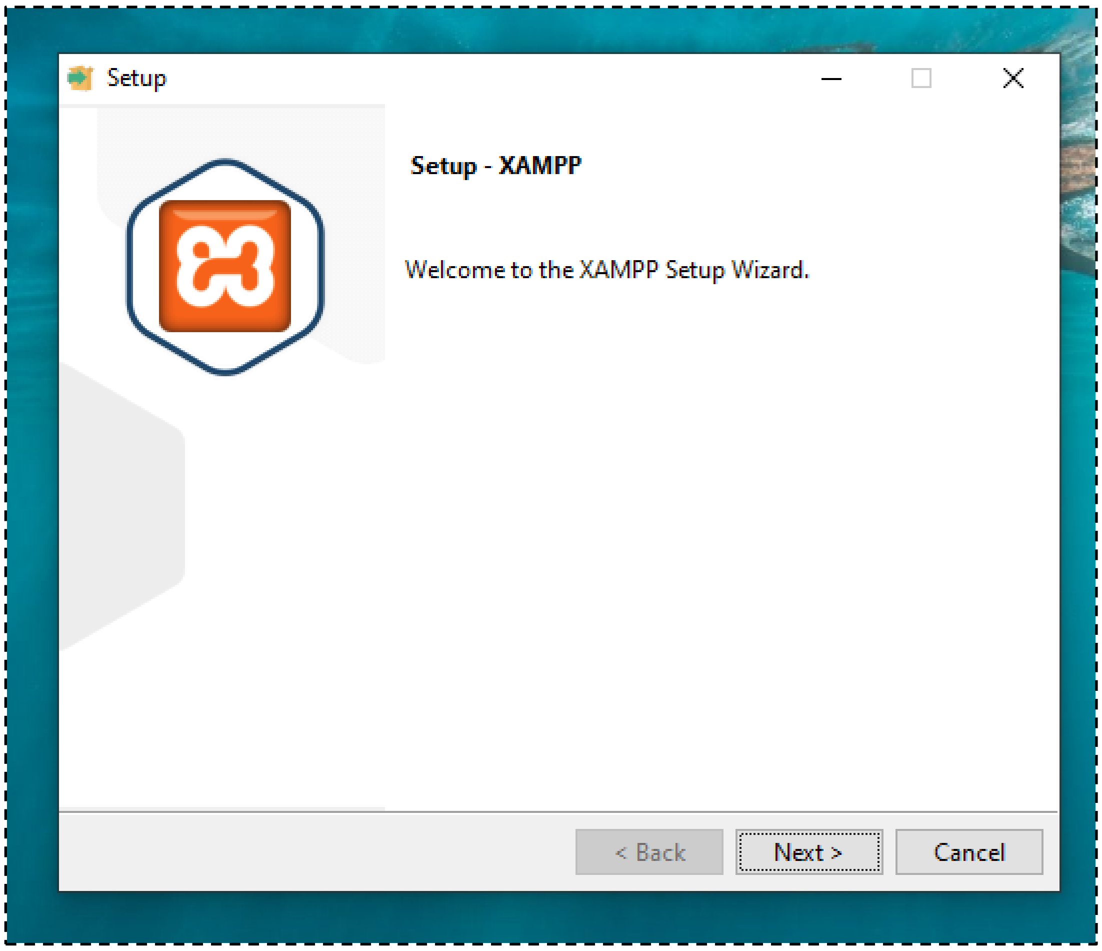
If you like my work, feel free to send some Gcash or be my Patron on Patreon to Support me!

WRITTEN WITH LOVE BY:

SEPIROTH X

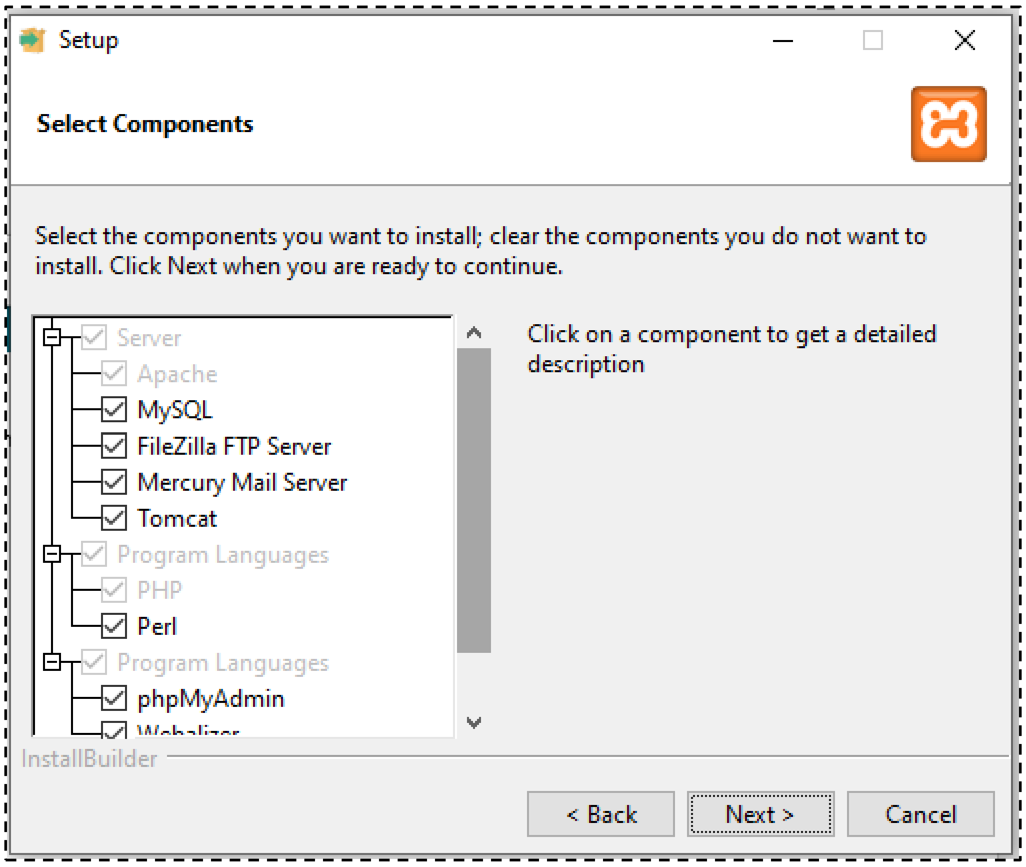
DOWNLOADING NECESSARY TOOLS

Download XAMPP via this link: https://www.apachefriends.org/download_success.html
note: You can use any other database platforms you want. Just search online how to set them up. This tutorial covers the setting up of database via XAMPP ONLY.



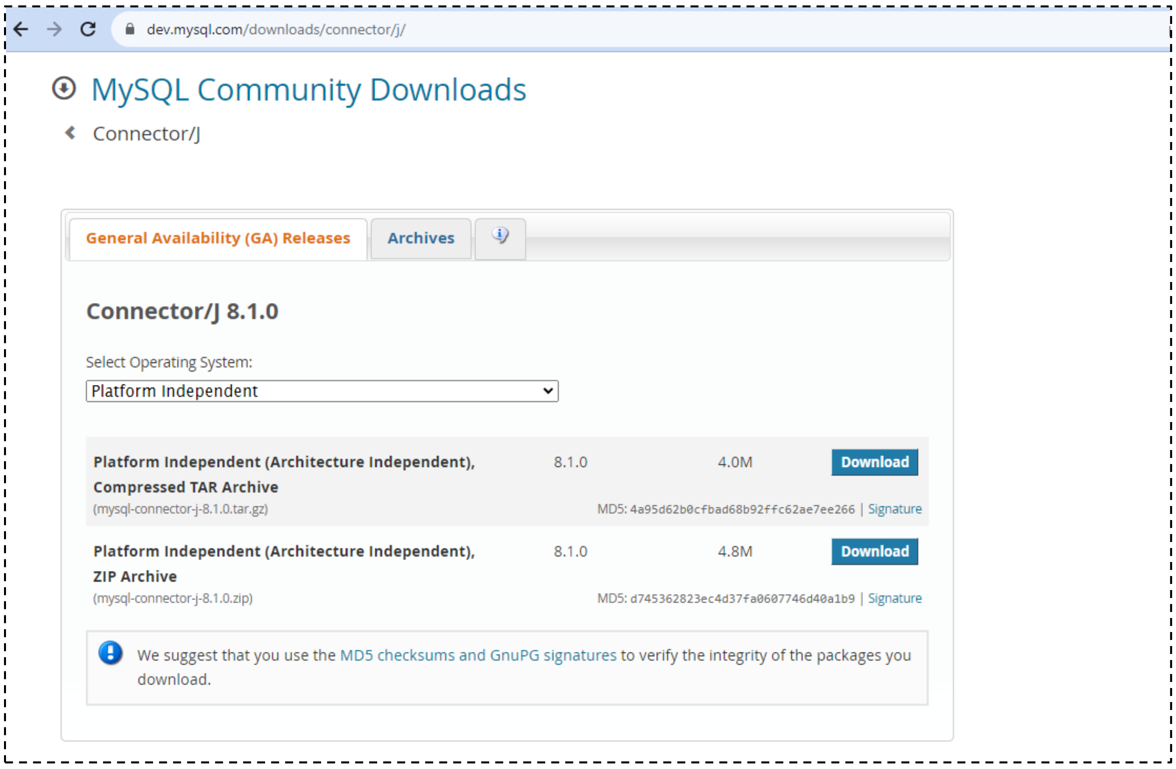
Setting up xampp is a straight forward process. Just click next, use common sense and launch it.

Just click next and finish the installation wizard .


Setup
Select Components
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.
Server
Apache
MySQL
FileZilla FTP Server
Mercury Mail Server
Tomcat
Program Languages
PHP
Perl
Program Languages
phpMyAdmin
Click on a component to get a detailed description
InstallBuilder
< Back
Next >
Cancel

Download Java JDBC Driver Connector/J via this link: https://dev.mysql.com/downloads/connector/j/

Choose **Platform Independent** and download the Zip Archive (not the TAR Archive for it is for Linux)


dev.mysql.com/downloads/connector/j/
MySQL Community Downloads
Connector/J
General Availability (GA) Releases
Archives

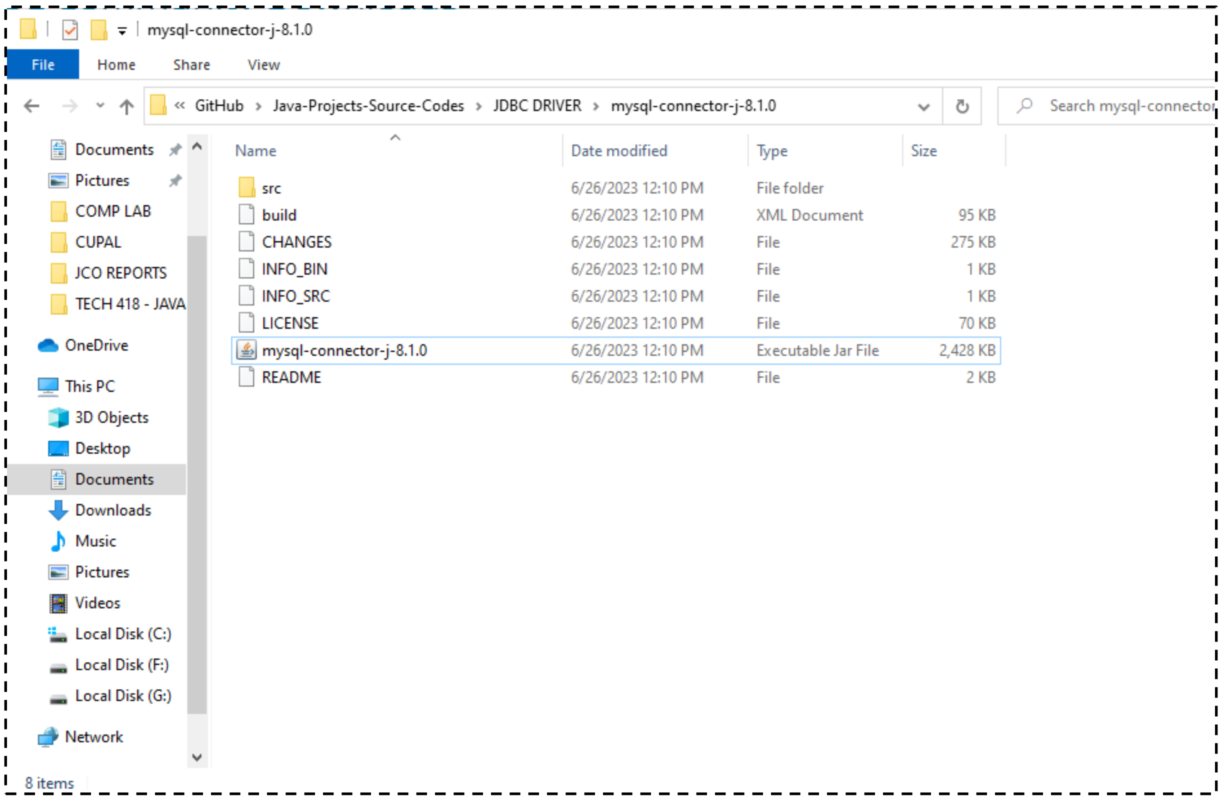
Connector/J 8.1.0

Select Operating System:
Platform Independent

Platform Independent (Architecture Independent), Compressed TAR Archive (mysql-connector-j-8.1.0.tar.gz)	8.1.0	4.0M	Download
MD5: 4a95d62b0cfbad68b92ffc62ae7ee266 | Signature			
Platform Independent (Architecture Independent), ZIP Archive (mysql-connector-j-8.1.0.zip)	8.1.0	4.8M	Download
MD5: d745362823ec4d37fa0607746d40a1b9 | Signature			

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

After downloading, unzip the file anywhere in your desired folder location in your computer.

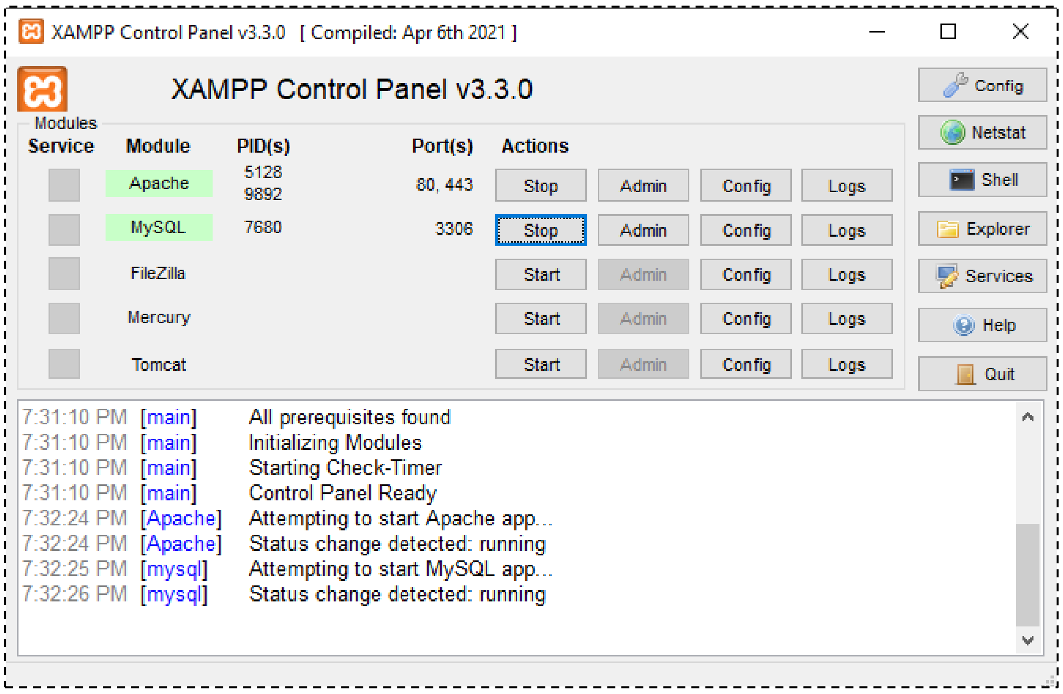


Name	Date modified	Type	Size
src	6/26/2023 12:10 PM	File folder	
build	6/26/2023 12:10 PM	XML Document	95 KB
CHANGES	6/26/2023 12:10 PM	File	275 KB
INFO_BIN	6/26/2023 12:10 PM	File	1 KB
INFO_SRC	6/26/2023 12:10 PM	File	1 KB
LICENSE	6/26/2023 12:10 PM	File	70 KB
mysql-connector-j-8.1.0	6/26/2023 12:10 PM	Executable Jar File	2,428 KB
README	6/26/2023 12:10 PM	File	2 KB

What you see above are the files you see when you extract the downloaded zip file. What you will need is just one single file which is the '**mysql-connector-j-8.1.0.jar**' file

After extracting, just take note of the location of your extracted folder where the mysql-connector-j-8.1.0.jar lives. You will need it later.

Start running your APACHE and MYSQL server via XAMPP



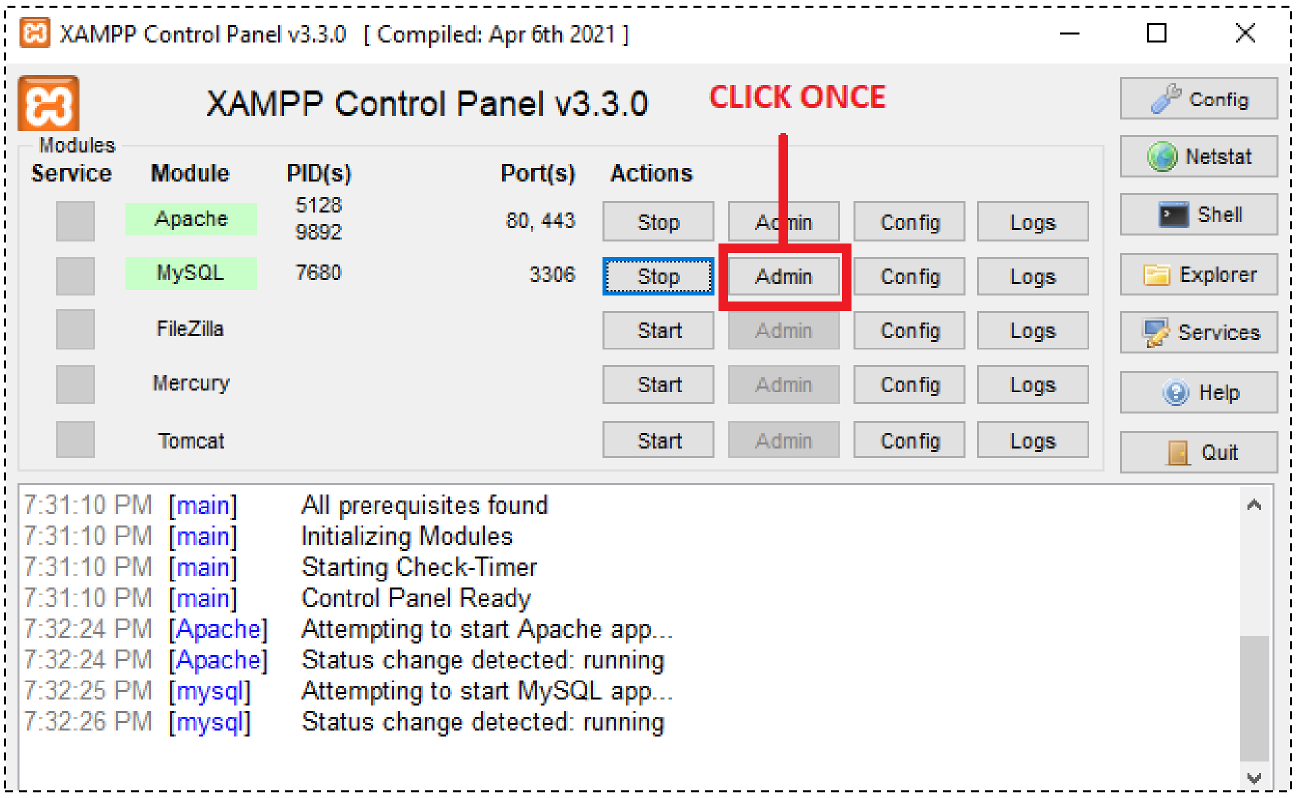
Module	PID(s)	Port(s)	Actions			
Apache	5128 9892	80, 443	Stop	Admin	Config	Logs
MySQL	7680	3306	Stop	Admin	Config	Logs
FileZilla			Start	Admin	Config	Logs
Mercury			Start	Admin	Config	Logs
Tomcat			Start	Admin	Config	Logs

```
7:31:10 PM  [main]     All prerequisites found
7:31:10 PM  [main]     Initializing Modules
7:31:10 PM  [main]     Starting Check-Timer
7:31:10 PM  [main]     Control Panel Ready
7:32:24 PM  [Apache]   Attempting to start Apache app...
7:32:24 PM  [Apache]   Status change detected: running
7:32:25 PM  [mysql]    Attempting to start MySQL app...
7:32:26 PM  [mysql]    Status change detected: running
```

You will know them running when you see the green color on their labels as well as the PID(s) and Port(s) being used . . . .

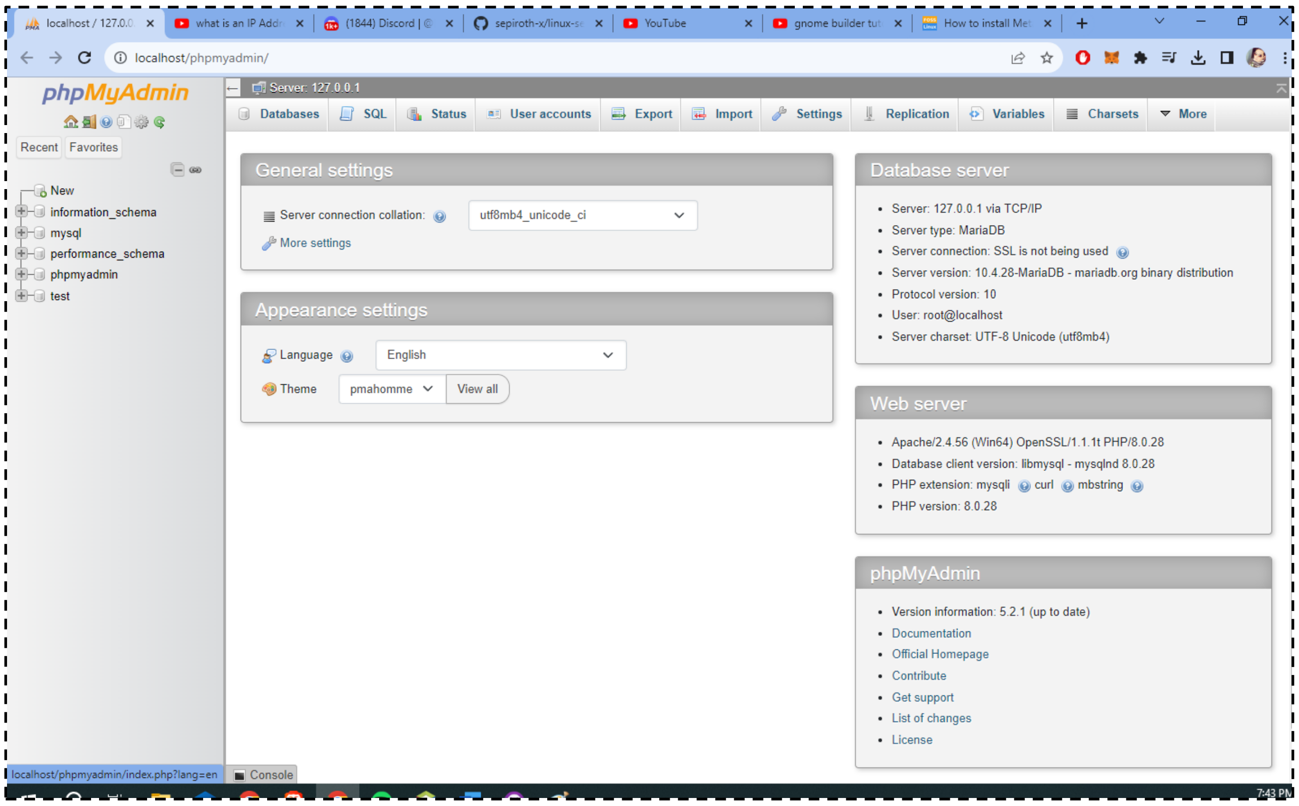
CREATING YOUR DATABASE IN PHPMYADMIN

To create database via PHPMYADMIN, head over on your XAMPP CONTROL PANEL and click on the Admin button on MySQL's action row as seen in the image below:



Module	PID(s)	Port(s)	Actions			
Apache	5128 9892	80, 443	Stop	Admin	Config	Logs
MySQL	7680	3306	Stop	Admin	Config	Logs
FileZilla			Start	Admin	Config	Logs
Mercury			Start	Admin	Config	Logs
Tomcat			Start	Admin	Config	Logs

```
7:31:10 PM  [main]     All prerequisites found
7:31:10 PM  [main]     Initializing Modules
7:31:10 PM  [main]     Starting Check-Timer
7:31:10 PM  [main]     Control Panel Ready
7:32:24 PM  [Apache]   Attempting to start Apache app...
7:32:24 PM  [Apache]   Status change detected: running
7:32:25 PM  [mysql]    Attempting to start MySQL app...
7:32:26 PM  [mysql]    Status change detected: running
```



General settings

Server connection collation: utf8mb4_unicode_ci

More settings

Appearance settings

Language: English

Theme: pmahomme View all

Database server

- Server: 127.0.0.1 via TCP/IP
- Server type: MariaDB
- Server connection: SSL is not being used
- Server version: 10.4.28-MariaDB - mariadb.org binary distribution
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8mb4)

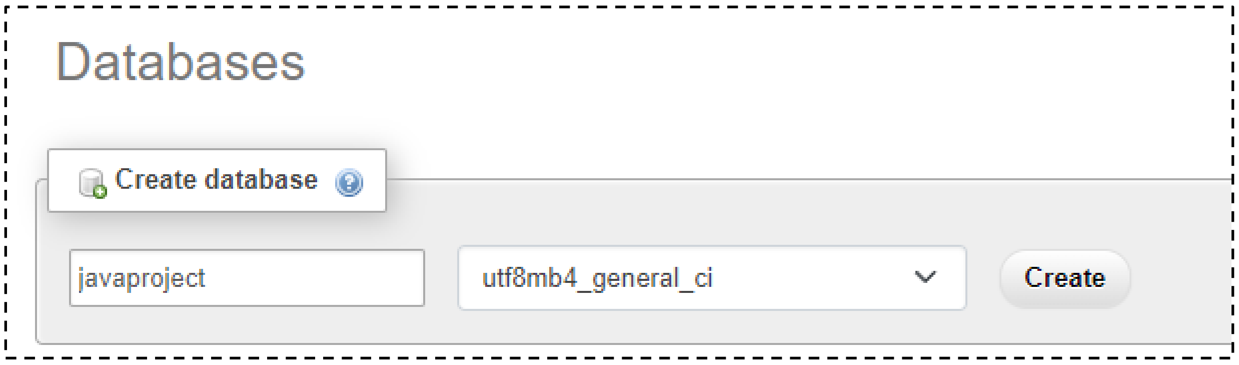
Web server

- Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.0.28
- Database client version: libmysql - mysqlnd 8.0.28
- PHP extension: mysqli curl mbstring
- PHP version: 8.0.28

phpMyAdmin

- Version information: 5.2.1 (up to date)
- Documentation
- Official Homepage
- Contribute
- Get support
- List of changes
- License

Click on NEW to create new database. I will not walk you through on what columns and fields to create for your design might be different than mine. For the sake of the tutorial, you can just follow the design of my database as follows:



javaproject

utf8mb4_general_ci

Let's say I will create a table having four (4) columns for the following fields

- **Id**
- **First Name**
- **Last Name**
- **School ID**

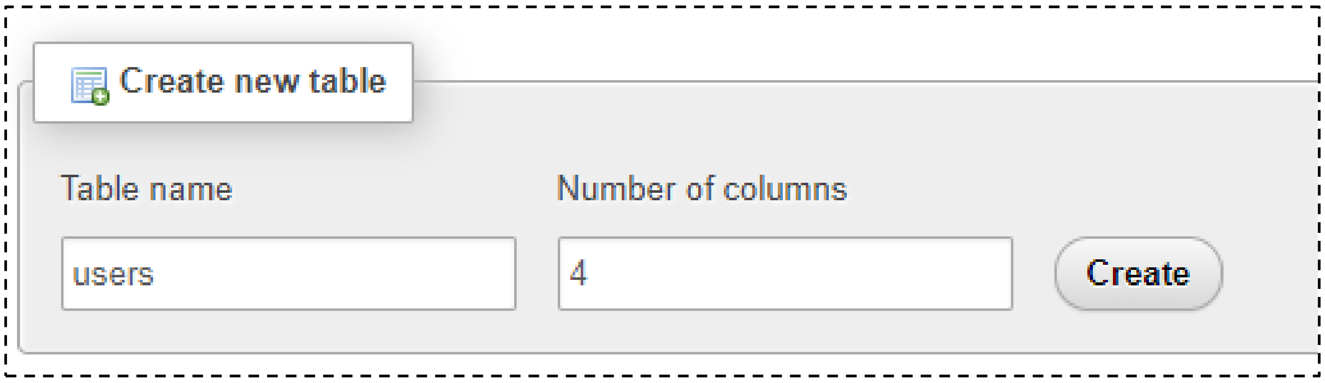


Table name: users

Number of columns: 4

Click **'Create'** to create the table.

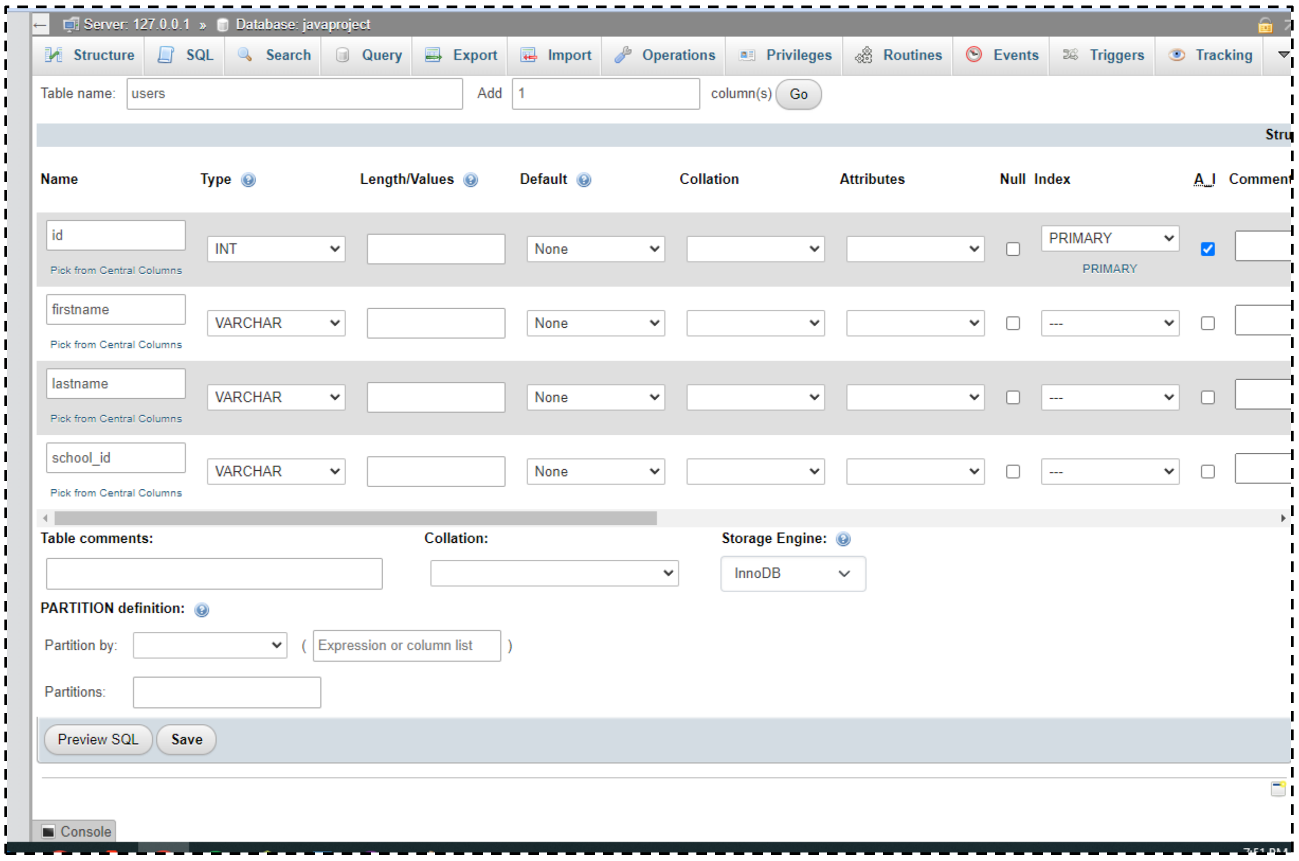

Server: 127.0.0.1 » Database: javaproject
Structure SQL Search Query Export Import Operations Privileges Routines Events Triggers Tracking

Table name: users Add 1 column(s) Go

Name	Type	Length/Values	Default	Collation	Attributes	Null	Index	A_I
id	INT		None			☐	PRIMARY	✓
firstname	VARCHAR		None			☐	---	☐
lastname	VARCHAR		None			☐	---	☐
school_id	VARCHAR		None			☐	---	☐

Table comments: Collation: Storage Engine: InnoDB

PARTITION definition: Partition by: (Expression or column list) Partitions:

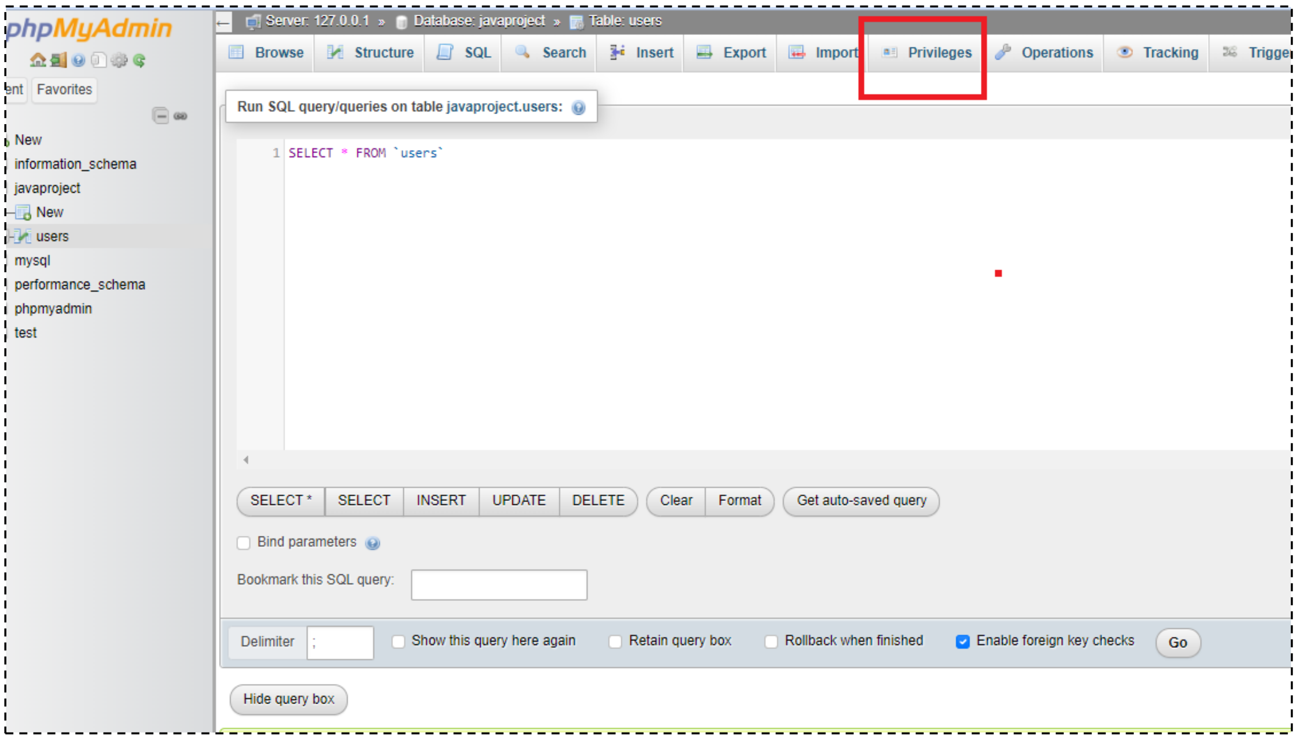
You can set the data types for the following fields (feel free to change in accordance to your needs)

- id(int) -> This field must be set to A_I (Auto Increment) and set as PRIMARY (for PRIMARY key)
- firstname(varchar) -> this field must contain length/values (mine is set to 16 characters)
- lastname(varchar) -> this field must contain length/values (mine is set to 16 characters)
- school_id(varchar) -> this field must contain length/values (mine is set to 32 characters)

For field names, it is better to follow the convention to use small letters. There should be NO SPACES. For multiple-word field names, you can just use underscore (_) instead of spaces. Click **'Save'**.

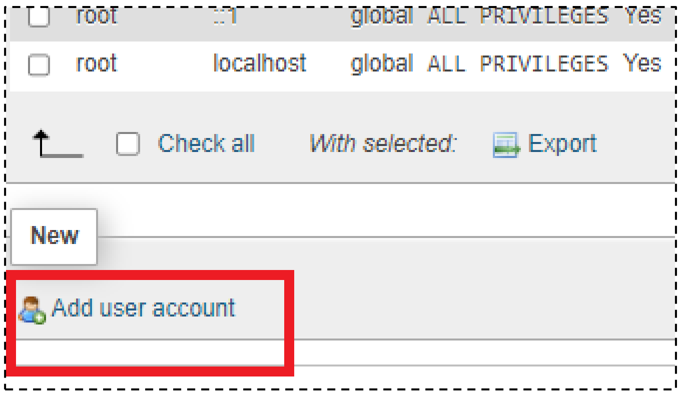
CREATING USER ACCOUNT TO ACCESS THE DATABASE

In PHPMYADMIN, create a user account that can access the databases by clicking on the Privileges tab as seen below:



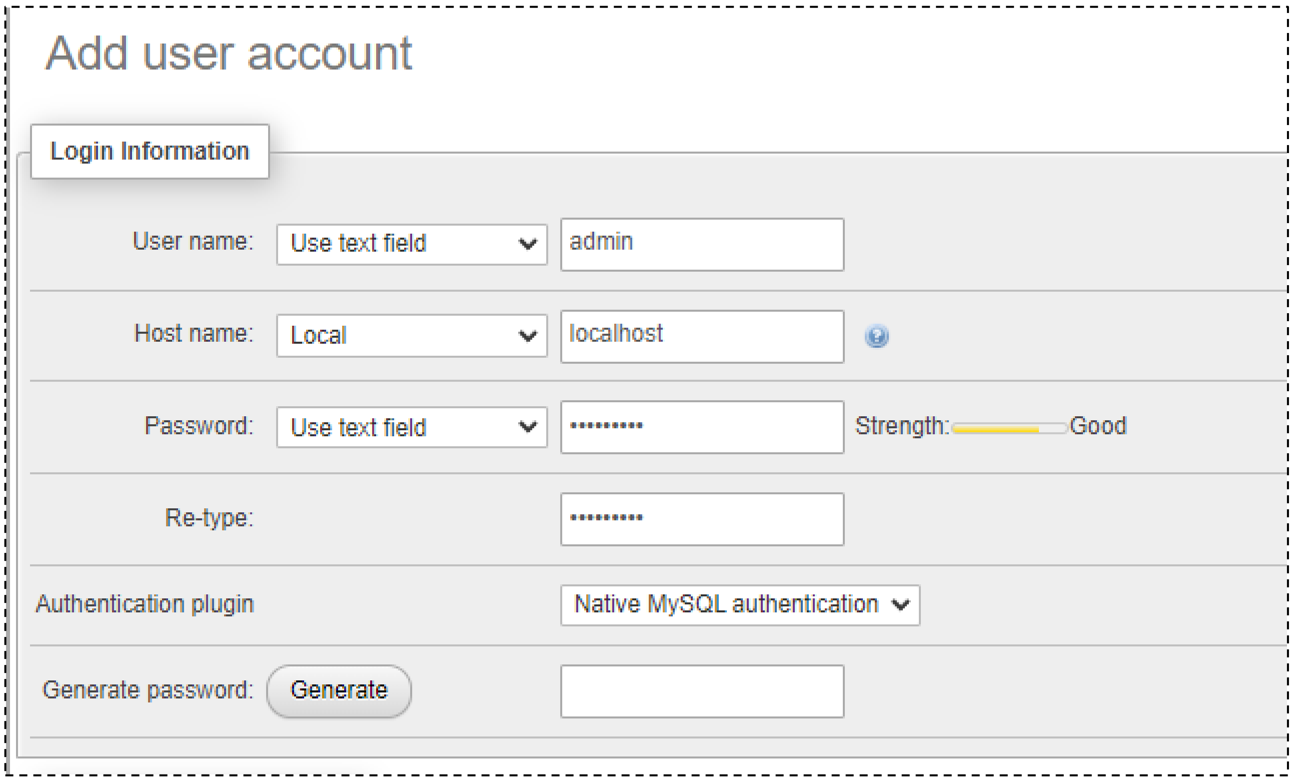
```
SELECT * FROM `users`
```

Click on the 'Add user account' under the 'New' section as seen below:



root	::1	global	ALL PRIVILEGES	Yes
root	localhost	global	ALL PRIVILEGES	Yes

Follow the settings below by setting username field as ‘Use text field’ , for Host Name as ‘Local’ types as seen in the image below:



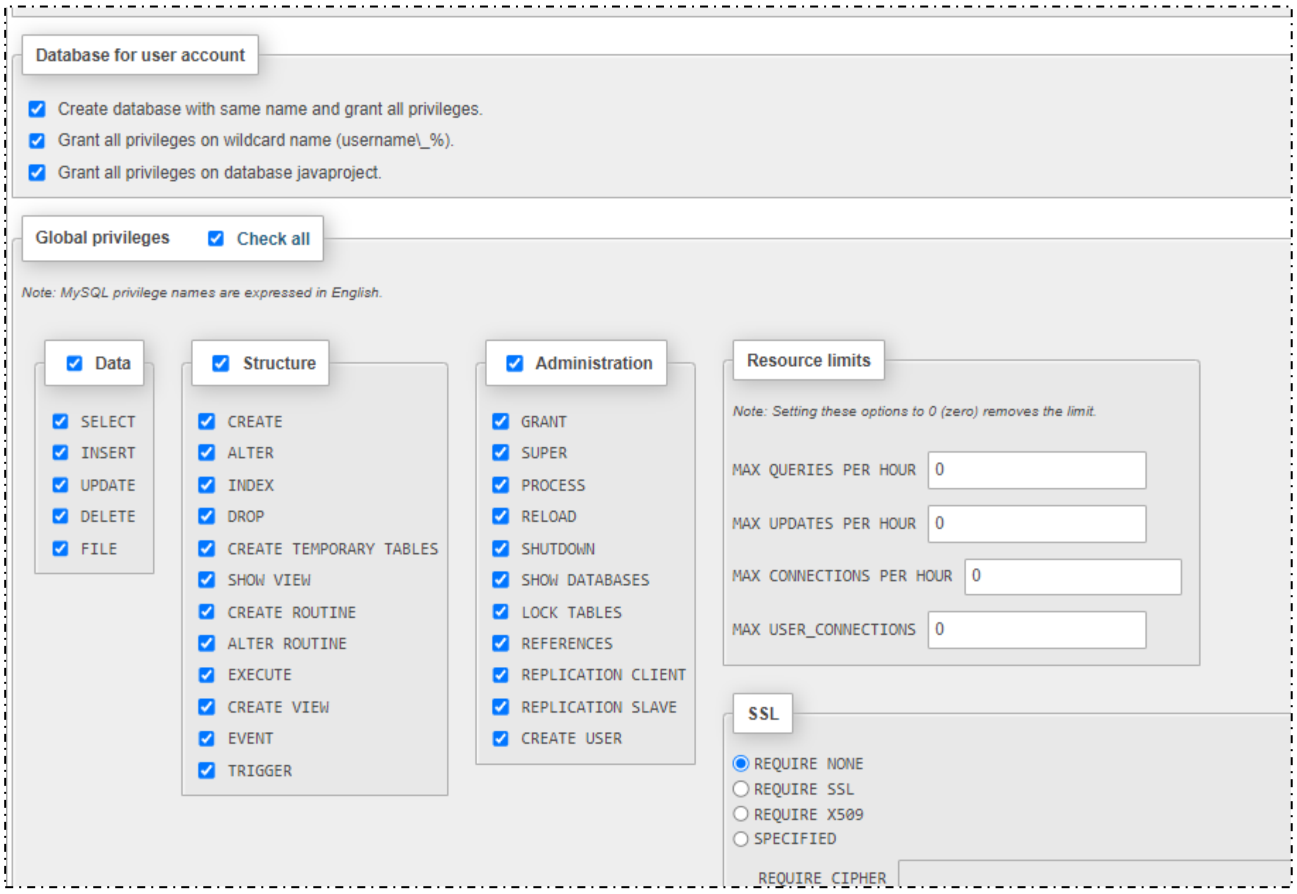
Add user account

Login Information

User name:	Use text field	admin
Host name:	Local	localhost
Password:	Use text field	••••••••• Strength: Good
Re-type:		•••••••••
Authentication plugin		Native MySQL authentication
Generate password:	Generate	

For the credentials, you can enter your own. In my case, for this example, I entered **admin** as username and **p@ssword**! For the password. This account should not be shared publicly for this holds the access of our application’s database.

After filling out the credentials, click all the available check marks under Database for user account and Global Privilges as seen below. This part is granting global access of the account for databases.



Database for user account

- ✓ Create database with same name and grant all privileges.
- ✓ Grant all privileges on wildcard name (username_%).
- ✓ Grant all privileges on database javaproject.

Global privileges ✓ Check all

Note: MySQL privilege names are expressed in English.

✓ Data
- ✓ SELECT
- ✓ INSERT
- ✓ UPDATE
- ✓ DELETE
- ✓ FILE

✓ Structure
- ✓ CREATE
- ✓ ALTER
- ✓ INDEX
- ✓ DROP
- ✓ CREATE TEMPORARY TABLES
- ✓ SHOW VIEW
- ✓ CREATE ROUTINE
- ✓ ALTER ROUTINE
- ✓ EXECUTE
- ✓ CREATE VIEW
- ✓ EVENT
- ✓ TRIGGER

✓ Administration
- ✓ GRANT
- ✓ SUPER
- ✓ PROCESS
- ✓ RELOAD
- ✓ SHUTDOWN
- ✓ SHOW DATABASES
- ✓ LOCK TABLES
- ✓ REFERENCES
- ✓ REPLICATION CLIENT
- ✓ REPLICATION SLAVE
- ✓ CREATE USER

Resource limits

Note: Setting these options to 0 (zero) removes the limit.

MAX QUERIES PER HOUR	0
MAX UPDATES PER HOUR	0
MAX CONNECTIONS PER HOUR	0
MAX USER_CONNECTIONS	0

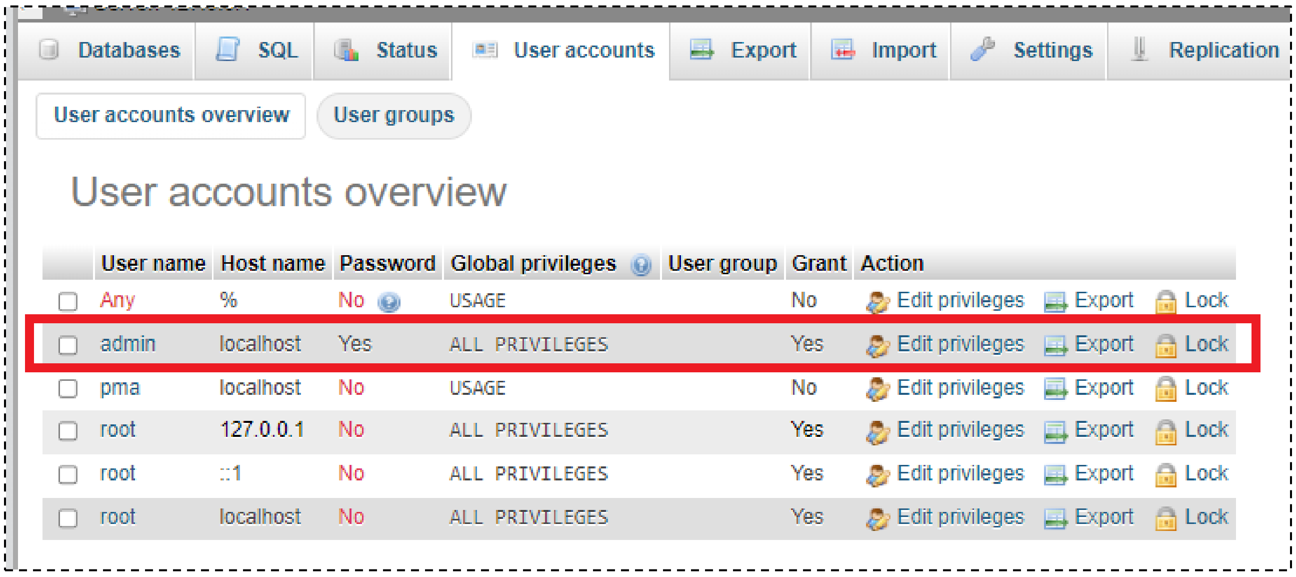
SSL

- ● REQUIRE NONE
- ○ REQUIRE SSL
- ○ REQUIRE X509
- ○ SPECIFIED

REQUIRE CIPHER

Click ‘**Go’** button to create the account.

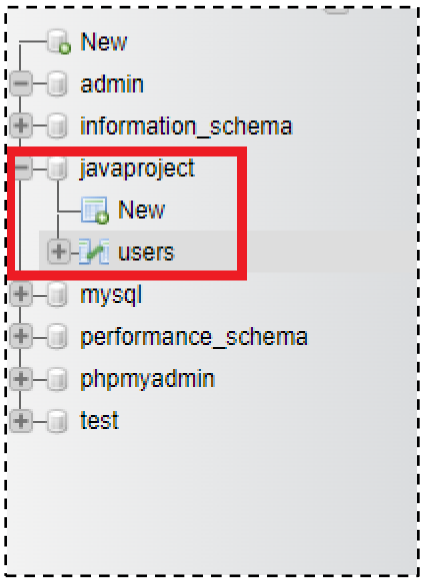
After the account has been created, you can double check its existence and privileges granted under User accounts tab.



User accounts overview

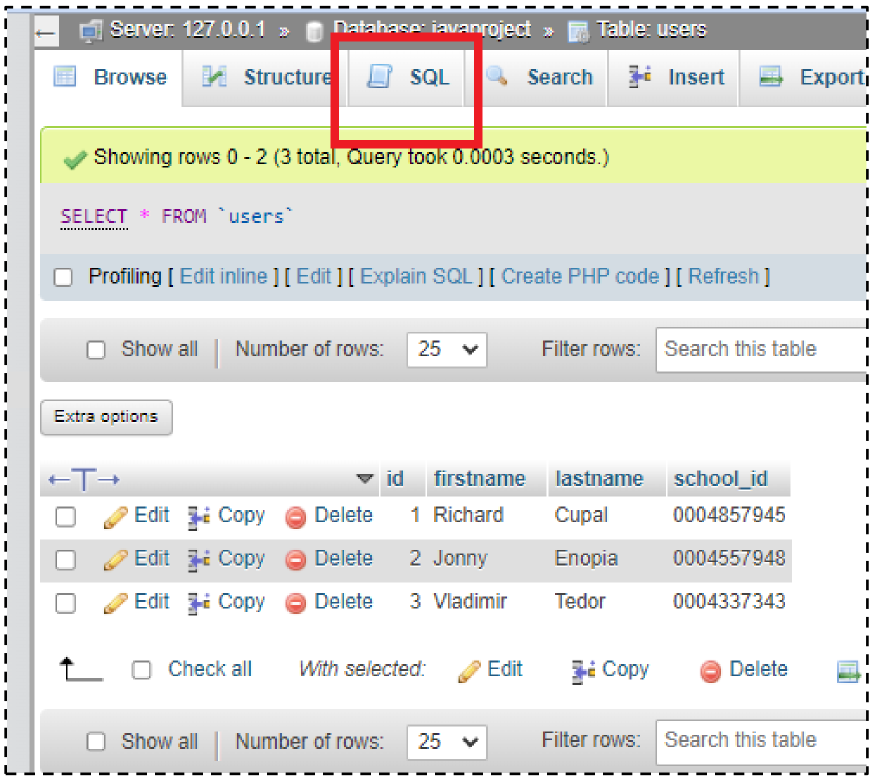
	User name	Host name	Password	Global privileges	User group	Grant	Action		
☐	Any	%	No	USAGE		No	Edit privileges	Export	Lock
☐	admin	localhost	Yes	ALL PRIVILEGES		Yes	Edit privileges	Export	Lock
☐	pma	localhost	No	USAGE		No	Edit privileges	Export	Lock
☐	root	127.0.0.1	No	ALL PRIVILEGES		Yes	Edit privileges	Export	Lock
☐	root	::1	No	ALL PRIVILEGES		Yes	Edit privileges	Export	Lock
☐	root	localhost	No	ALL PRIVILEGES		Yes	Edit privileges	Export	Lock

You can then check your database and the table you created on the left side of the panel. Your table has no data yet, so we need to populate sample data into it first so we can do this activity of querying the data inside the database.



When you click your database and the table inside it, you can see no data in it yet.

To start populating, click on the SQL tab and type the SQL command to insert records to the database.


Server: 127.0.0.1 » Database: javaproject » Table: users
Browse
Structure
SQL
Search
Insert
Export

Showing rows 0 - 2 (3 total, Query took 0.0003 seconds.)

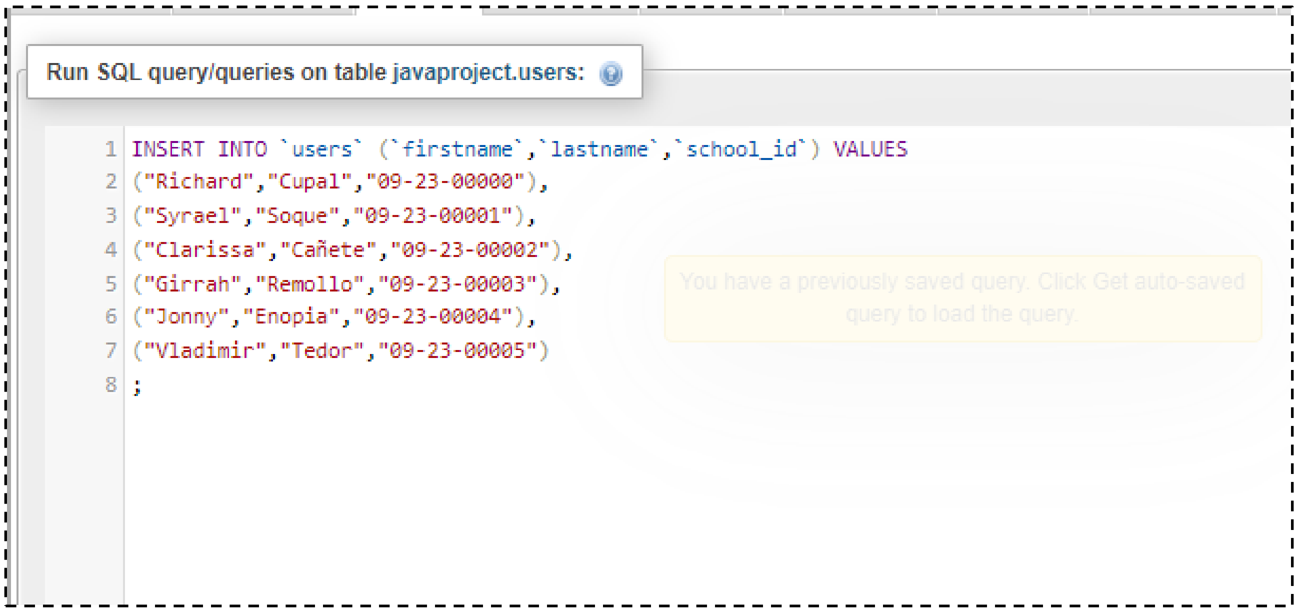
```
SELECT * FROM `users`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table
Extra options

				id	firstname	lastname	school_id
☐	Edit	Copy	Delete	1	Richard	Cupal	0004857945
☐	Edit	Copy	Delete	2	Jonny	Enopia	0004557948
☐	Edit	Copy	Delete	3	Vladimir	Tedor	0004337343

Check all With selected: Edit Copy Delete
Show all | Number of rows: 25 Filter rows: Search this table

You can follow my query below however feel free to change the data to populate inside your database. See image below:



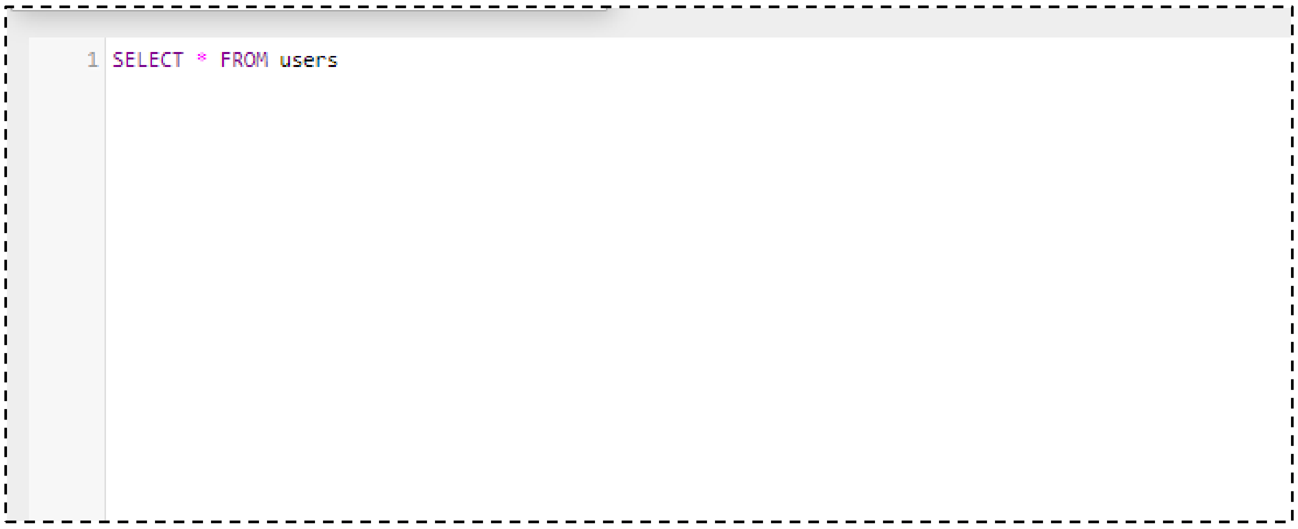
Run SQL query/queries on table javaproject.users:

```sql
INSERT INTO `users` (`firstname`,`lastname`,`school_id`) VALUES
("Richard","Cupal","09-23-00000"),
("Syrael","Soque","09-23-00001"),
("Clarissa","Cañete","09-23-00002"),
("Girrah","Remollo","09-23-00003"),
("Jonny","Enopia","09-23-00004"),
("Vladimir","Tedor","09-23-00005")
;
```

You have a previously saved query. Click Get auto-saved query to load the query.

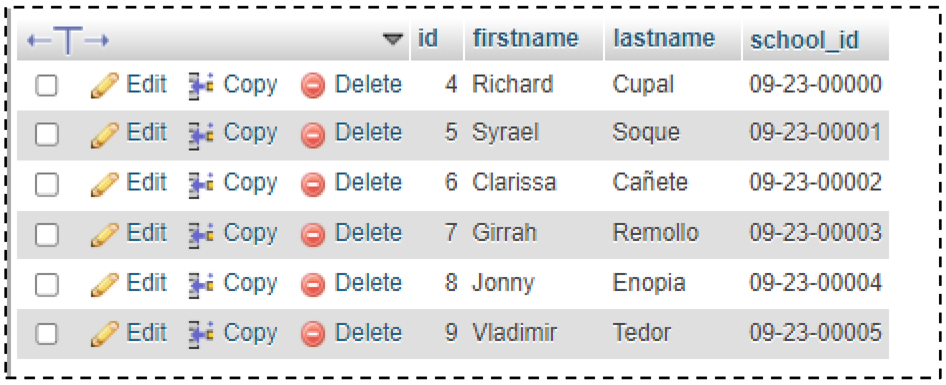
Click ‘Go’ to run the query and populate your table with the necessary data.

You can then run this query to check the data inside your database.



```sql
SELECT * FROM users
```

The data populated the database and it is confirmed by the result of the above query as seen below:

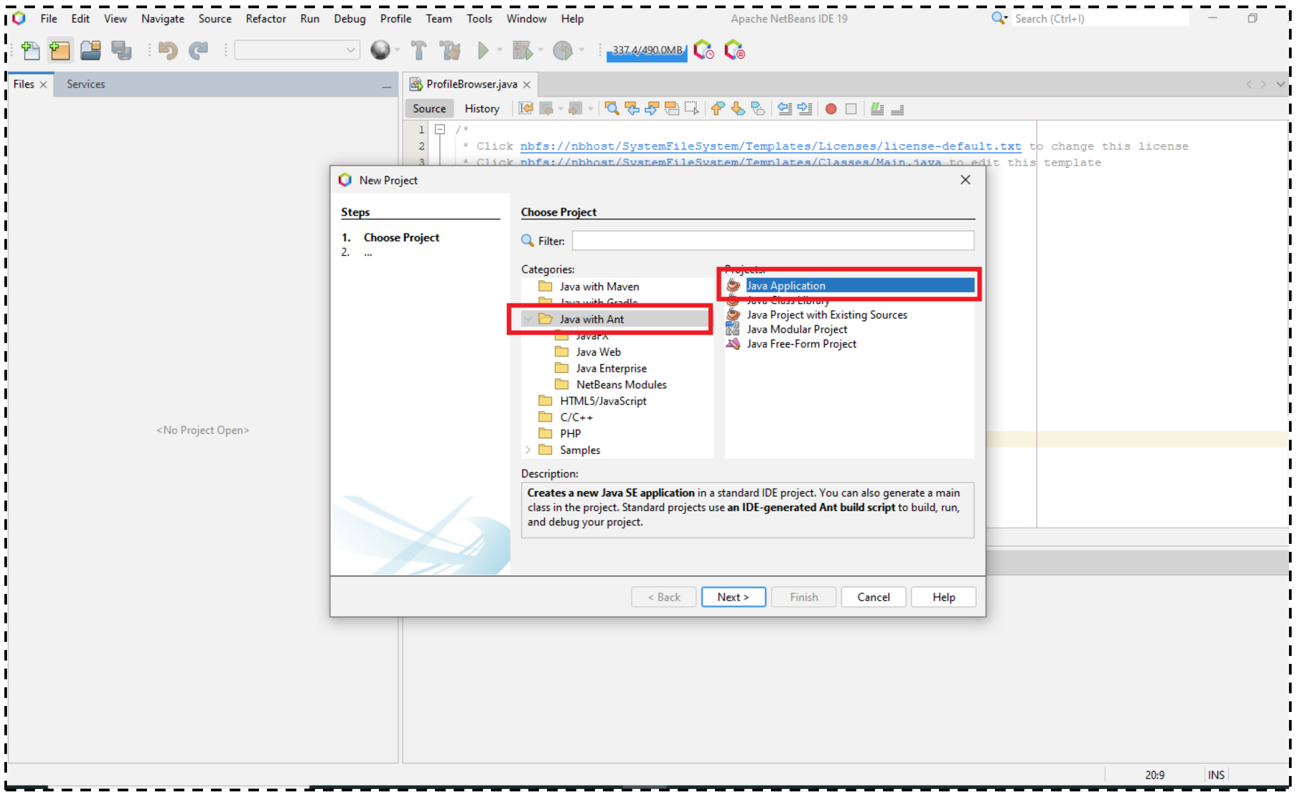


				id	firstname	lastname	school_id
☐	Edit	Copy	Delete	4	Richard	Cupal	09-23-00000
☐	Edit	Copy	Delete	5	Syrael	Soque	09-23-00001
☐	Edit	Copy	Delete	6	Clarissa	Cañete	09-23-00002
☐	Edit	Copy	Delete	7	Girrah	Remollo	09-23-00003
☐	Edit	Copy	Delete	8	Jonny	Enopia	09-23-00004
☐	Edit	Copy	Delete	9	Vladimir	Tedor	09-23-00005

If you’re confused why the id numbers of my above example start with #4, it is because I have inserted data in it before. Since I have set that id column as auto-increment, it will naturally increment its number every time a data is added.

Now, our database looks good. We now have the necessary data to pull in our application coming from our database.

CREATE YOUR APPLICATION PROJECT IN NETBEANS



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
```

New Project

Steps
1. Choose Project
2. ...

Choose Project

Filter:

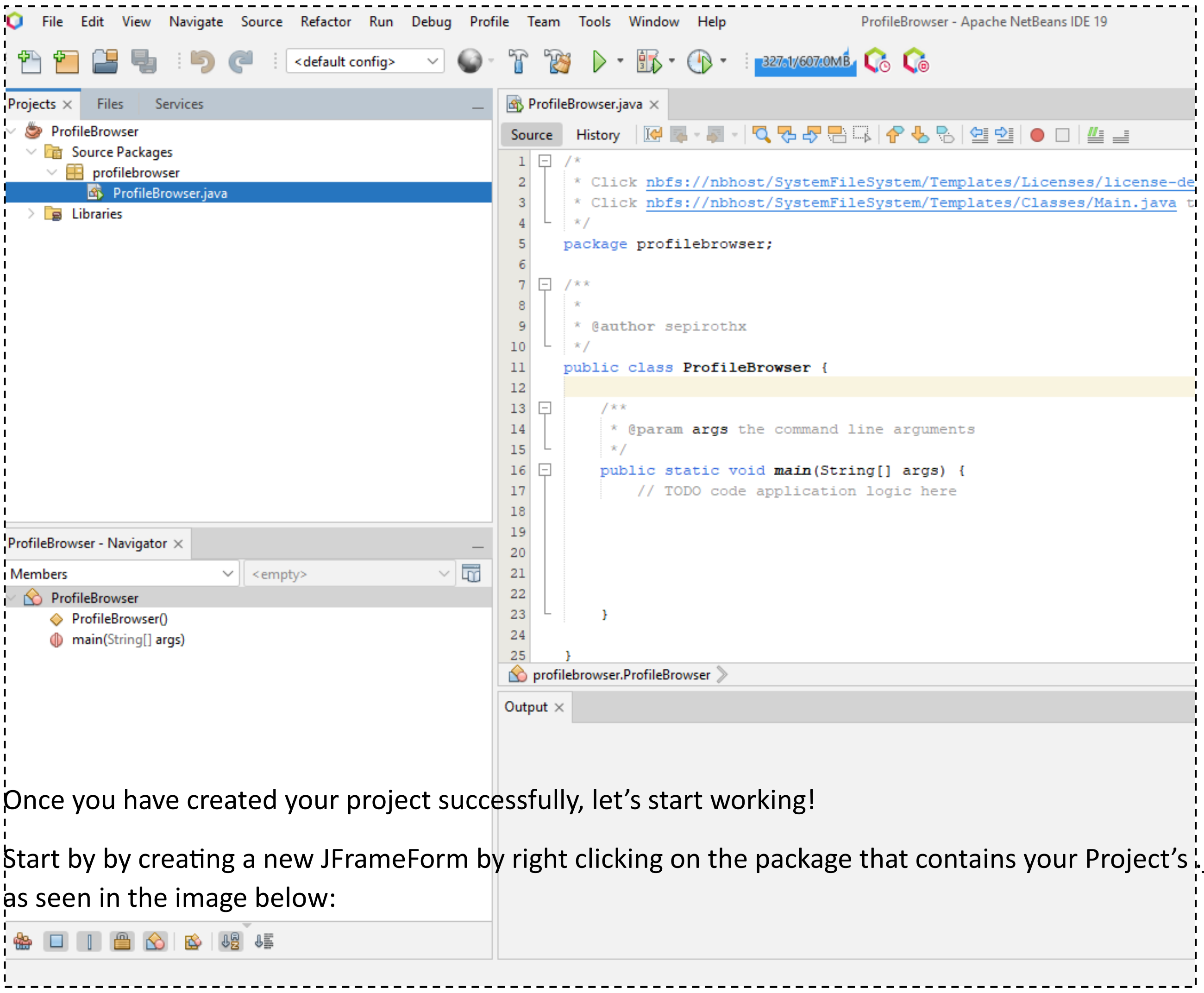
Categories:
- Java with Maven
- Java with Gradle
- Java with Ant
  - JavaFX
  - Java Web
  - Java Enterprise
  - NetBeans Modules
- HTML5/JavaScript
- C/C++
- PHP
- Samples

Projects:
- Java Application
- Java Class Library
- Java Project with Existing Sources
- Java Modular Project
- Java Free-Form Project

Description:
Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

Create your Java Project in Netbeans by clicking on the New Project > Java with Ant > Java Application.

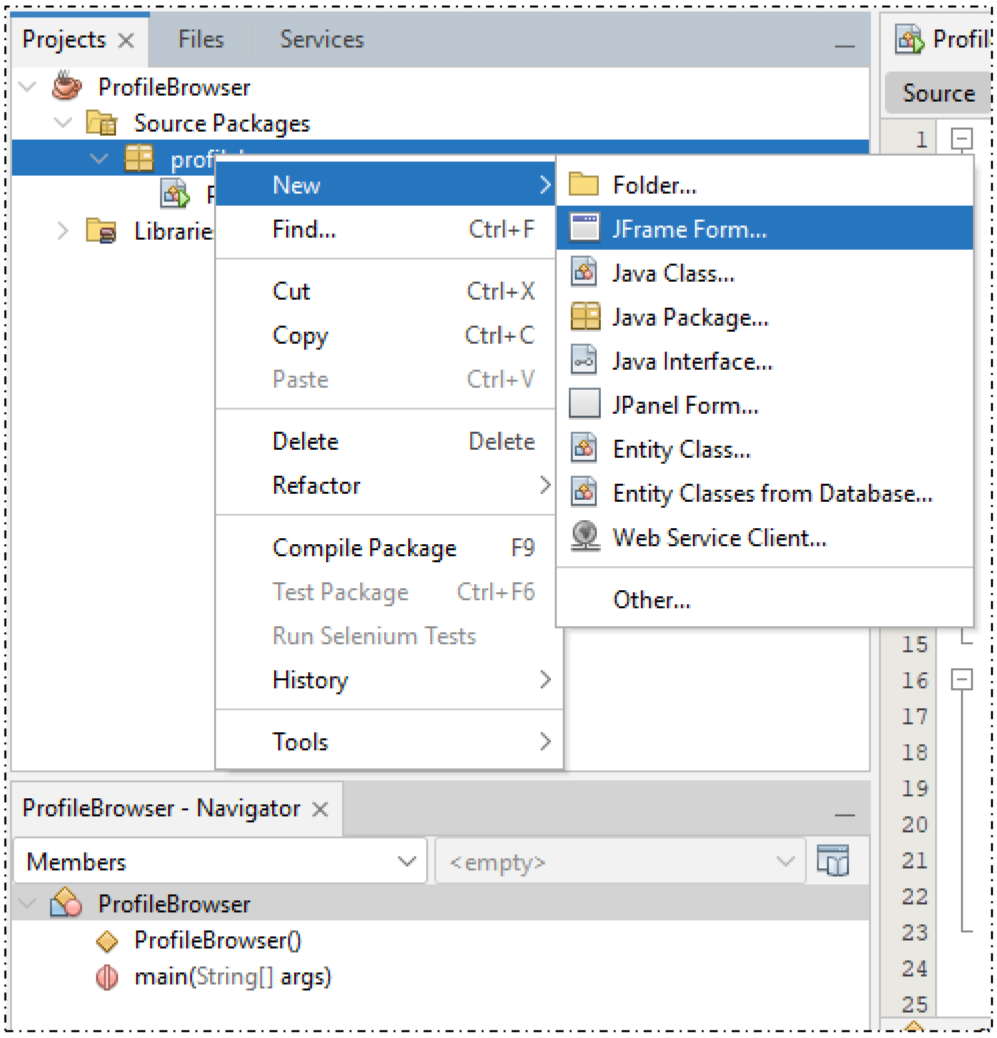
Let's use Java with Ant build tool this time so we can literally and explicitly import our mysql jdbc driver connector/j file in our project. You can also do this with other build tools such as Maven and Gradle but I prefer Ant for this tutorial so you can literally see how to add the jdbc driver connector/j very easily.

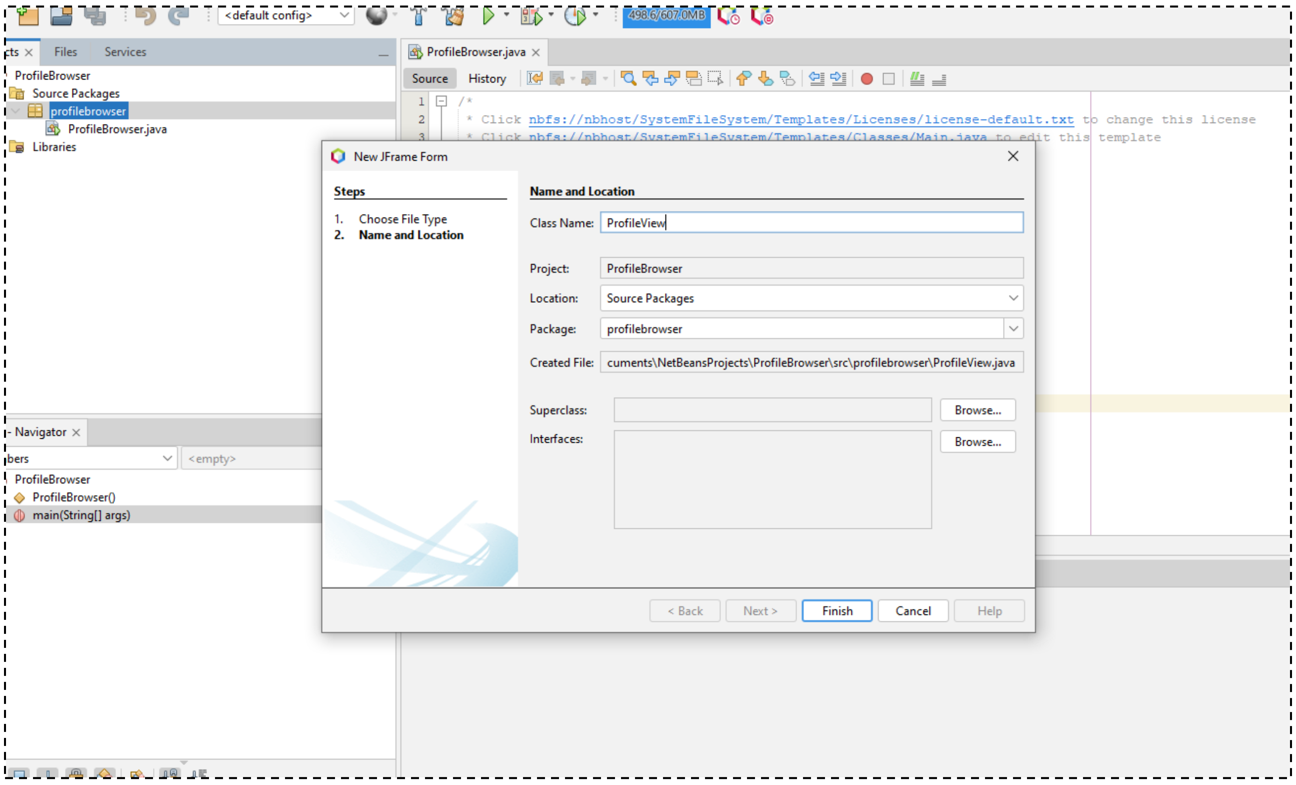


```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-de
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java t
 */
package profilebrowser;

/**
 *
 * @author sepirothx
 */
public class ProfileBrowser {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here




    }

}
```

Once you have created your project successfully, let's start working!

Start by by creating a new JFrameForm by right clicking on the package that contains your Project's .java class as seen in the image below:





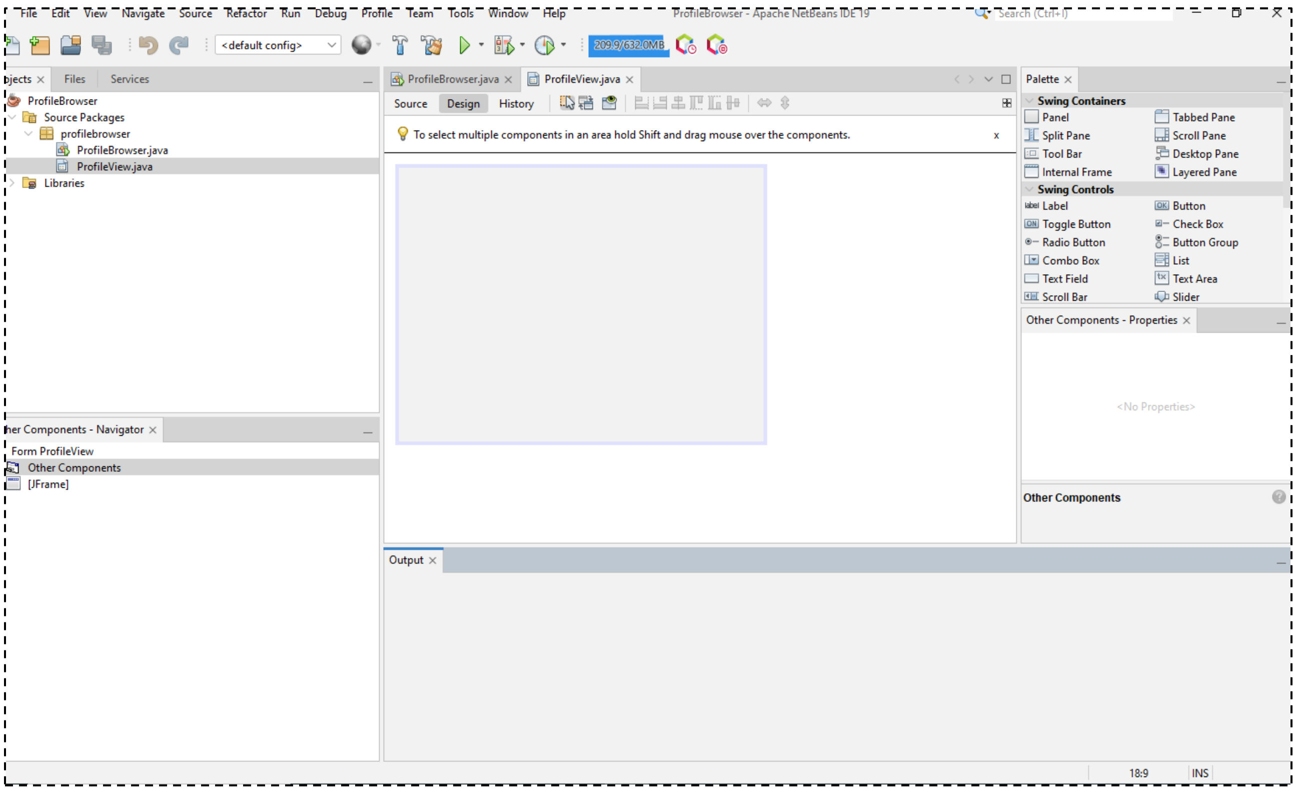
```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
```

Class Name: ProfileView
Project: ProfileBrowser
Location: Source Packages
Package: profilebrowser
Created File: cuments\NetBeansProjects\ProfileBrowser\src\profilebrowser\ProfileView.java
Superclass:
Interfaces:

For the new jframe form, you can give it name as mine but feel free to use any other name you want. Just be sure to start it with capital letter (good practice / convention) and no spaces in between words. In my case, it's **ProfileView**. Click **Finish.**

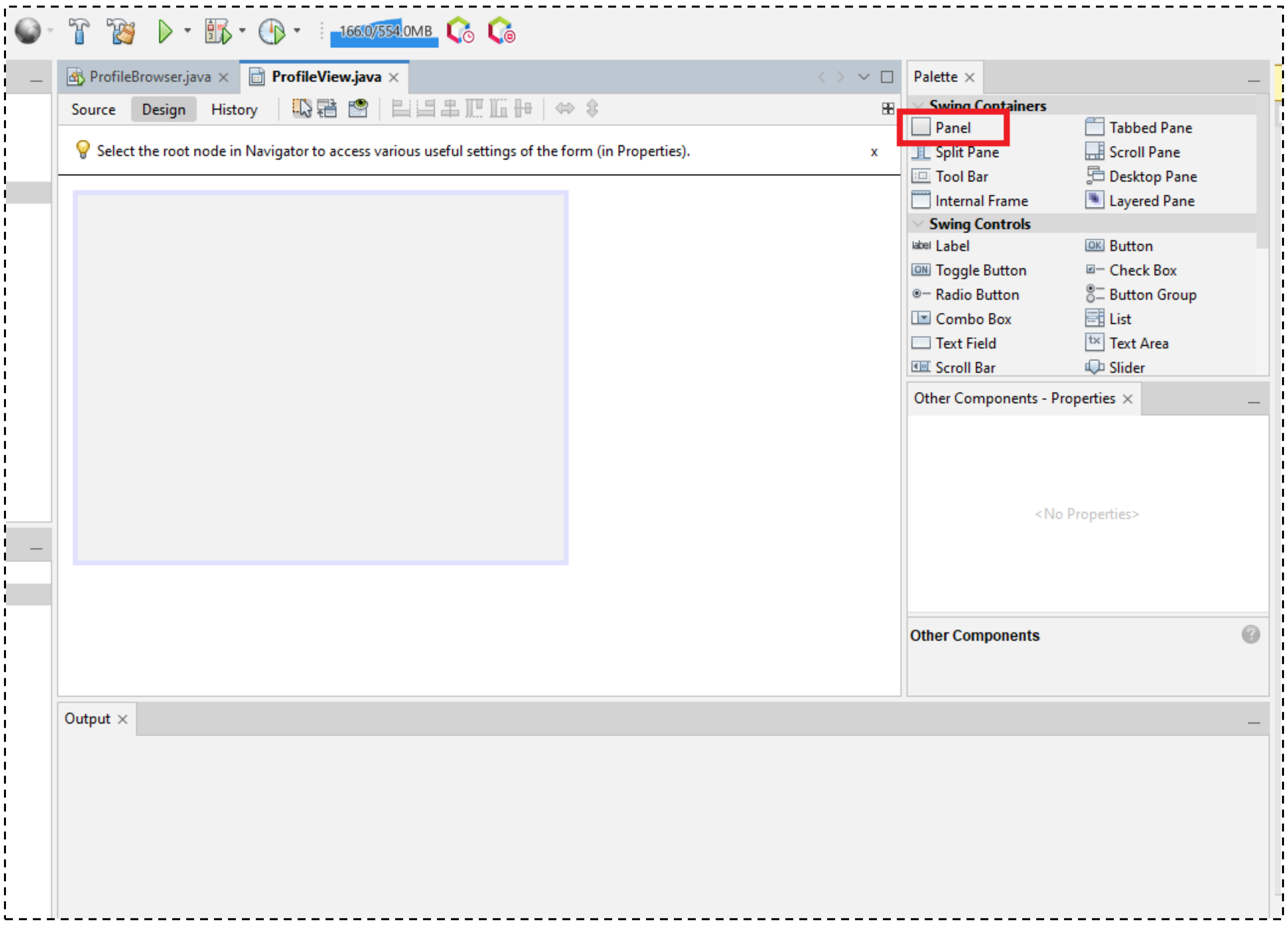
DESIGNING THE GRAPHICAL USER INTERFACE

Now, you can start designing your UI. Just click hold and drag the necessary objects you can find inside the Palette on the right side panel of the netbeans application.

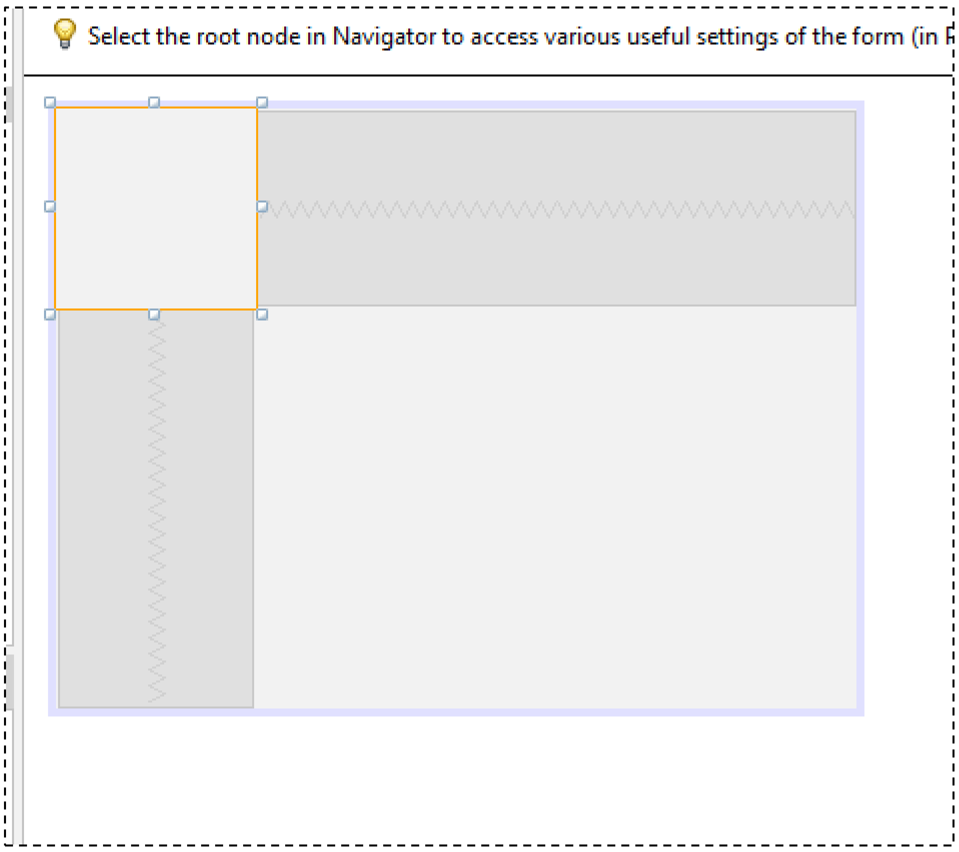


To select multiple components in an area hold Shift and drag mouse over the components.

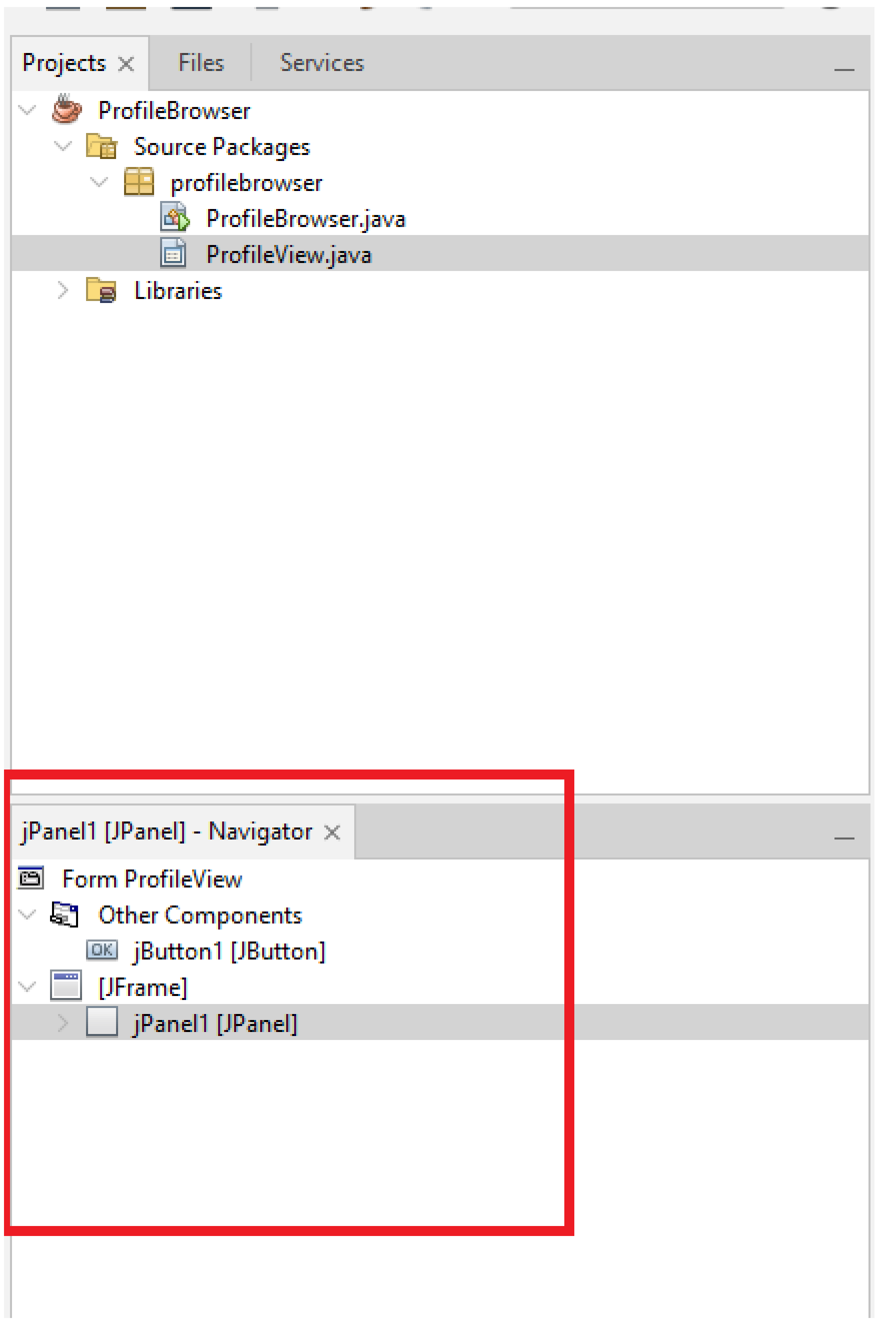
Before you add any other objects inside your jframe form, be sure to add a Panel first and make it fit the whole container of the Jframe Form. Add the Panel by click, hold and drag it inside the JFrame.



Select the root node in Navigator to access various useful settings of the form (in Properties).

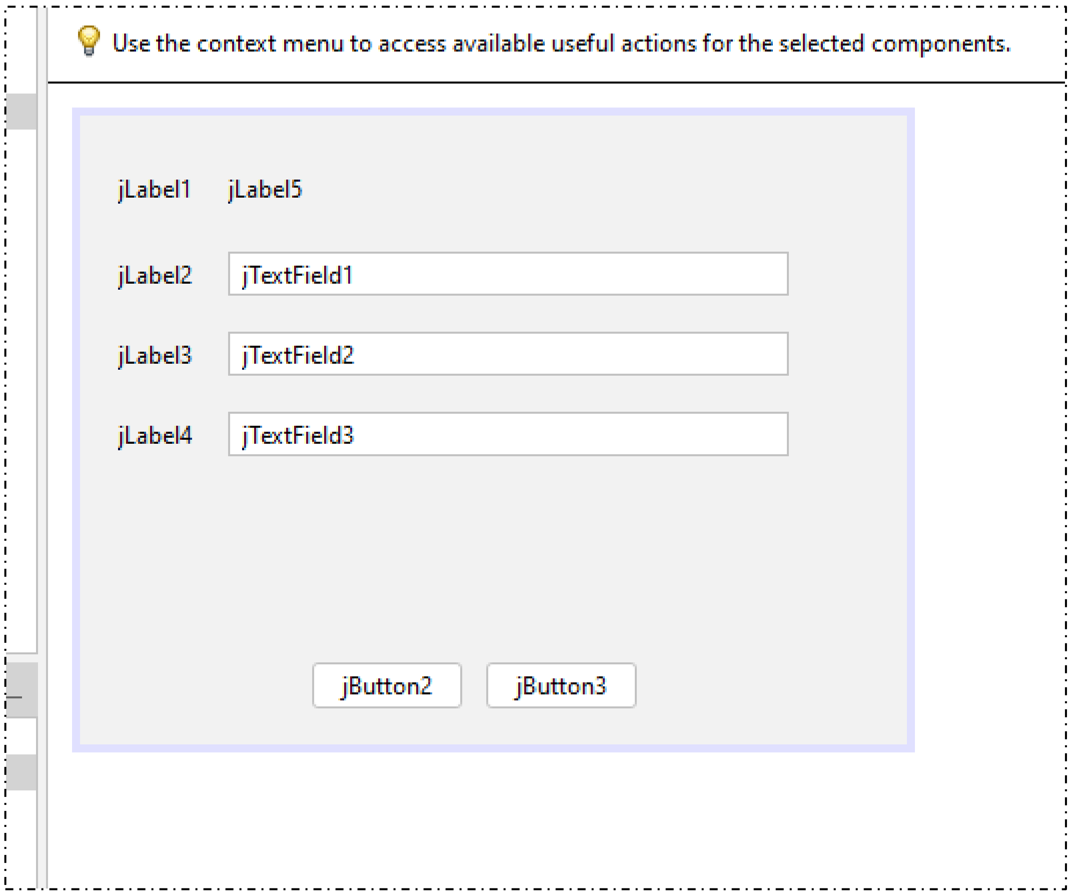

Select the root node in Navigator to access various useful settings of the form (in F

Click and drag the nodes of the Panel to adjust its size and fit it inside the JFrame Form.



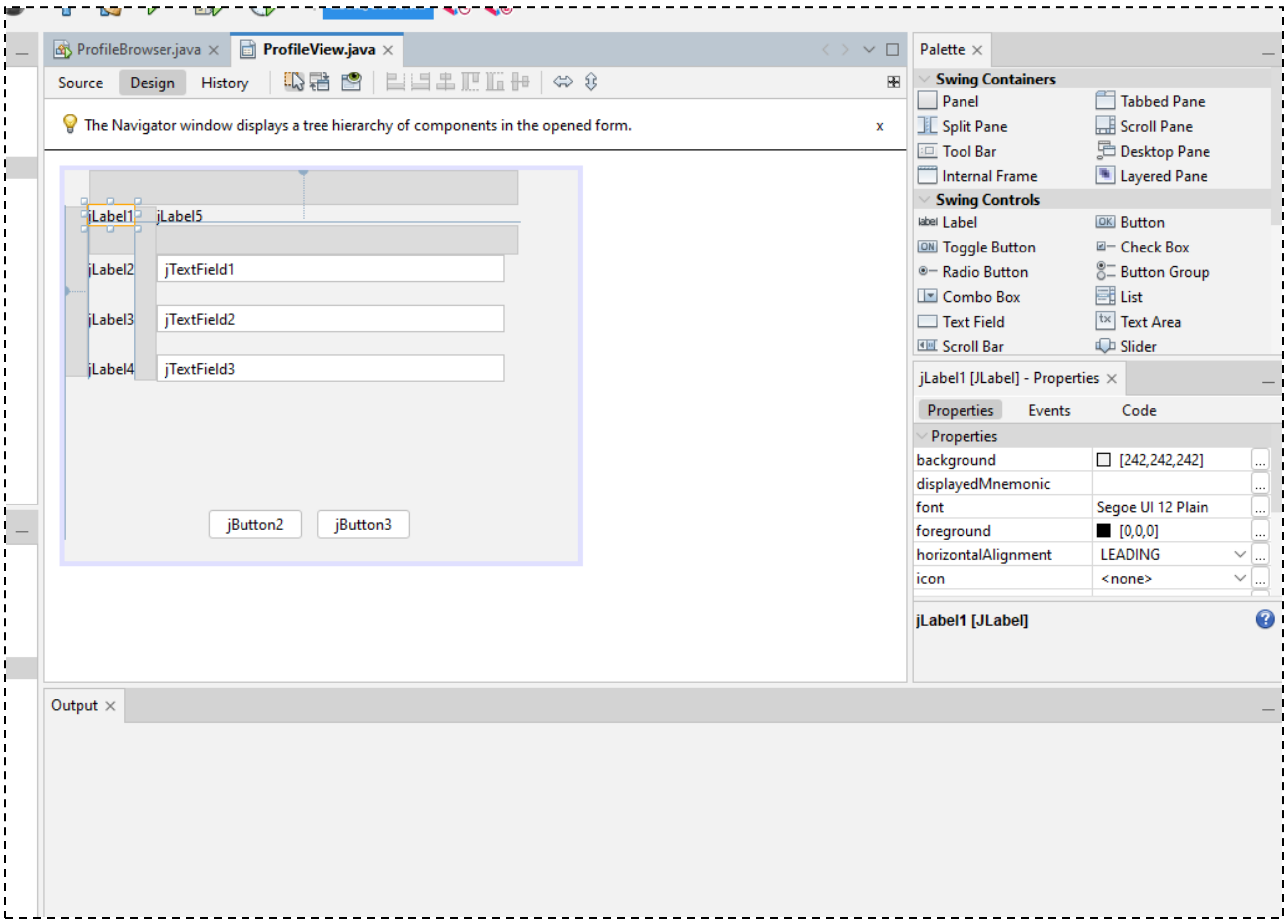
If you notice, every time you add an object inside your JFrame form, they will be listed inside the Navigator view. It is very helpful to keep track of your object's Identifiers.

Add the objects and design the UI as seen below:


Use the context menu to access available useful actions for the selected components.

After DESIGNING the layout, we need to change the VARIABLE NAMES (IDENTIFIERS) and TEXTS of the objects you just added.

To change the text of an object, just click it once to view its properties panel as seen below:


The Navigator window displays a tree hierarchy of components in the opened form.

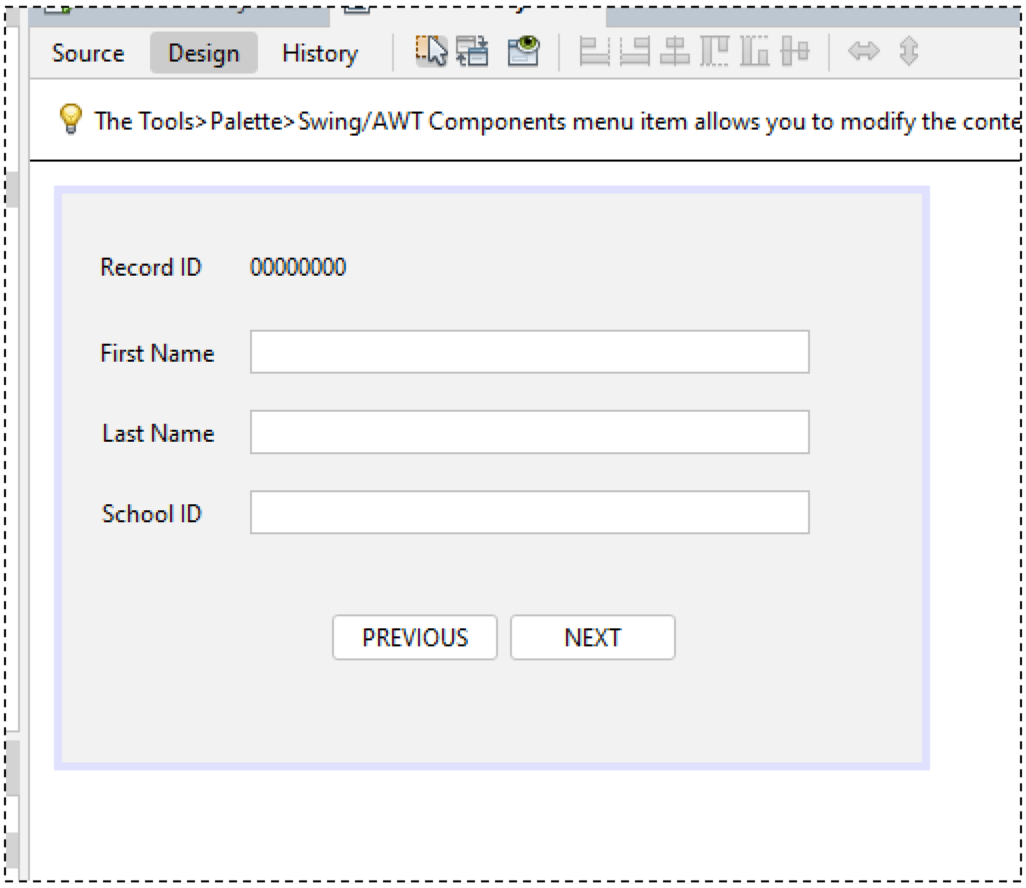
jLabel1 [JLabel] - Properties

background	[242,242,242]
displayedMnemonic	
font	Segoe UI 12 Plain
foreground	[0,0,0]
horizontalAlignment	LEADING
icon	<none>

jLabel1 [JLabel]

I clicked on the JLabel1 object and its properties panel appears on the right side of the screen. In that panel, you can see three (3) different tabs: Properties, Events and Code. By default, Properties is selected. It is on that tab you can find the ‘Text’ option to change the label of our object. Just scroll down and look for it.

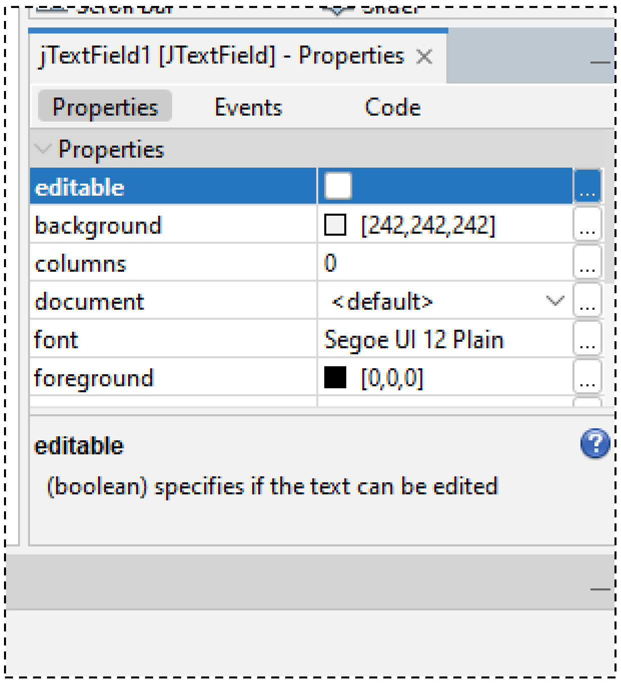
by default, the text of the object we selected is jLabel1 and it is what you can see in the text options under the properties. Just change the label to the necessary text you need. In our case, it's ID. You can do the same with other objects too such as the jLabels for the First Name, Last Name and School ID. For the jLabel next to the Record ID, change its text to any data, let's say, the default format of our record ID. For Text Fields, just leave the text with blanks. Change the texts of the buttons as well. My layout below:


Source
Design
History
The Tools>Palette>Swing/AWT Components menu item allows you to modify the conte
Record ID
00000000
First Name
Last Name
School ID
PREVIOUS
NEXT

Feel free to adjust and re-layout the alignment of the objects as desired.

By default, the text fields are active and the user can input data in it. For our activity project, we don't want to user to interact with the text field other than viewing the data coming from the database. We want the user only to click next or previous button to browse on the data inside the database.

Therefore, let's set the editable option of our textfields to off by clicking on each of the text field and look for the editable check mark. Just uncheck it as seen below:



editable	☐
background	☐ [242,242,242]
columns	0
document	<default>
font	Segoe UI 12 Plain
foreground	■ [0,0,0]

editable
(boolean) specifies if the text can be edited

Apply the same setting to all text fields

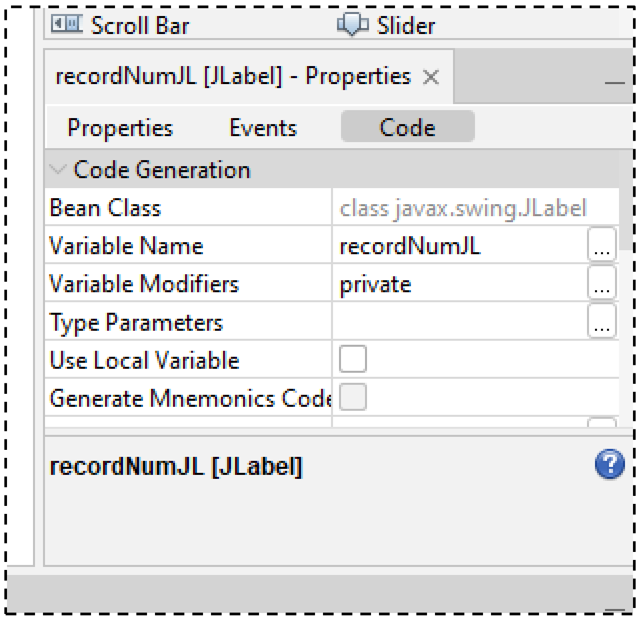
After the layout, we need to set the identifiers (variable names) of our objects so we can start coding.

In our project, we have added plenty of objects but there are three types of them only which are: jLabel, Text Field and Button.

For the objects we want to interact with our code, we need to change their identifiers to something descriptive and easier for us to remember.

In our case, we will need to change the identifier of the jlabel besides Record ID because we want its data to be dynamic in accordance to what's found in the database. The same for the three (3) text fields below for First Name, Last Name and School ID.

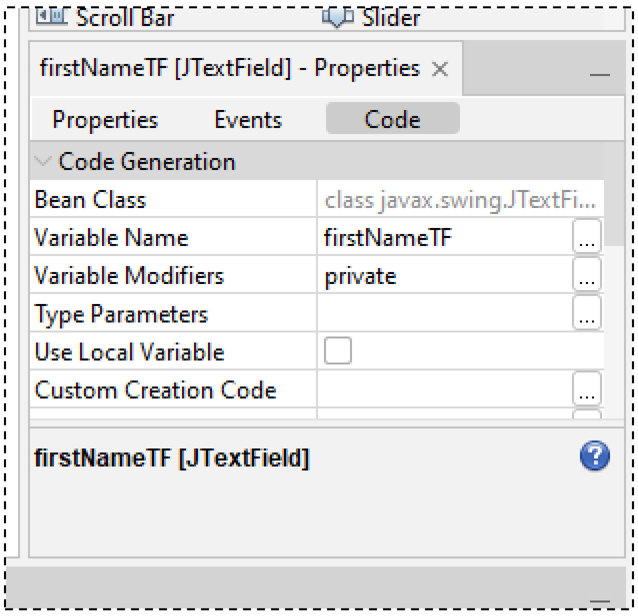
To change the identifiers (variable names) of the object, just click on the object and click on the 'Code' tab under Properties' panel as seen below:



Bean Class	class javax.swing.JLabel
Variable Name	recordNumJL
Variable Modifiers	private
Type Parameters	
Use Local Variable	☐
Generate Mnemonics Code	☐

recordNumJL [JLabel]

In my case, I changed the Variable Name my jLabel (next to record ID) to **recordNumJL**.



Bean Class	class javax.swing.JTextFi...
Variable Name	firstNameTF
Variable Modifiers	private
Type Parameters	
Use Local Variable	☐
Custom Creation Code	

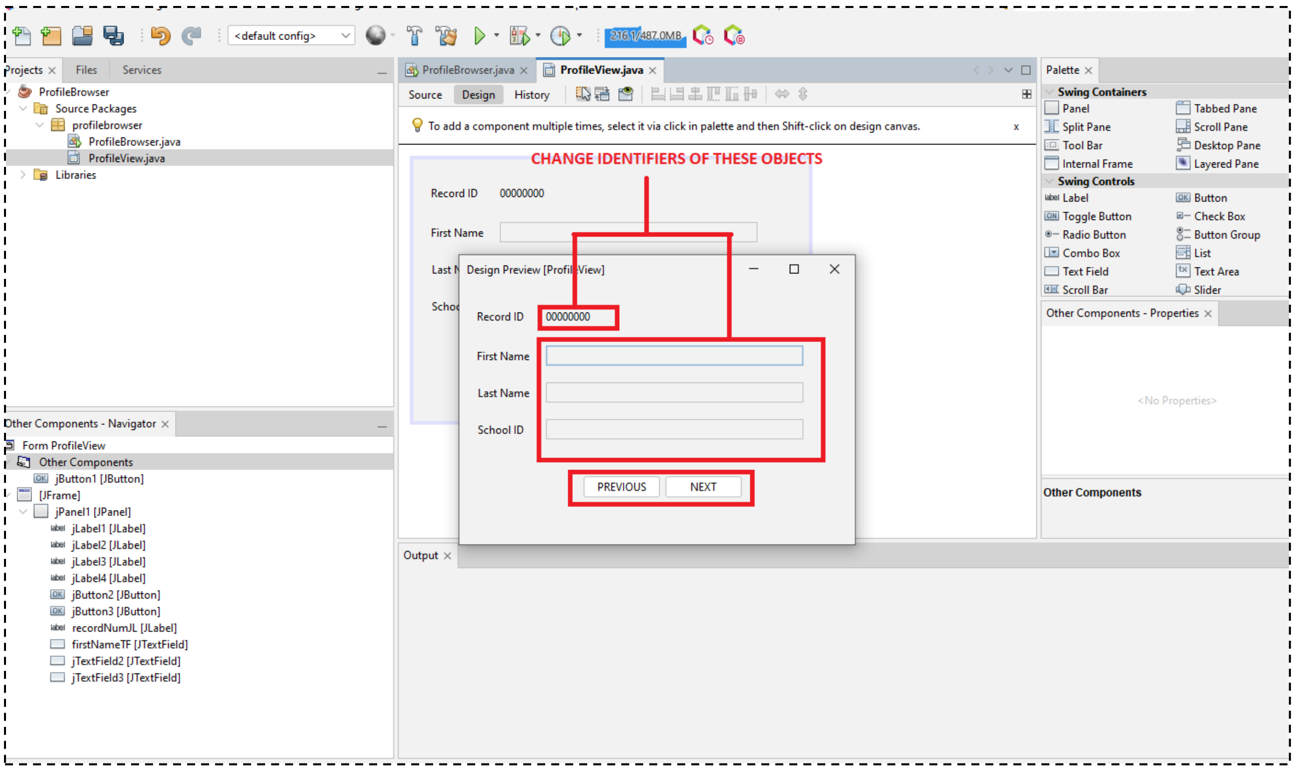
firstNameTF [JTextField]

I changed the variable for the First Name Text Field to **firstNameTF**.

Do the same with your other objects that need dynamic data changing. In my case, here are the identifiers I used:

- jLabel5 -> recordNumJL
- jTextField1 -> firstNameTF
- jTextField2 -> lastNameTF
- jTextField3 -> school_numTF
- jButton1 -> previousButton
- jButton2 -> nextButton

In case you're confused, I changed the identifiers of the objects I boxed below:

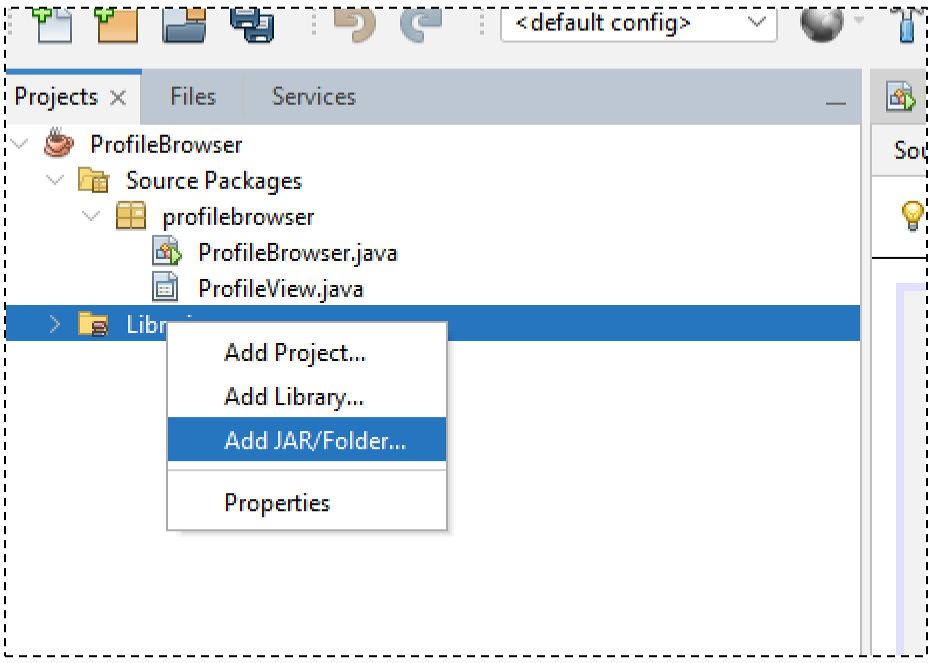

CHANGE IDENTIFIERS OF THESE OBJECTS
Design Preview [ProfileView]
Record ID
00000000
First Name
Last Name
School ID
PREVIOUS
NEXT

Other Components - Navigator
Form ProfileView
Other Components
jButton1 [JButton]
[JFrame]
jPanel1 [JPanel]
jLabel1 [JLabel]
jLabel2 [JLabel]
jLabel3 [JLabel]
jLabel4 [JLabel]
jButton2 [JButton]
jButton3 [JButton]
recordNumJL [JLabel]
firstNameTF [JTextField]
jTextField2 [JTextField]
jTextField3 [JTextField]

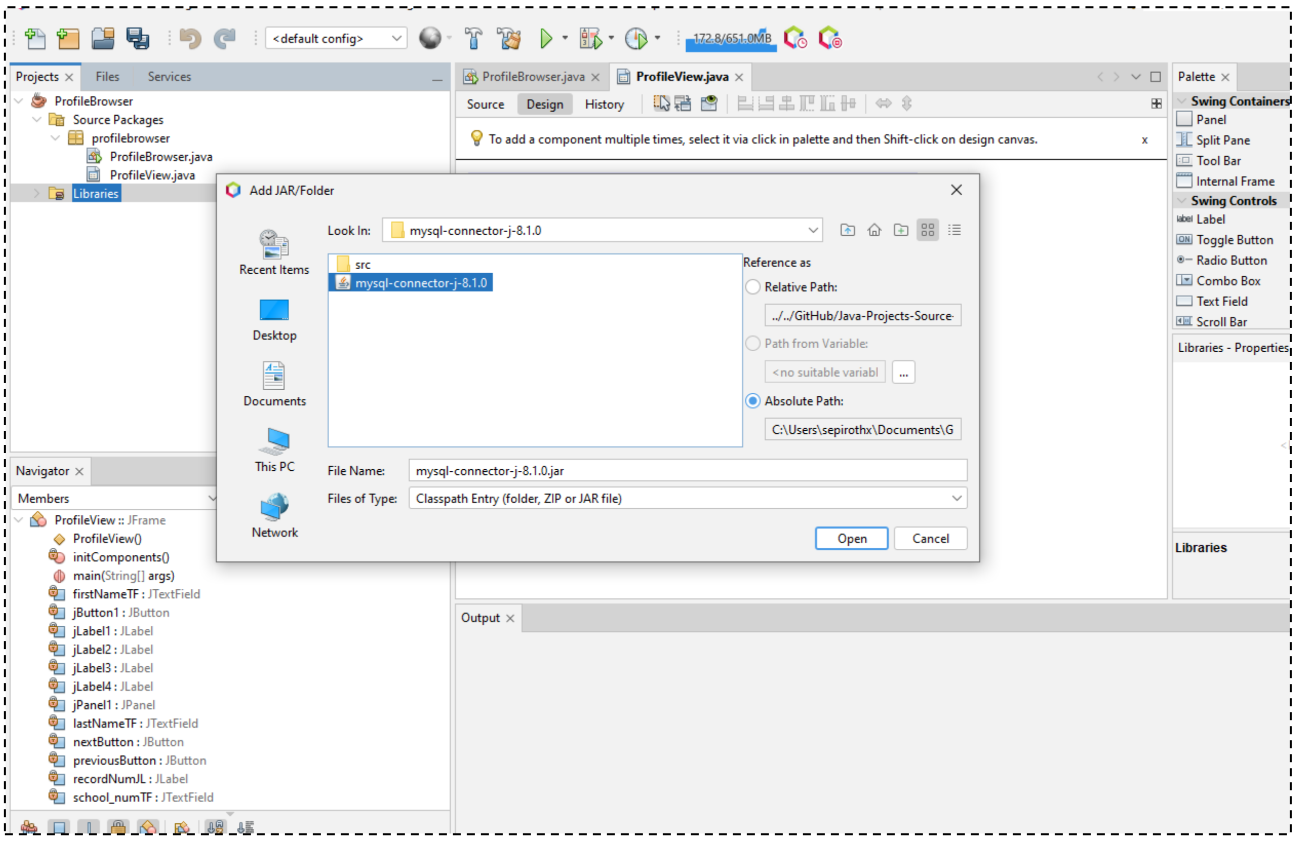
Since we now have our UI and objects. Let's get into coding!!!

ADDING JDBC DRIVER CONNECTOR / J

To start setting up the database connection between our java application and our mysql database running on phpMyAdmin via XAMPP, we need to add the JDBC DRIVER CONNECTOR/J file first to our project. Head over to Libraries under Projects Panel, right click on It and click Add **JAR/Folder…**



Locate the my-sql-connector-j jar file that you just downloaded. Click Open to add it inside your project.



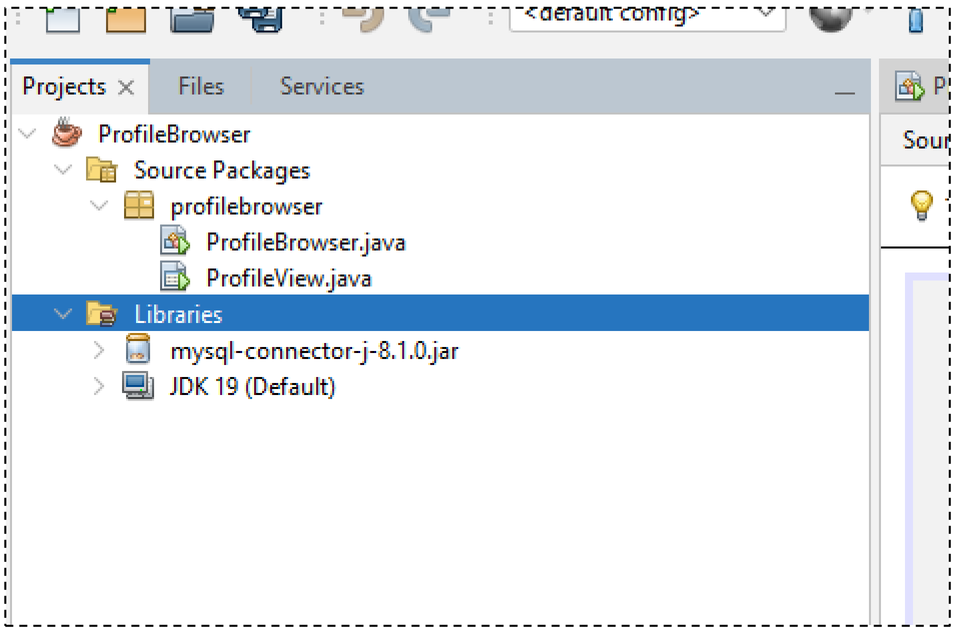
Look In: mysql-connector-j-8.1.0

Reference as
Relative Path: ../../GitHub/Java-Projects-Source
Path from Variable: <no suitable variabl
Absolute Path: C:\Users\sepirothx\Documents\G

File Name: mysql-connector-j-8.1.0.jar

Files of Type: Classpath Entry (folder, ZIP or JAR file)

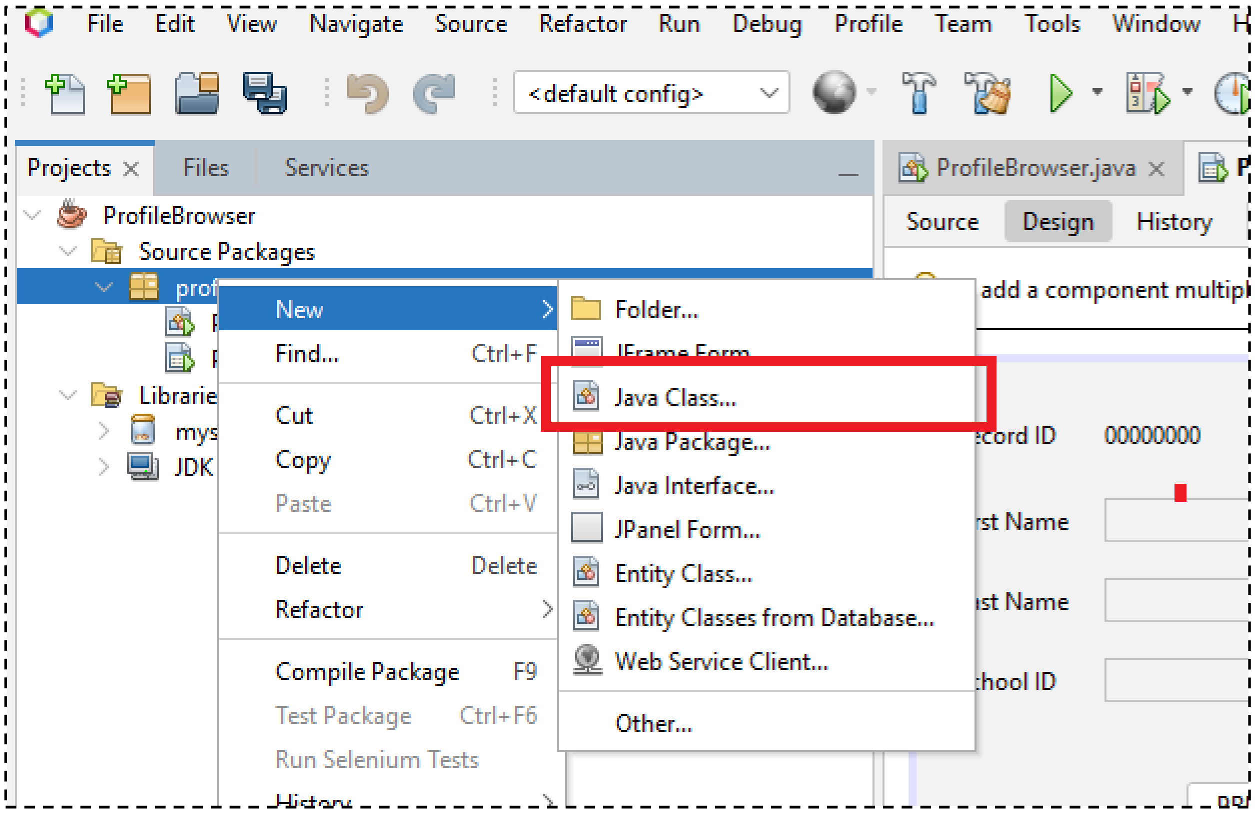
To confirm the presence of your mysql connector, just collapse the libraries inside your Projects Panel as seen below:



ProfileBrowser
Source Packages
profilebrowser
ProfileBrowser.java
ProfileView.java
Libraries
mysql-connector-j-8.1.0.jar
JDK 19 (Default)

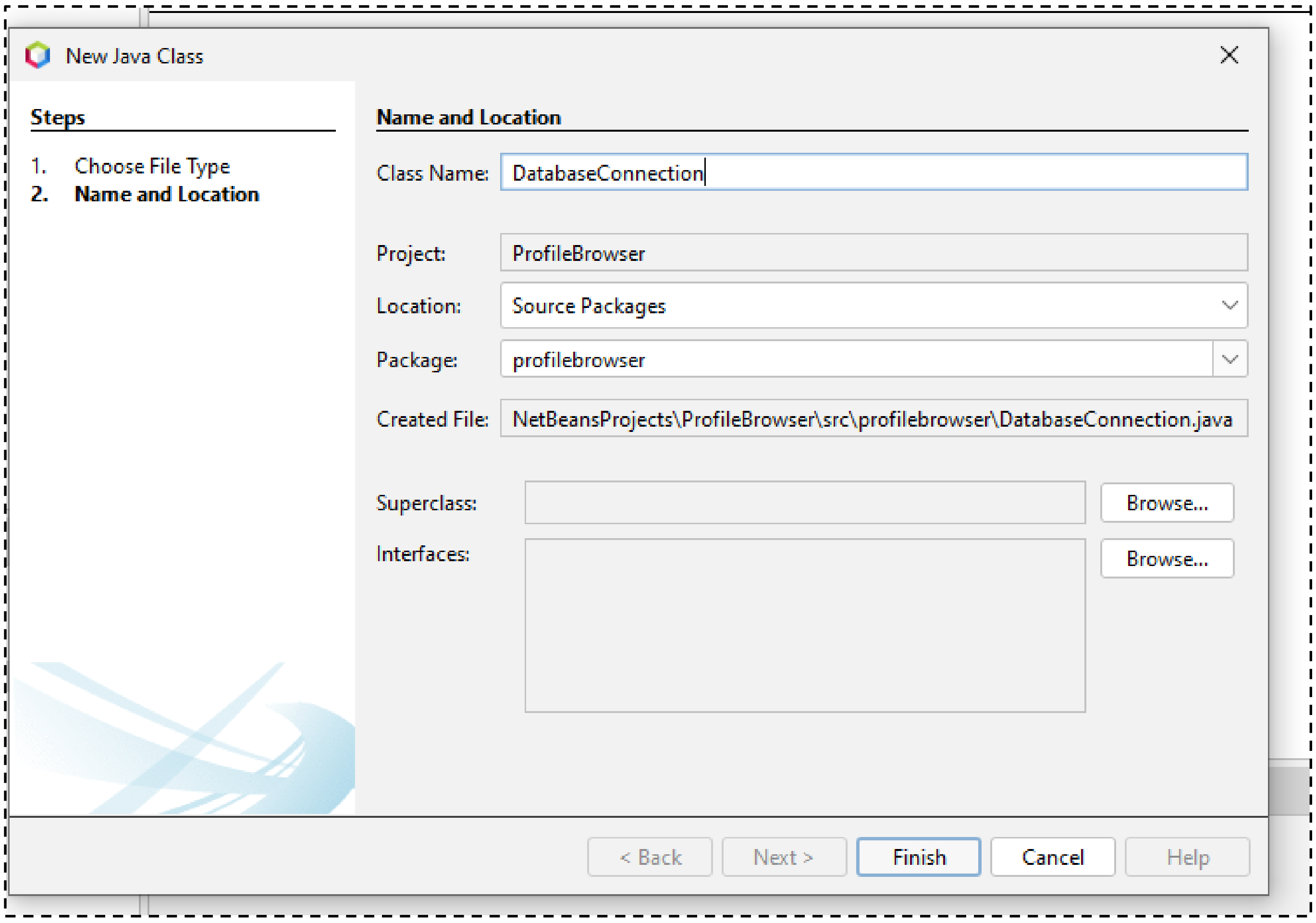
WRITING DATABASE CONNECTION CLASS

After adding the jar file, let's create our DatabaseConnection Class by right clicking on the package of your project and create a new class as seen below:



WRITING THE DATABASE CONNECTION CLASS

Give it a name **'DatabaseConnection'** however, feel free to use any name but it must be **'Descriptive'**. No spaces allowed! Click Finish right away.


New Java Class

Steps
1. Choose File Type
2. **Name and Location**

Name and Location

Class Name: DatabaseConnection

Project: ProfileBrowser

Location: Source Packages

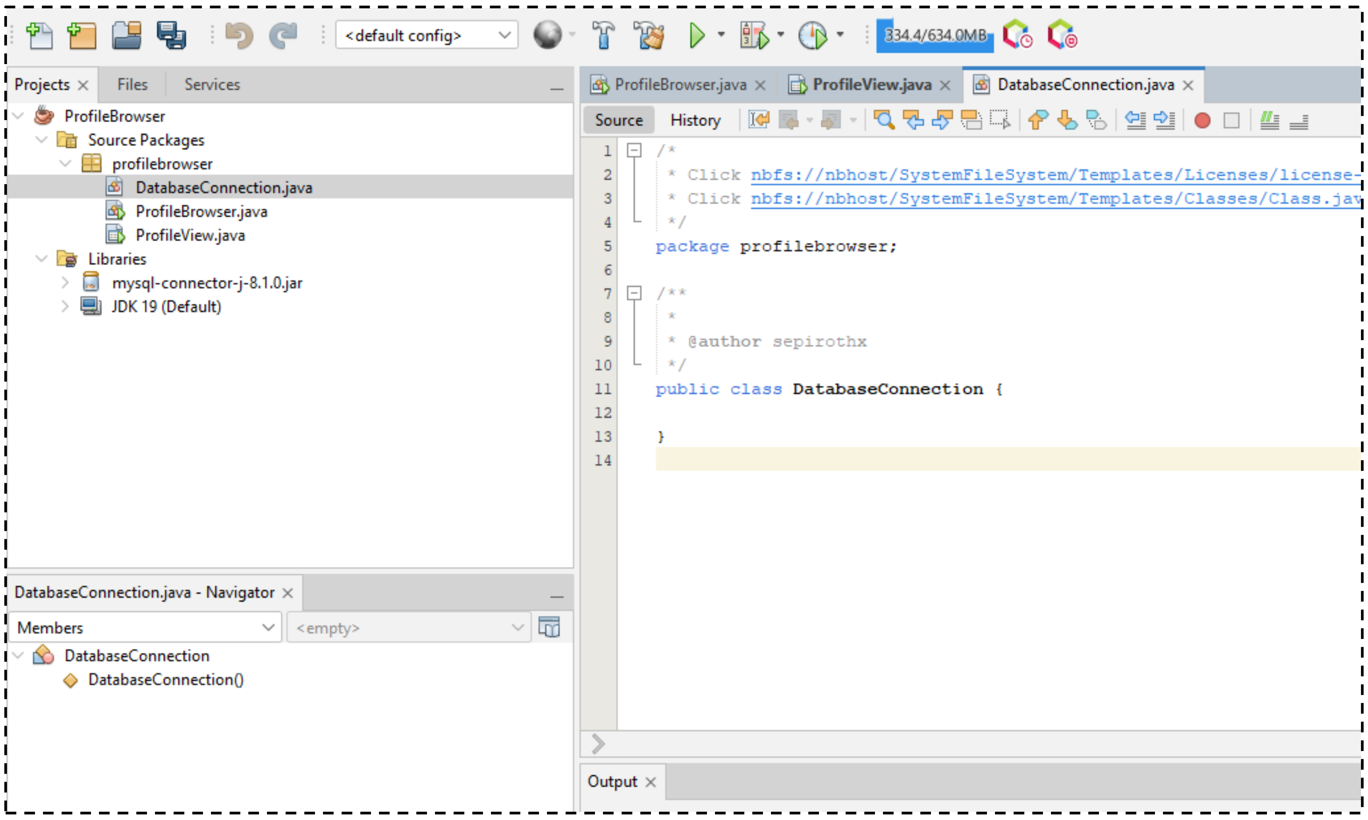
Package: profilebrowser

Created File: NetBeansProjects\ProfileBrowser\src\profilebrowser\DatabaseConnection.java

Superclass:

Interfaces:

In this part, you'll see a new file is added in your project and it is freshly opened inside your code editor.



ProfileBrowser
- Source Packages
  - profilebrowser
    - DatabaseConnection.java
    - ProfileBrowser.java
    - ProfileView.java
- Libraries
  - mysql-connector-j-8.1.0.jar
  - JDK 19 (Default)

ProfileBrowser.java | ProfileView.java | DatabaseConnection.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.jav
 */
package profilebrowser;

/**
 *
 * @author sepirothx
 */
public class DatabaseConnection {

}
```

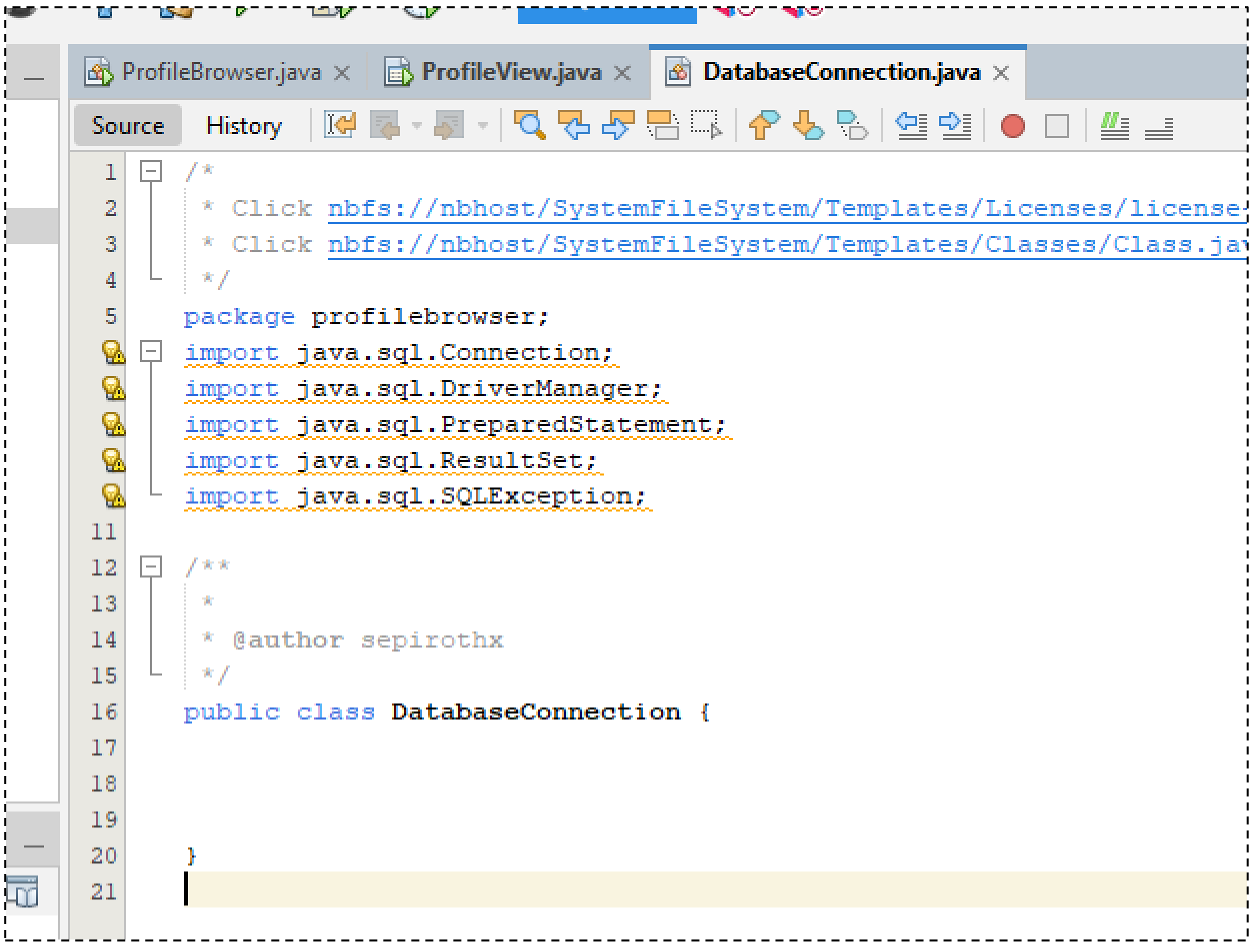
DatabaseConnection.java - Navigator

Members | <empty>

- DatabaseConnection
  - DatabaseConnection()

Now, let's import important packages needed to establish database connection. Import the following packages:

- import java.sql.Connection;
- import java.sql.DriverManager;
- import java.sql.PreparedStatement;
- import java.sql.ResultSet;
- import java.sql.SQLException;

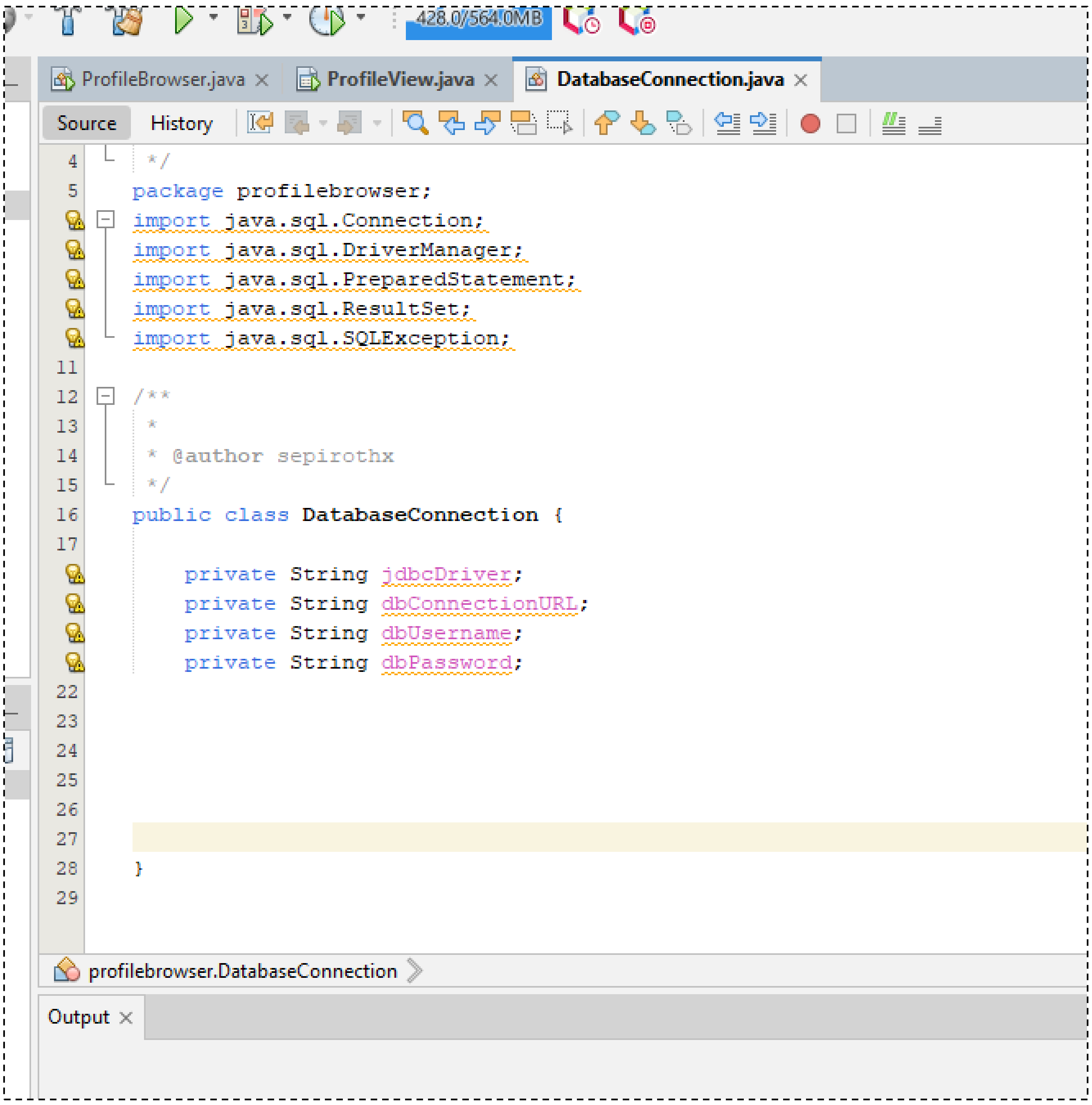


```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.ja
 */
package profilebrowser;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 *
 * @author sepirothx
 */
public class DatabaseConnection {



}
```

The next thing we need to do is to create private fields for the following data:

- JDBC Driver
- Database Connection URL
- Database Username
- Database Password

Set each field as 'private' because we want it to live inside DatabaseConnection class only and not be available now modifiable outside this class.

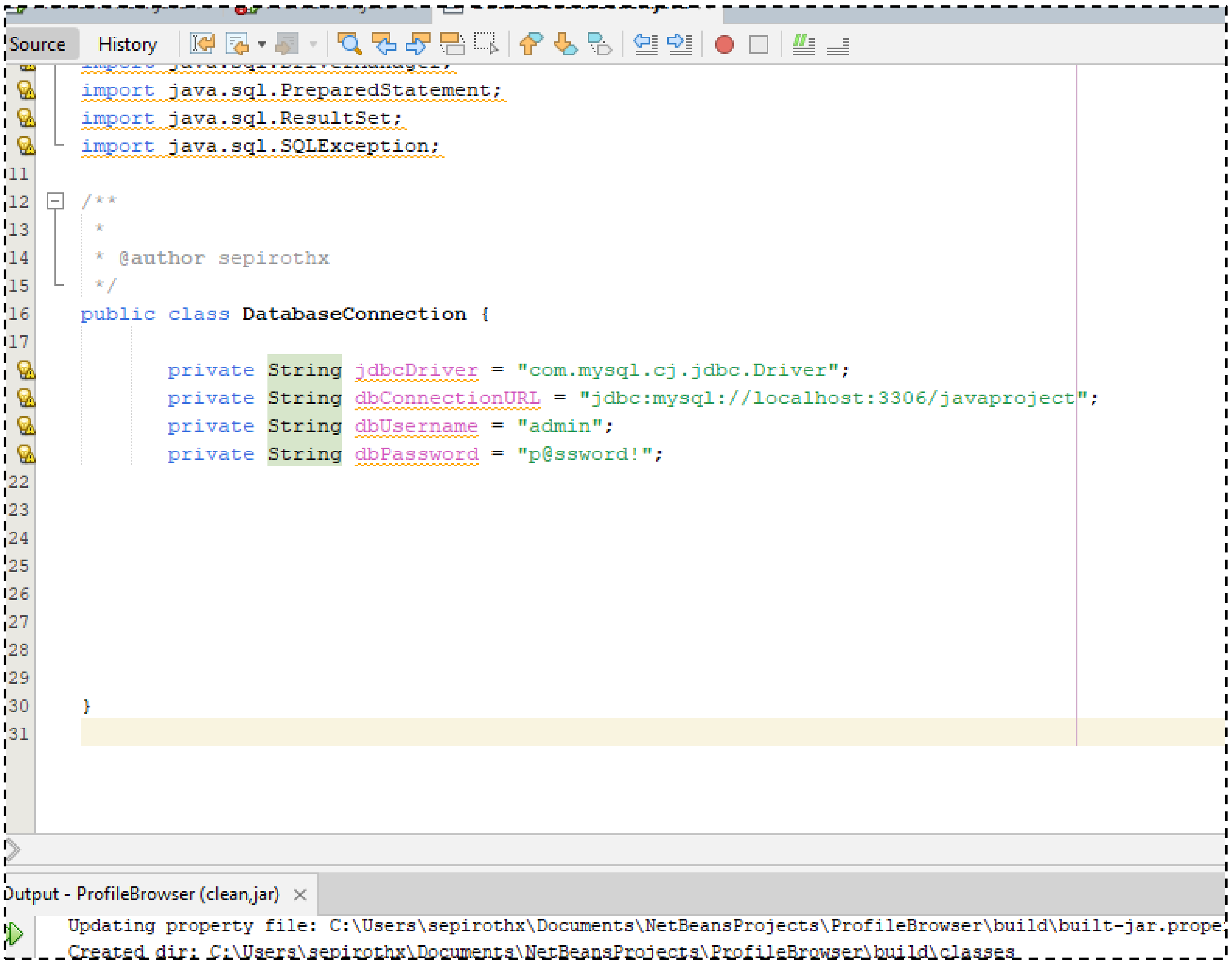


```
 */
package profilebrowser;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 *
 * @author sepirothx
 */
public class DatabaseConnection {

    private String jdbcDriver;
    private String dbConnectionURL;
    private String dbUsername;
    private String dbPassword;






}
```

Add the following source code inside your `**DatabaseConnection**` class' body as seen from above code.

```
    private String jdbcDriver;

    private String dbConnectionURL;

    private String dbUsername;

    private String dbPassword;
```

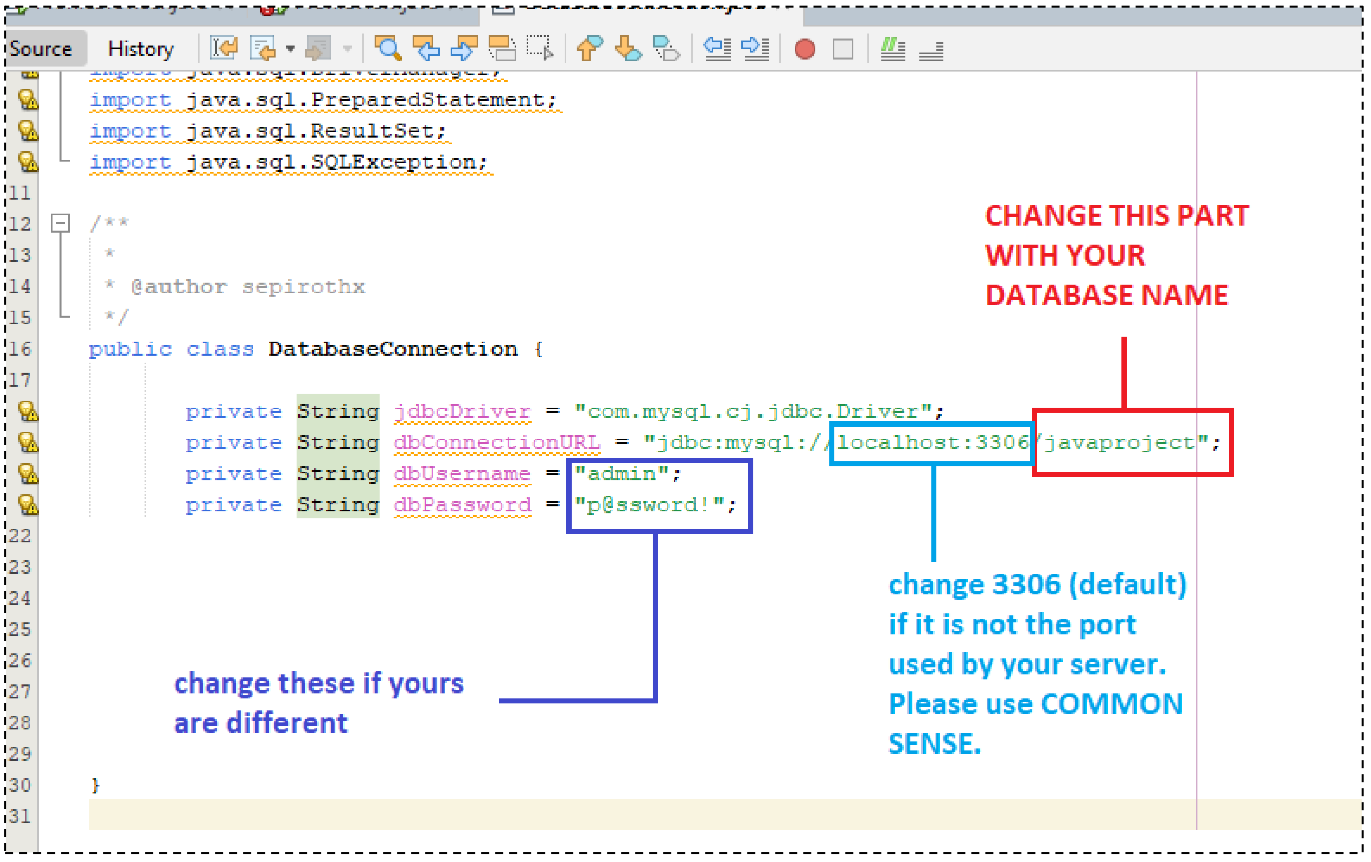
Next is to initialize the credentials for the private fields. The data in these fields will be accessed by our object's getters and will be used once we establish connection to the database. Write the code as follows:



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

/**
 *
 * @author sepirothx
 */
public class DatabaseConnection {

        private String jdbcDriver = "com.mysql.cj.jdbc.Driver";
        private String dbConnectionURL = "jdbc:mysql://localhost:3306/javaproject";
        private String dbUsername = "admin";
        private String dbPassword = "p@ssword!";









}
```

```
Output - ProfileBrowser (clean,jar)
Updating property file: C:\Users\sepirothx\Documents\NetBeansProjects\ProfileBrowser\build\built-jar.prope
Created dir: C:\Users\sepirothx\Documents\NetBeansProjects\ProfileBrowser\build\classes
```

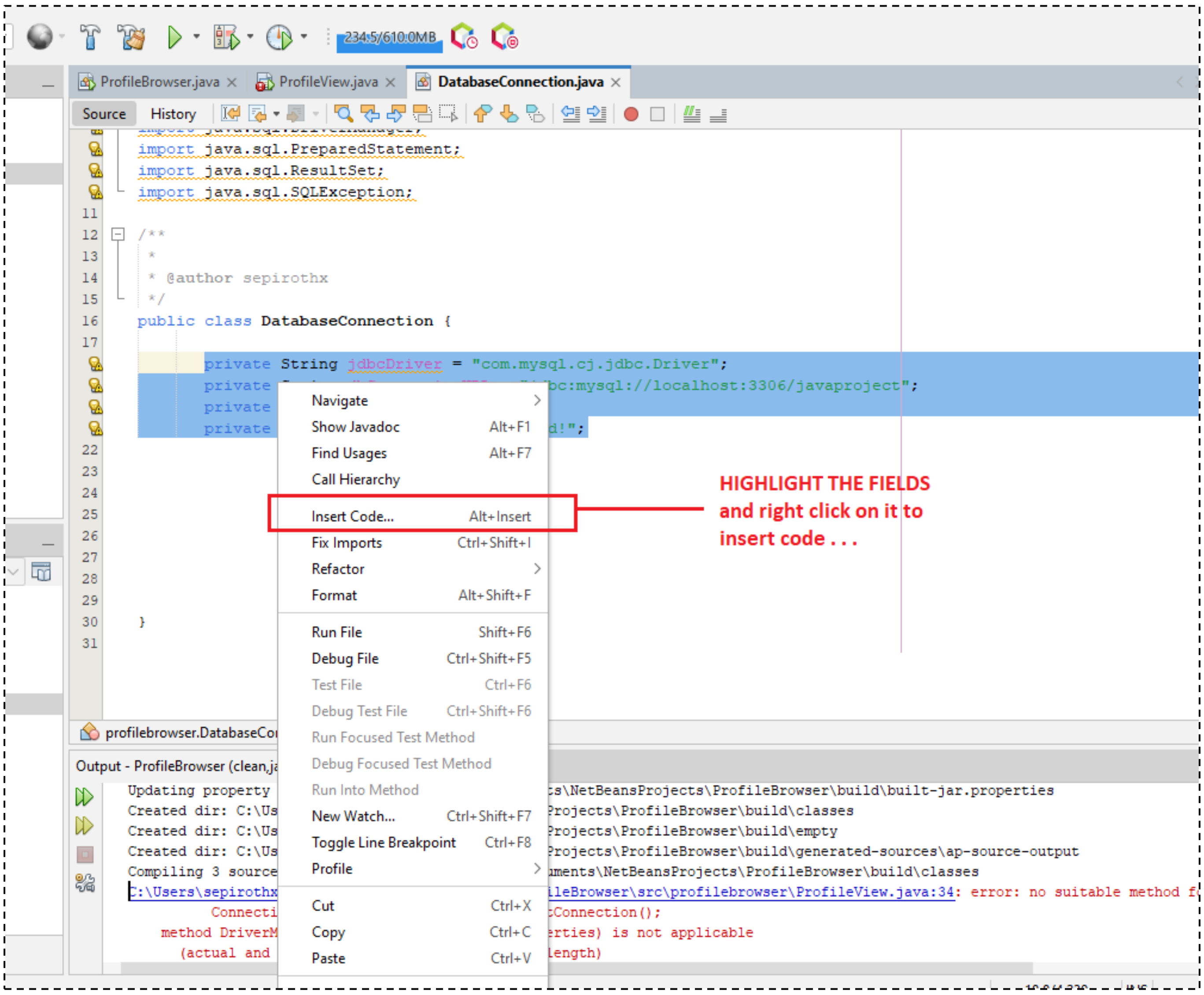
You can follow everything aside from the ***dbConnectionURL***, ***dbUsername*** and ***dbPassword***.



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

/**
 *
 * @author sepirothx
 */
public class DatabaseConnection {

        private String jdbcDriver = "com.mysql.cj.jdbc.Driver";
        private String dbConnectionURL = "jdbc:mysql://localhost:3306/javaproject";
        private String dbUsername = "admin";
        private String dbPassword = "p@ssword!";









}
```

CHANGE THIS PART WITH YOUR DATABASE NAME
change 3306 (default) if it is not the port used by your server. Please use COMMON SENSE.
change these if yours are different

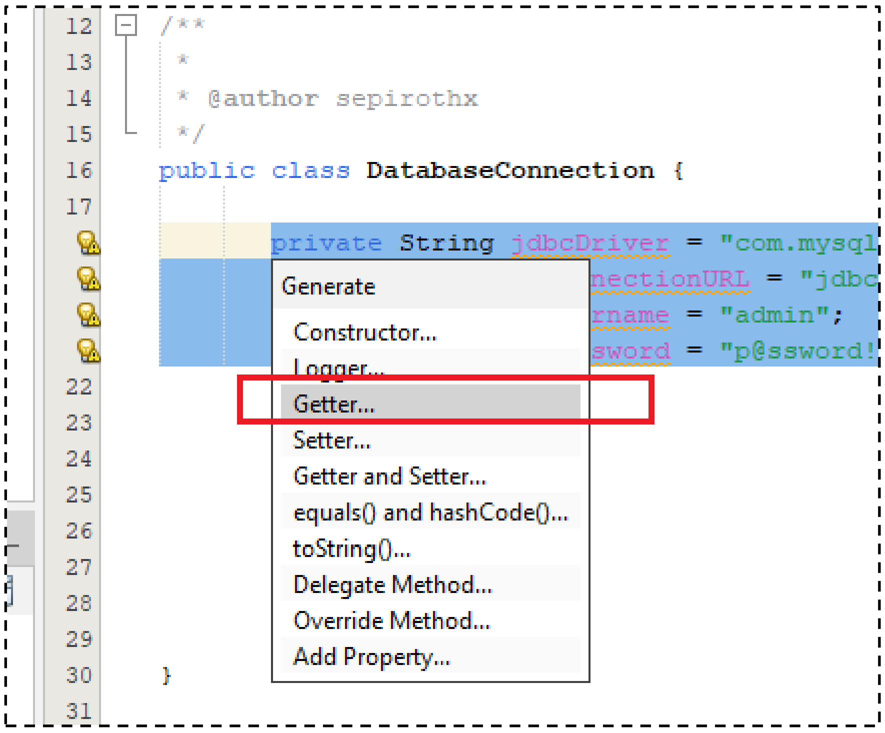
For the above fields, we need to encapsulate it for security purposes. In order for us to access the data assigned to these fields (variables), we need to use the so-called **'public getters'** methods. These methods we need to add so we can pull out the data and establish a database connection in order classes. We can write these methods manually however the fastest way is to use our IDE's built-in insert code feature as follows:



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

/**
 *
 * @author sepirothx
 */
public class DatabaseConnection {

    private String jdbcDriver = "com.mysql.cj.jdbc.Driver";
    private                              bc:mysql://localhost:3306/javaproject";
    private
    private                              d!";






}
```

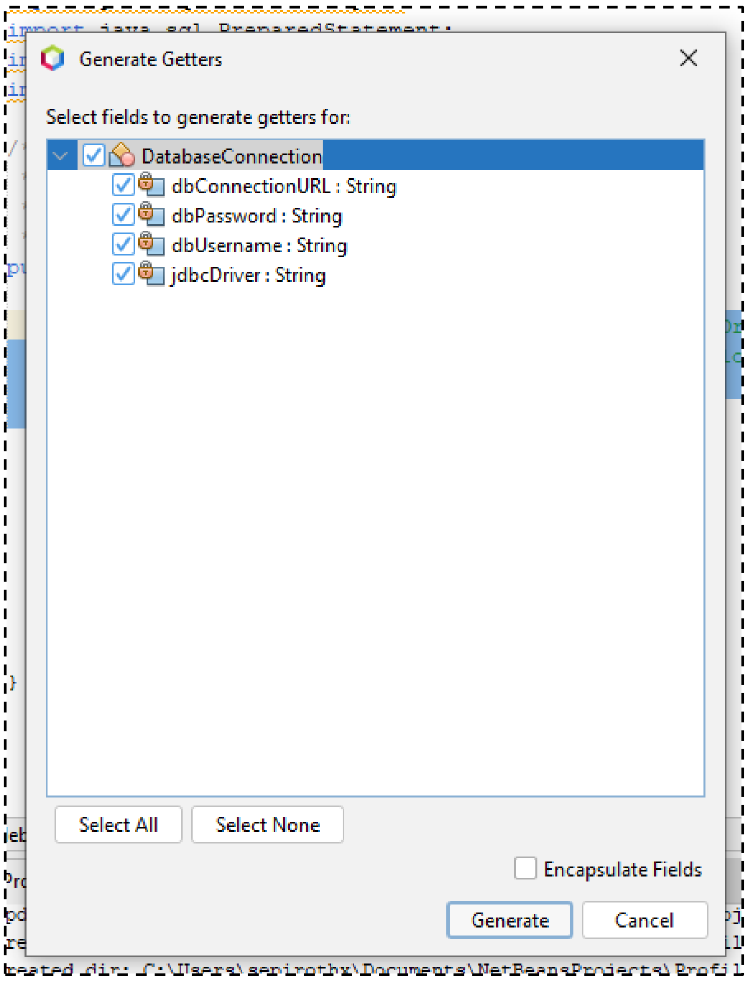
```
Updating property                    :s\NetBeansProjects\ProfileBrowser\build\built-jar.properties
Created dir: C:\Us                   Projects\ProfileBrowser\build\classes
Created dir: C:\Us                   Projects\ProfileBrowser\build\empty
Created dir: C:\Us                   Projects\ProfileBrowser\build\generated-sources\ap-source-output
Compiling 3 source                   uments\NetBeansProjects\ProfileBrowser\build\classes
C:\Users\sepirothx                   ileBrowser\src\profilebrowser\ProfileView.java:34: error: no suitable method f
          Connecti                   Connection();
    method DriverM                   erties) is not applicable
      (actual and                    length)
```

Highlight the declared fields (variables) and right click to pop-out the context menu. Click on **Insert Code . . .**

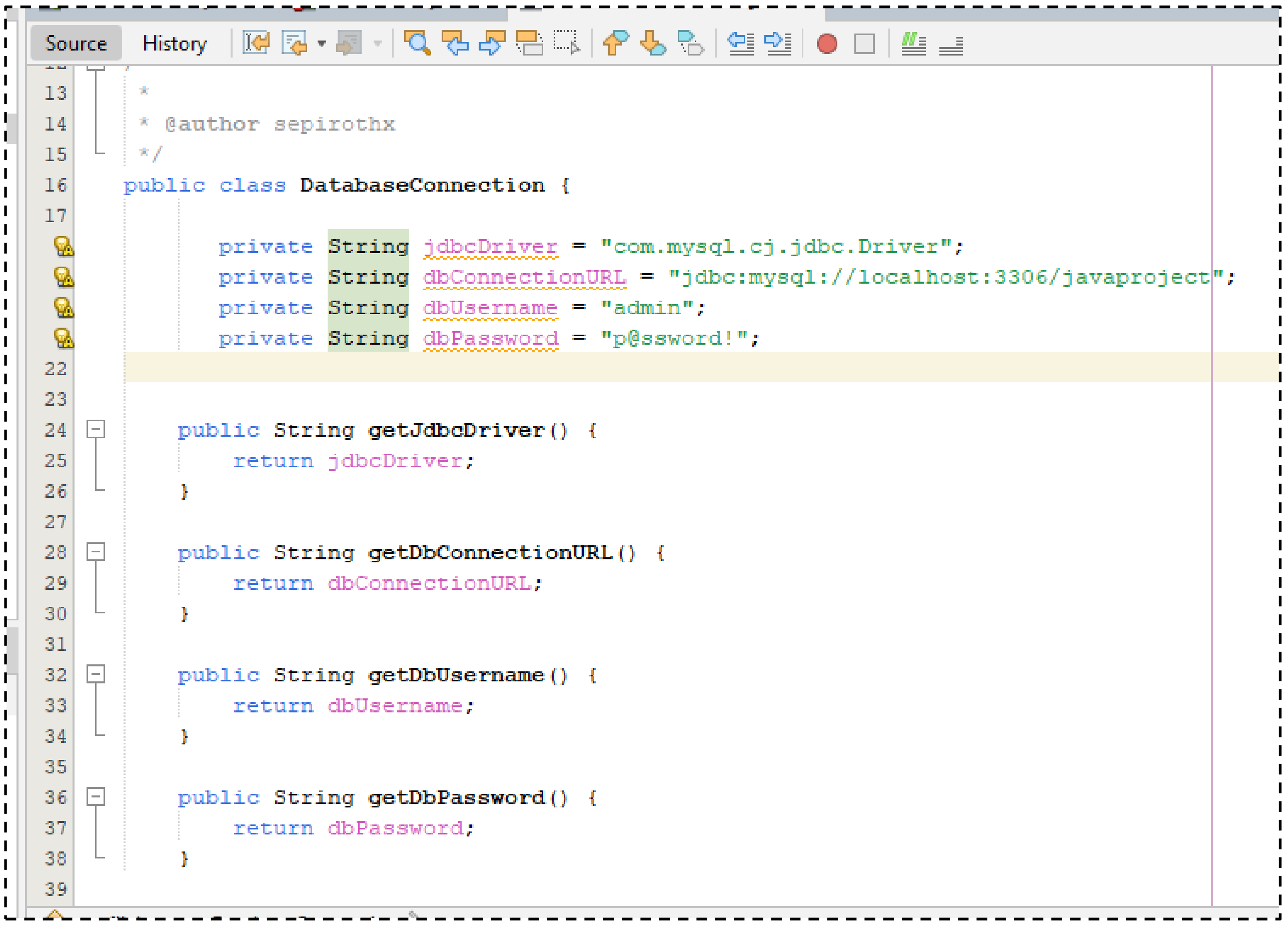


```
/**
 *
 * @author sepirothx
 */
public class DatabaseConnection {

    private String jdbcDriver = "com.mysql
                     nectionURL = "jdbc
                     rname = "admin";
                     sword = "p@ssword!

}
```

This time, let's auto-generate the Getters for the fields we declared. Click on Getter . . .



Click on **Generate** to automatically insert needed codes.

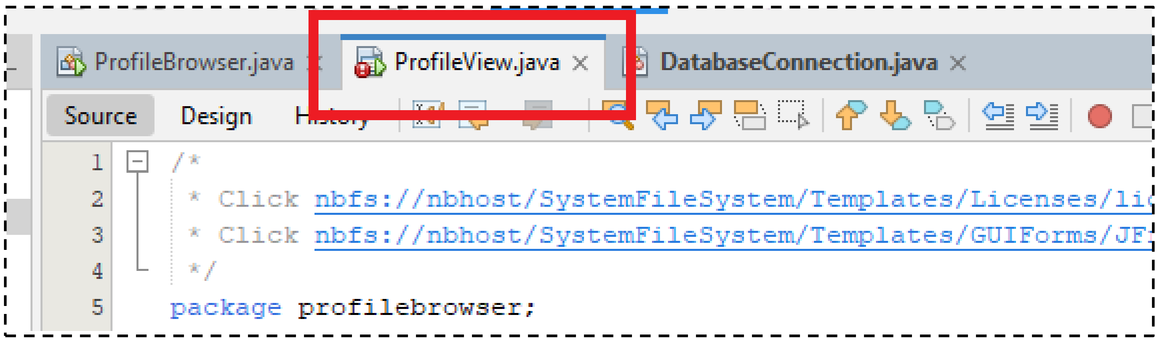


```
 *
 * @author sepirothx
 */
public class DatabaseConnection {

        private String jdbcDriver = "com.mysql.cj.jdbc.Driver";
        private String dbConnectionURL = "jdbc:mysql://localhost:3306/javaproject";
        private String dbUsername = "admin";
        private String dbPassword = "p@ssword!";


    public String getJdbcDriver() {
        return jdbcDriver;
    }

    public String getDbConnectionURL() {
        return dbConnectionURL;
    }

    public String getDbUsername() {
        return dbUsername;
    }

    public String getDbPassword() {
        return dbPassword;
    }

```

After auto-generating the Getters, will code will look something like what you can see above. However, feel free to cut the inserted codes and move them below your declared fields if in-case the generated getters were inserted above your declared private fields.

WRITING THE SOURCE CODE FOR THE PROFILE VIEW

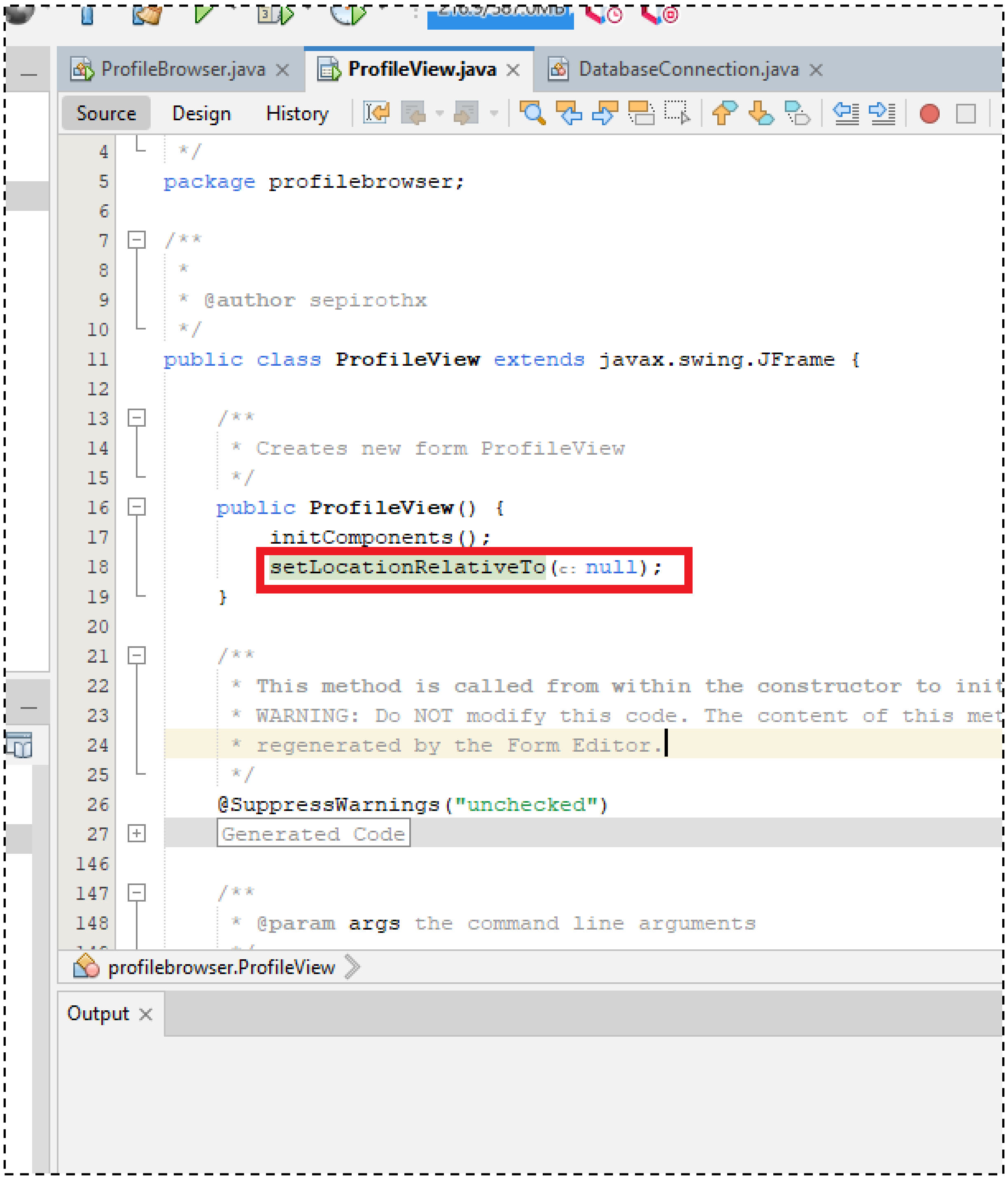
This time, let's work on our ProfileView.java class. Head over on its tab to start editing its source code.



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/li
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JF
 */
package profilebrowser;
```

In ProfileView's source code, you can add this statement inside its constructor. This is to set the entire UI to display in the middle of the screen once this class starts running.

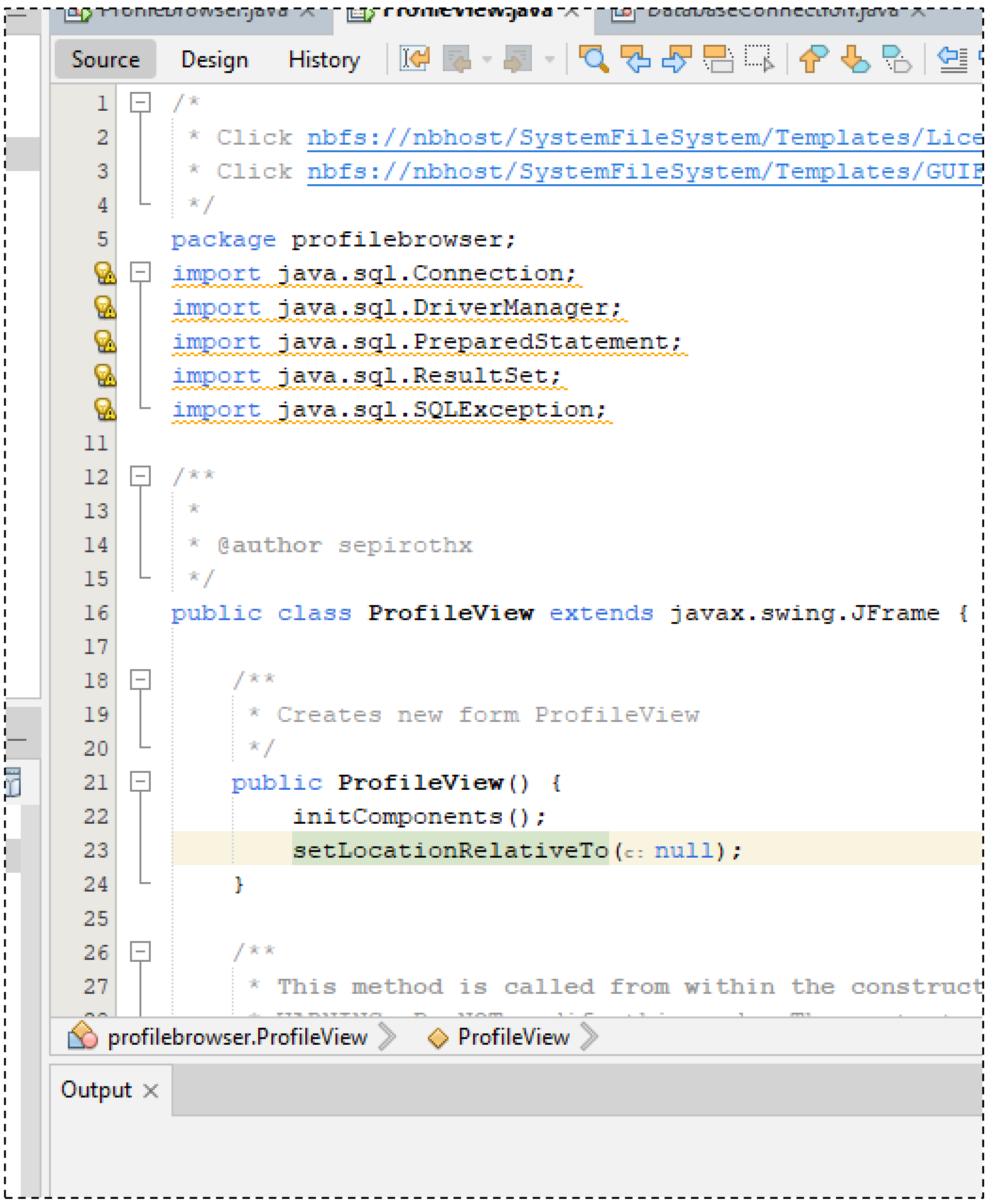
```
setLocationRelativeTo(null);
```



```
 */
package profilebrowser;

/**
 *
 * @author sepirothx
 */
public class ProfileView extends javax.swing.JFrame {

    /**
     * Creates new form ProfileView
     */
    public ProfileView() {
        initComponents();
        setLocationRelativeTo(c: null);
    }

    /**
     * This method is called from within the constructor to init
     * WARNING: Do NOT modify this code. The content of this met
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    /**
     * @param args the command line arguments
```

Since we want our application to pull out the data right at the start of the UI display, therefore, we need to write the query of our database and the object of our DatabaseConnection inside the constructor of our ProfileView.

Import the necessary packages as seen below. These same packages were the once we also imported inside DatabaseConnection.java class.



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Lice
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIF
 */
package profilebrowser;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 *
 * @author sepirothx
 */
public class ProfileView extends javax.swing.JFrame {

    /**
     * Creates new form ProfileView
     */
    public ProfileView() {
        initComponents();
        setLocationRelativeTo(c: null);
    }

    /**
     * This method is called from within the construct
```

Next part is to do `try and catch` and pass our database query and connection within the try block to pull-out the data coming from the database. You can insert this code for the entire **try** block:

```
try {
     // this sql query will pull out all database records
        String query = "SELECT * FROM users"; //SQL Syntax
     //create object for the DatabaseConnection
        DatabaseConnection dbc = new DatabaseConnection();

     //Declare variables to capture database credentials
        String jdbcDriver = dbc.getJdbcDriver();
        String dbConnectionURL = dbc.getDbConnectionURL();
        String dbUsername = dbc.getDbUsername();
        String dbPassword = dbc.getDbPassword();
} catch (ClassNotFoundException | SQLException e  ) {
        e.printStackTrace();
  }
```

Let me explain the above code. The statements inside the try block declares the syntax of our SQL Query, which is what we need to pull out the data stored in our database.

```
DatabaseConnection dbc = new DatabaseConnection();
```

The above statement creates the `dbc` object (instance) which is we need to pull out the private fields (variables) declared in a separate class with an identifier DatabaseConnection.

```
String jdbcDriver = dbc.getJdbcDriver();
String dbConnectionURL = dbc.getDbConnectionURL();
String dbUsername = dbc.getDbUsername();
String dbPassword = dbc.getDbPassword();
```

The above statements prepare the variables we need to pull out the credentials stored in the fields of the DatabaseConnection. We invoked the **`dbc`** object and used the available public **getters** we auto-matically generated earlier for each field we need**.**

The next statements we need to insert inside our try block are the following:

```
Class.forName(jdbcDriver);
Connection connection = DriverManager.getConnection(dbConnectionURL,
                dbUsername,dbPassword);
PreparedStatement statement = connection.prepareStatement(query);
ResultSet resultSet = statement.executeQuery();
```

Let me explain each statement.

```
Class.forName(jdbcDriver);
```

The above statement registers the jdbcDriver.

```
Connection connection = DriverManager.getConnection(dbConnectionURL,
                dbUsername,dbPassword);
```

The above statement creates our instance for the Connection class. I created the object and named it ***`connection`*** and initialized the data to it, the data coming from the ***DriverManager*** class using DriverManager's class method ***.getConnection()*** and passed inside the variables ***dbConnectionURL***, ***dbUsername*** and ***dbPassword*** as parameter arguments. The data held by these variables come from the private fields we declared in ***DatabaseConnection*** class.

```
PreparedStatement statement = connection.prepareStatement(query);
ResultSet resultSet = statement.executeQuery();
```

In the above statements, I used two pre-defined classes and declared objects of their type. The **`statement`** object is assigned for establishing a connection to the database for the given **`query`** while **`resultSet`** is assigned with the execution of our **`statement`** variable's query.

Next code we need to insert our try block is the condition to check if there's a result data coming from the query. We can do it by adding the following code as continuation of what we have written above, inside our try block:

```
if(resultSet.next()) {

 System.out.println("Connection established!");
 String record_id = resultSet.getString("id"); //id from sql column
 String firstName = resultSet.getString("firstname"); //firstname from sql column
 String lastName = resultSet.getString("lastname");
 String school_id = resultSet.getString("school_id");

 //below statements will display data coming from the database columns
 System.out.println("Record ID: " + record_id);
 System.out.println("First Name: " + firstName);
 System.out.println("Last Name: " + lastName);
 System.out.println("School ID: " + school_id);
}
```

The above code will help us check if there's data pulled out coming from the database, display the appropriate data for each column we need from the database.

Now, outside our ***if statement***, we can add the close statements for ***connection***, ***resultset*** and ***statement*** objects. It is necessary to close these if they're not in use anymore to avoid memory leaks and unnecessary usage of ram resources.

```
connection.close();
resultSet.close();
statement.close();
```

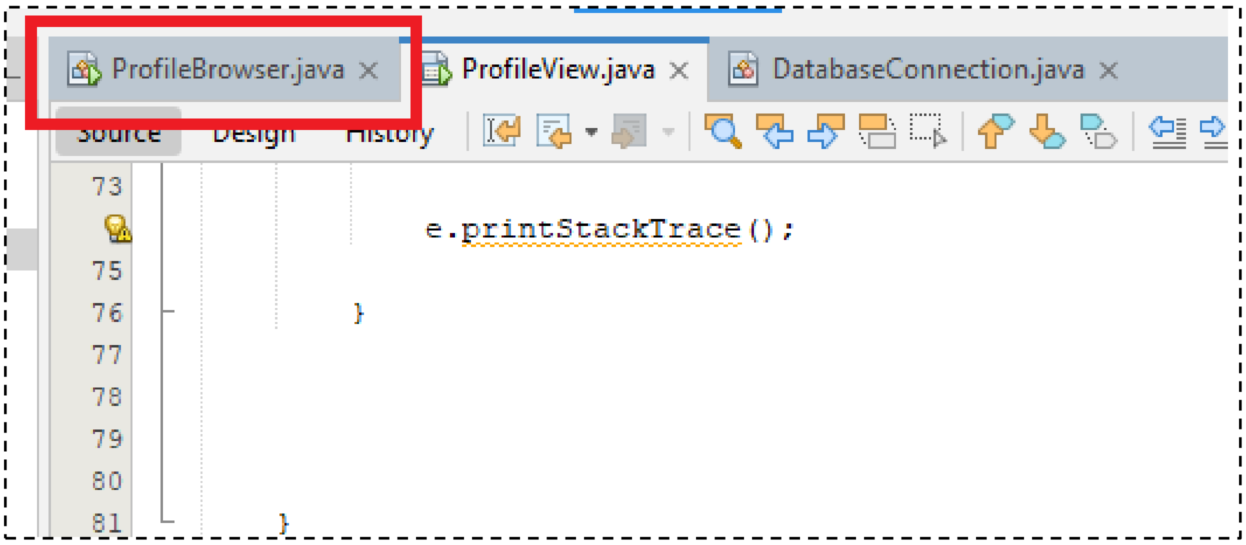
Outside the try block, we are to add the catch definition. Insert this code outside the try block:

```
catch (ClassNotFoundException | SQLException e  ) {

      e.printStackTrace();


 }
```

Here’s the full code of our ProfileView constructor looks like:

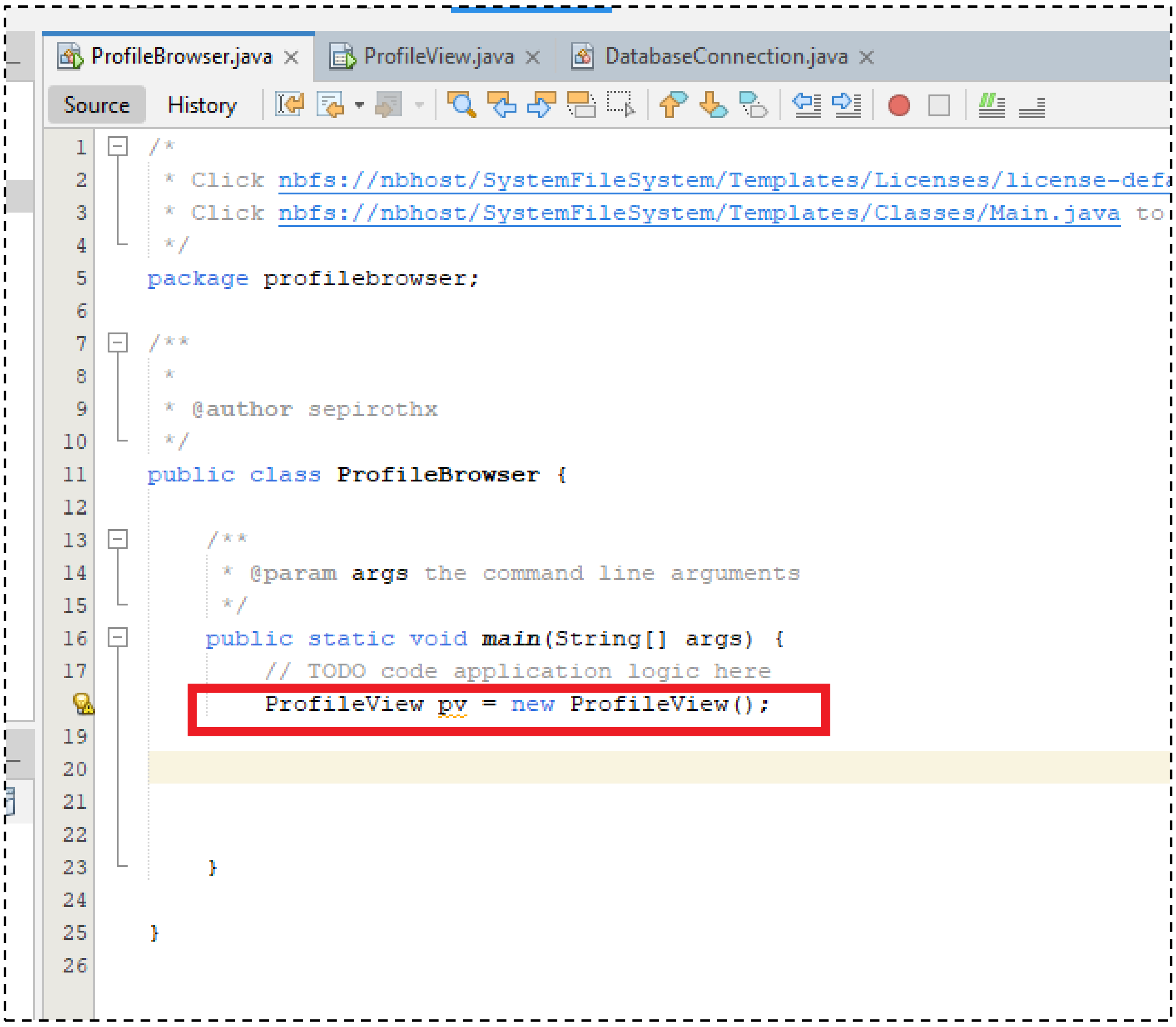
```
public ProfileView() {
        initComponents();
        setLocationRelativeTo(null);
        try {
          // this sql query will pull out all database records
        String query = "SELECT * FROM users"; //SQL Syntax
         //create object for the DatabaseConnection
        DatabaseConnection dbc = new DatabaseConnection();
        //Declare variables to capture database credentials
        String jdbcDriver = dbc.getJdbcDriver();
        String dbConnectionURL = dbc.getDbConnectionURL();
        String dbUsername = dbc.getDbUsername();
        String dbPassword = dbc.getDbPassword();

        Class.forName(jdbcDriver);
        Connection connection = DriverManager.getConnection(dbConnectionURL,
                dbUsername,dbPassword);
        PreparedStatement statement = connection.prepareStatement(query);
        ResultSet resultSet = statement.executeQuery();
        if(resultSet.next()) {
            System.out.println("Connection established!");
            String record_id = resultSet.getString("id");
            String firstName = resultSet.getString("firstname");
            String lastName = resultSet.getString("lastname");
            String school_id = resultSet.getString("school_id");

            System.out.println("Record ID: " + record_id);
            System.out.println("First Name: " + firstName);
            System.out.println("Last Name: " + lastName);
            System.out.println("School ID: " + school_id);
        }
            connection.close();
            resultSet.close();
            statement.close();
        } catch (ClassNotFoundException | SQLException e  ) {

            e.printStackTrace();

        }
```

Now, to run the code inside ProfileView constructor, we need to create an instance (object) for it. Head over to your project's main class and inside the main method, create the needed object. In my case, I have to click on ***ProfileBrowser.java*** because this class contains the ***main()*** method of my project.



```
                        e.printStackTrace();

                    }




            }
```

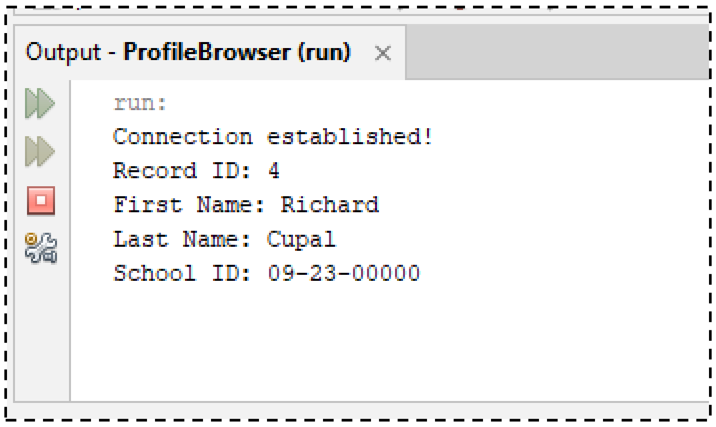
Inside the main method, create the statement to create new object (instance) for ProfileView class as seen below:



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-def
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to
 */
package profilebrowser;

/**
 *
 * @author sepirothx
 */
public class ProfileBrowser {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        ProfileView pv = new ProfileView();



    }

}
```

After adding the necessary code, when the main method runs, it will also run the code inside the constructor of ProfileView Class. Click on **BUILD**, and finally click on **RUN**!

If you get as output the data from your database, **CONGRATULATIONS!** You made a break-through in your Programming Career! You already learned how to connect your Java Application to your database and pull-out its data!



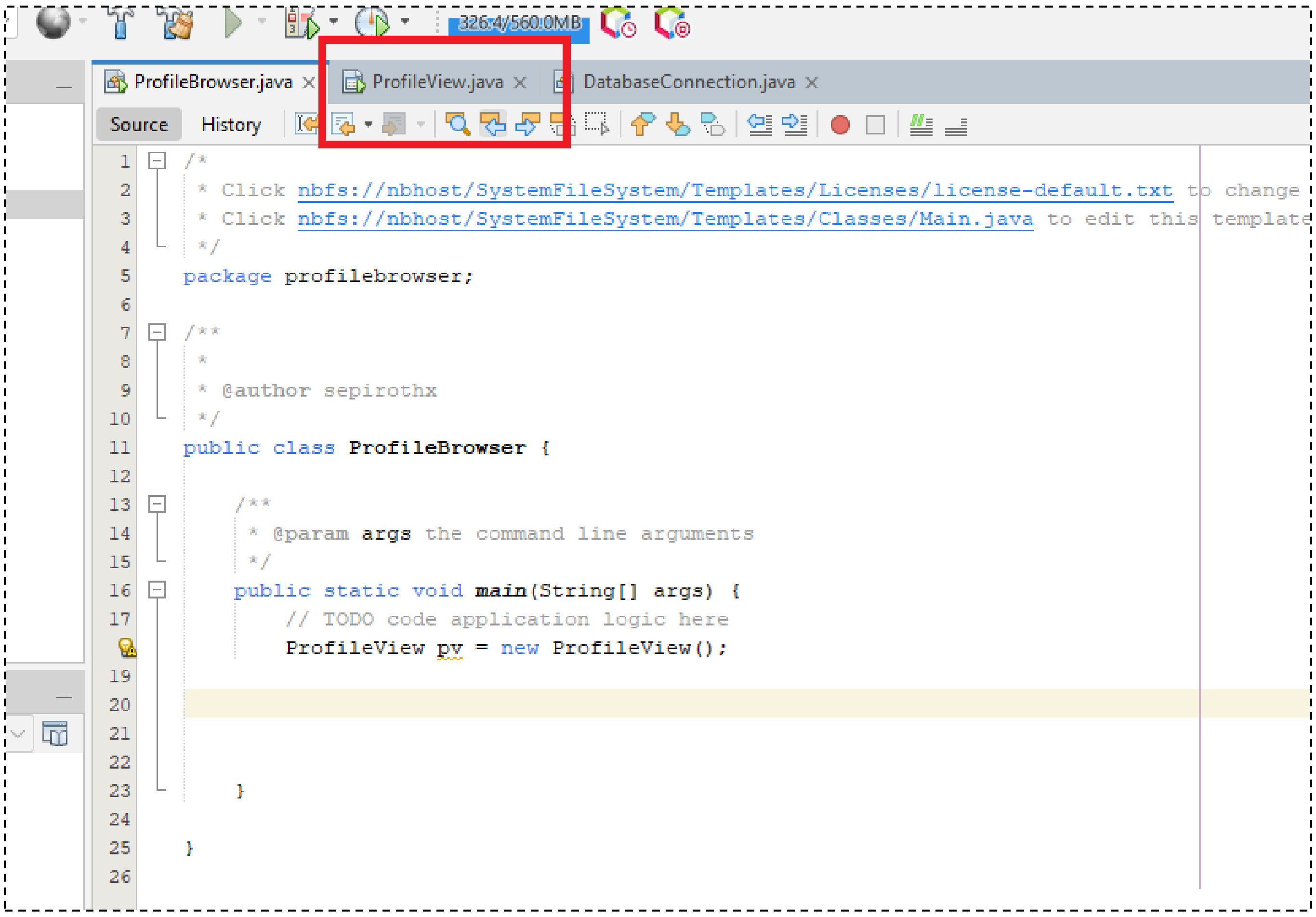
```
run:
Connection established!
Record ID: 4
First Name: Richard
Last Name: Cupal
School ID: 09-23-00000
```

However, we're not done yet! 😉

Let's continue!

Let's display the output data we get in the console to the ProfileView UI we designed earlier in this course.

Click on the **ProfileView.java** tab and let's edit its source code!



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
 */
package profilebrowser;

/**
 *
 * @author sepirothx
 */
public class ProfileBrowser {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        ProfileView pv = new ProfileView();



    }

}
```

Let's add the following statements inside our try block:

```
    //display the output in the UI
    recordNumJL.setText(record_id);
    firstNameTF.setText(firstName);
    lastNameTF.setText(lastName);
    school_numTF.setText(school_id);
```

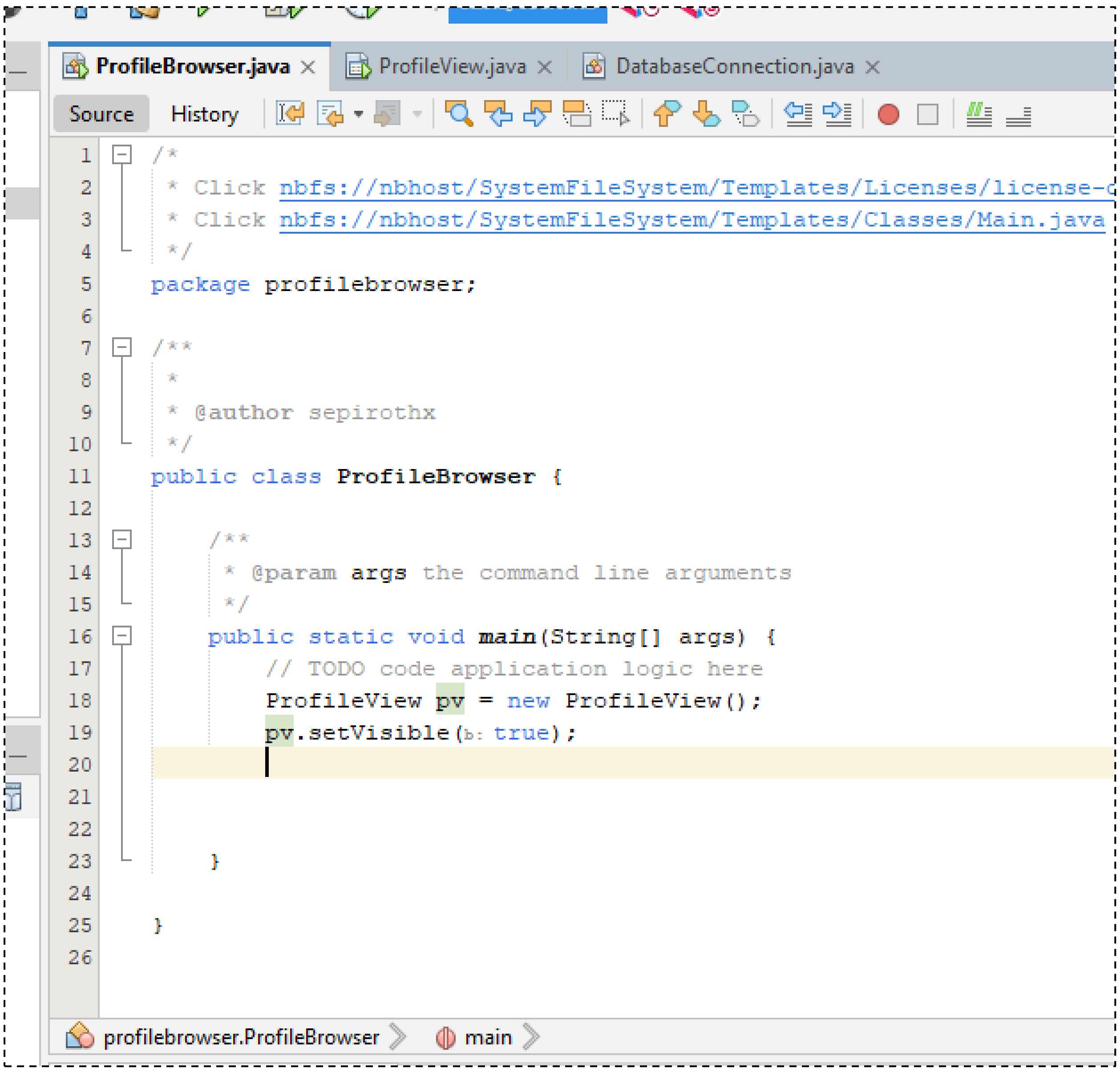
Let me explain the above statements. Remember the identifier (variable names) we gave to our UI objects (jlabel, text fields and buttons)? I just invoked their names and used the **.setText()** available for them. **.setText()** is a pre-defined function and it will set any data passed to it as parameter to any object it is attached to. In our case, I just passed the variables we created which stores the actual data coming from the database query we just did earlier.

After adding the above statements, we just need to add one more statement to show visibly the ***ProfileView*** JFrame so it becomes visible to the user.

Go back to main function inside the main class (in my case, it is ***ProfileBrowser.java***) and add the following statement to show the **ProfileView** UI visibly:

```
//show visible the ui object of ProfileView
pv.setVisible(true);
```

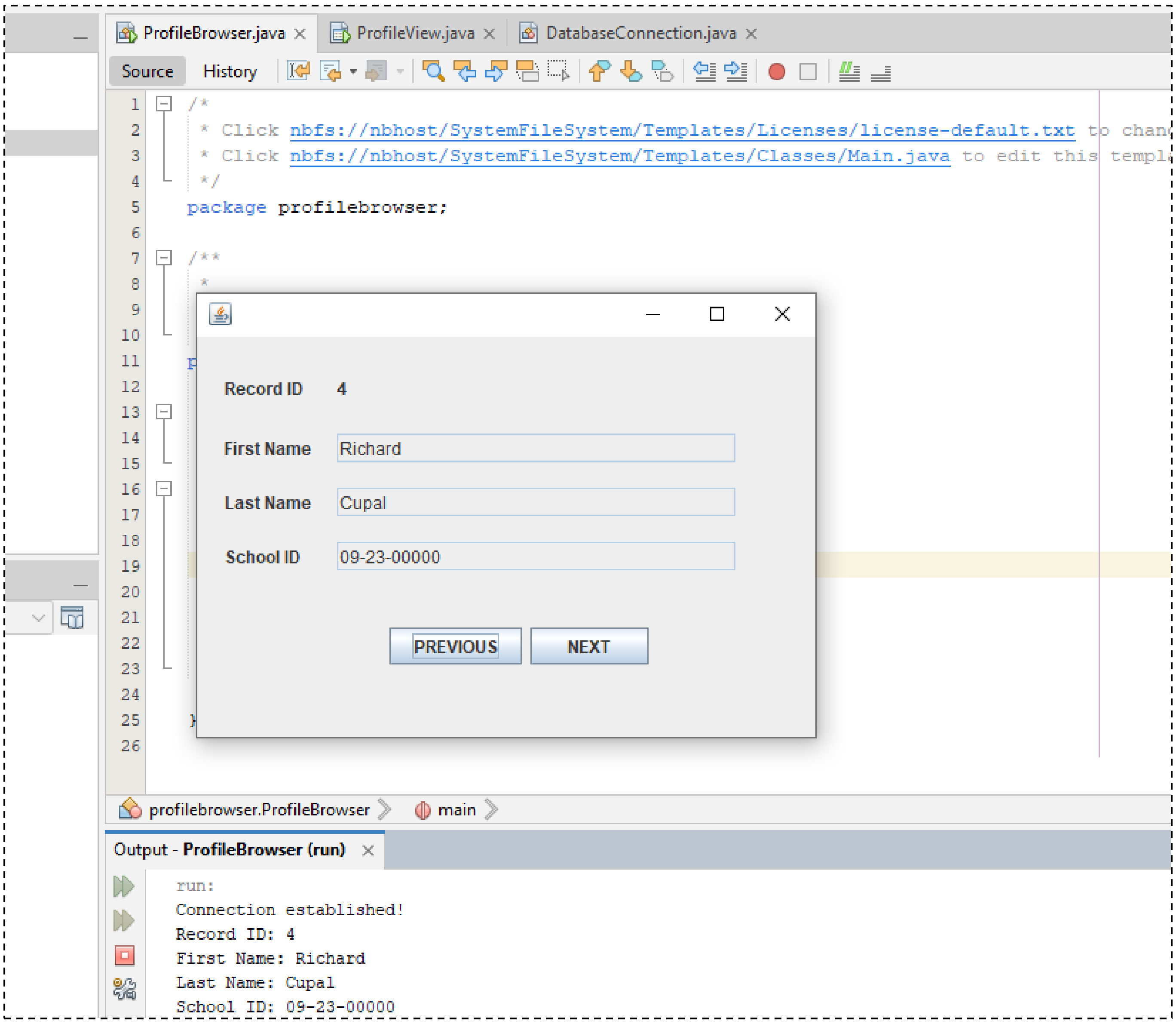
See image below:



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-d
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java
 */
package profilebrowser;

/**
 *
 * @author sepirothx
 */
public class ProfileBrowser {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        ProfileView pv = new ProfileView();
        pv.setVisible(b: true);



    }

}
```

After adding the statement, click on **BUILD** and **RUN!**

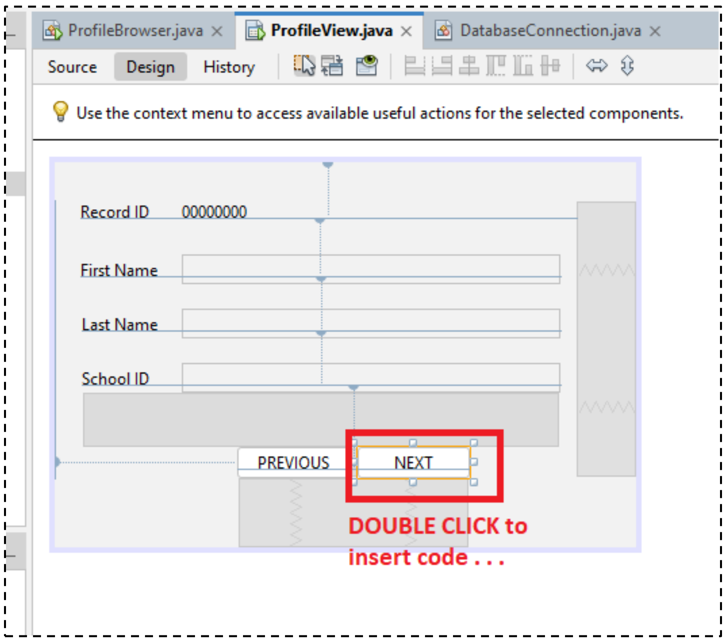
If you get the data to display in the UI just as what you can see below, then **CONGRATULATIONS** again! You made another break-through in your Programming career!



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to chan
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this templ
 */
package profilebrowser;

/**
 *
```

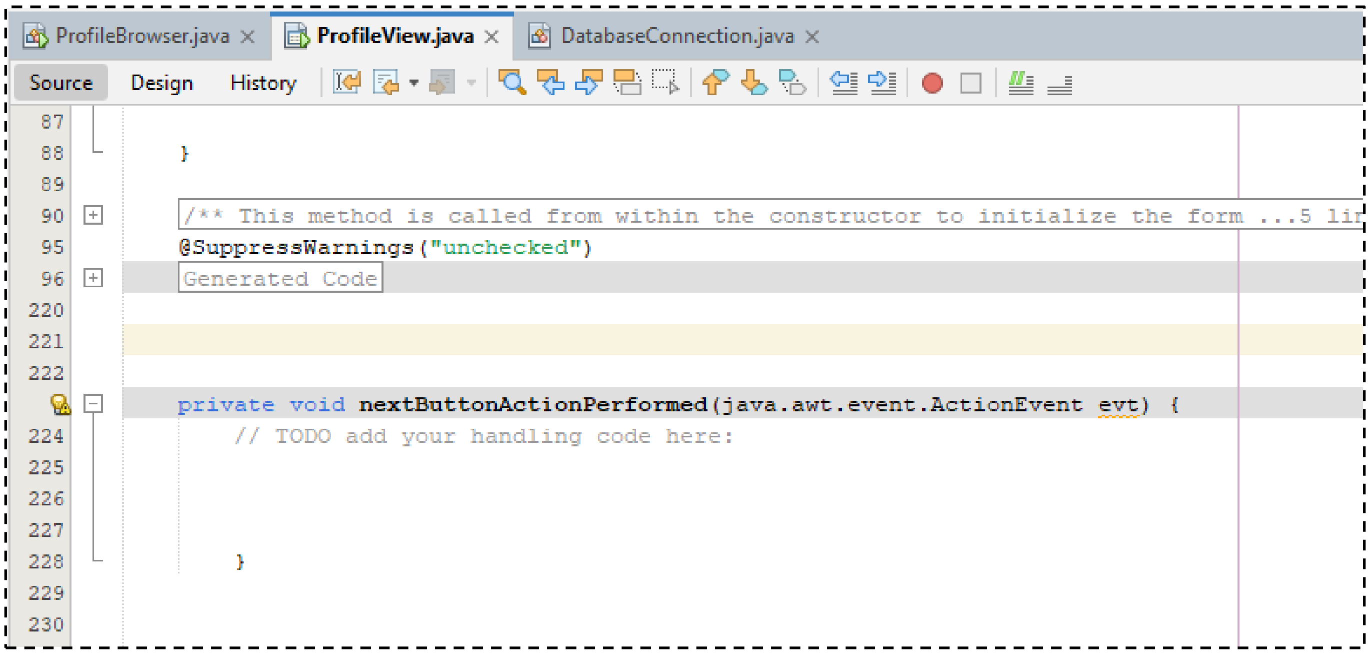
```
run:
Connection established!
Record ID: 4
First Name: Richard
Last Name: Cupal
School ID: 09-23-00000
```

BUT!!! We're not done yet!!! We still need to work on our **'NEXT'** button remember? We want our data to change and display the next record found in the database. Alright, let's get to coding!



WRITING THE SOURCE CODE FOR NEXT BUTTON

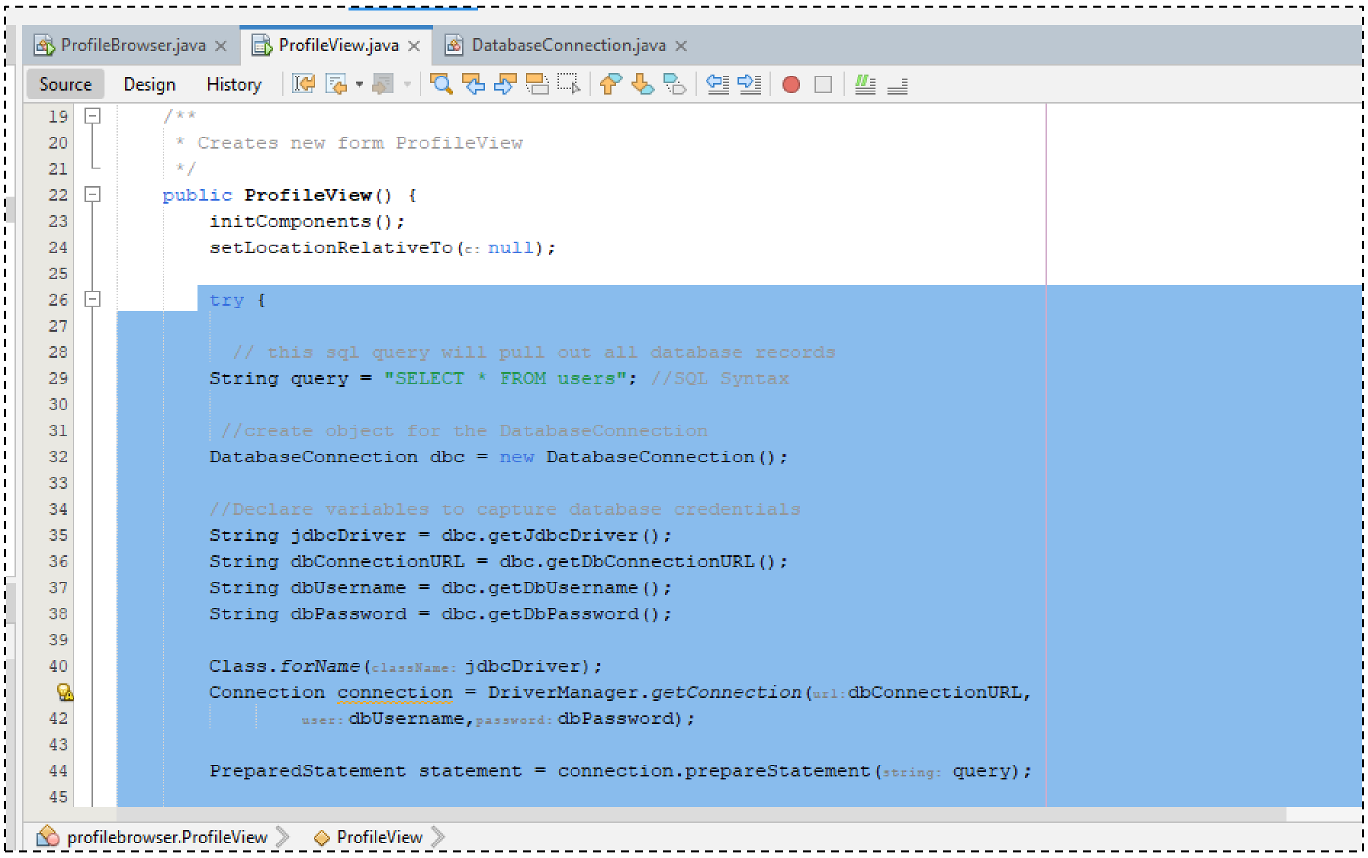
Click on the ***ProfileView.java*** tab again, click on Design and double click on the Next Button to open It's method definition as seen below:



```
    }

/** This method is called from within the constructor to initialize the form ...5 lin
@SuppressWarnings("unchecked")
Generated Code


private void nextButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:



    }
```

To make the Next Button works, we just need to paste the code we wrote inside the try & catch blocks of our ProfileView constructor and do some tweaks to do what we want for next button. Let's copy the code first and paste it inside our button's function definition. Take note of the button identifier. In my case, it is **nextButtonActionPerformed()** yours could be different, please take note! **USE COMMON SENSE!!!**

Copy the entire try and catch blocks and paste them inside our **nextButtonActionPerformed()**;

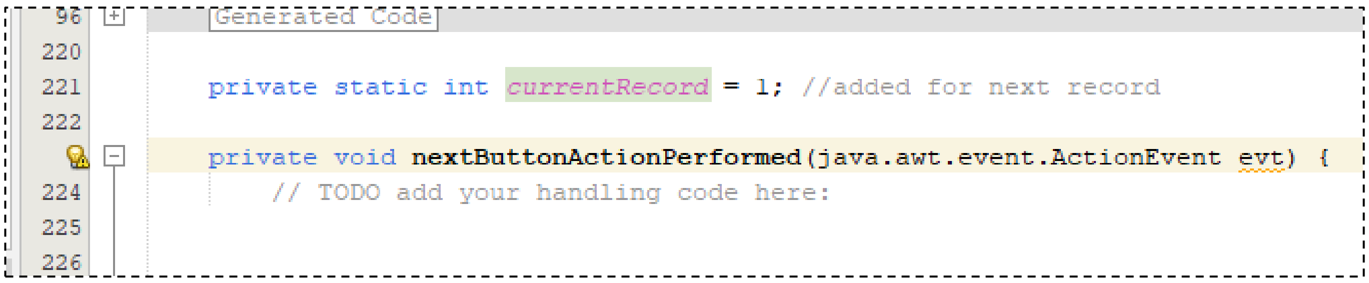


```
/**
 * Creates new form ProfileView
 */
public ProfileView() {
    initComponents();
    setLocationRelativeTo(c: null);

    try {

        // this sql query will pull out all database records
        String query = "SELECT * FROM users"; //SQL Syntax

        //create object for the DatabaseConnection
        DatabaseConnection dbc = new DatabaseConnection();

        //Declare variables to capture database credentials
        String jdbcDriver = dbc.getJdbcDriver();
        String dbConnectionURL = dbc.getDbConnectionURL();
        String dbUsername = dbc.getDbUsername();
        String dbPassword = dbc.getDbPassword();

        Class.forName(className: jdbcDriver);
        Connection connection = DriverManager.getConnection(url: dbConnectionURL,
                user: dbUsername, password: dbPassword);

        PreparedStatement statement = connection.prepareStatement(string: query);
```

Next thing we need to do is to declare a private static integer variable that will keep track of the number of records we are browsing. Let's say, I browse this record the first time, it is one (1), when I browse the record, it becomes (2), so on . . . we will then use this variable as a counter as we traverse each record.

```
// you can add the statement below for the counter
private static int currentRecord = 1;
```

In my case, I will just declare that variable just above my next ButtonActionPerformed() method as seen below:



```
Generated Code

    private static int currentRecord = 1; //added for next record

    private void nextButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:

```

Since we already pasted the try and catch blocks of ProfileView constructor, we can still make use of it and just do minor tweaks for our next button. First to edit is the SQL query.

Within the try block, look for the statement you see below:

```
String query = "SELECT * FROM users";
```

And change it to this statement below:

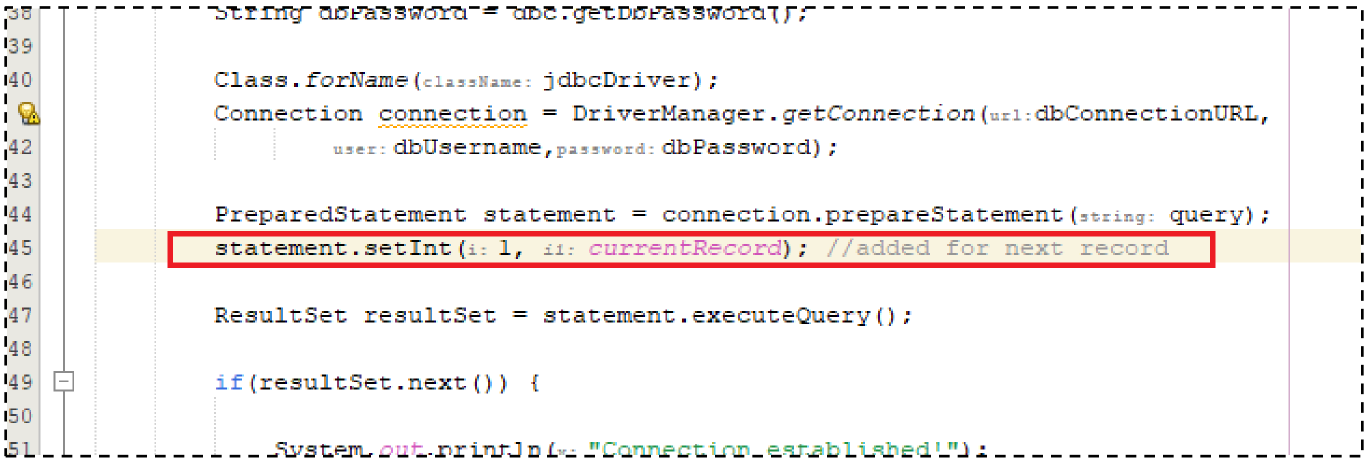
```
String query = "SELECT * FROM users LIMIT ?, 1";
```

What we changed above is the limit of the record to be pulled out by our **PreparedStatement** object (instance). We need to pull it one record only at a time, once a button is clicked and it will automatically update to the next record found.

The next statement we need to add is:

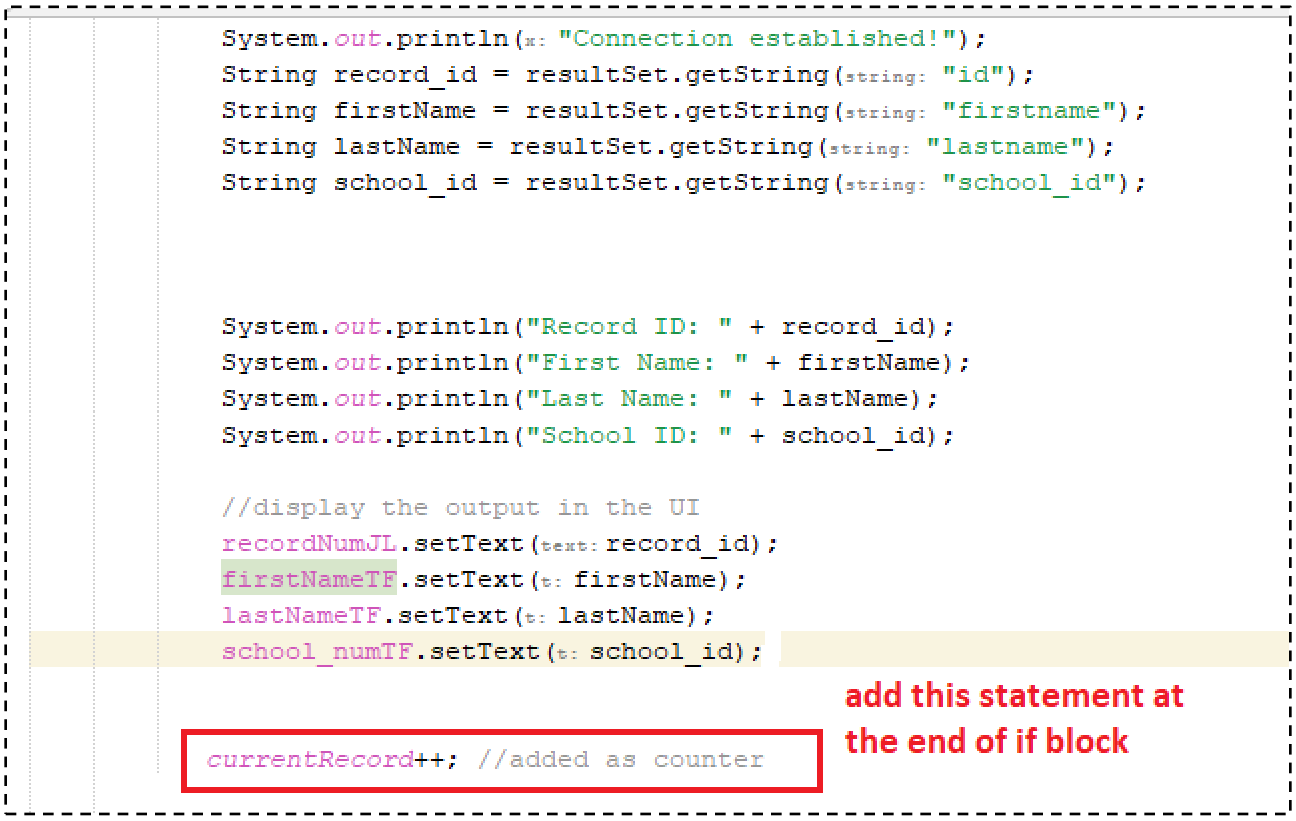
```
statement.setInt(1, currentRecord); //this statement will scan according // to the value
held by the variable currentRecord
```

Just add it below the `PreparedStatement = connection.prepareStatement(query);` as seen below:



```
            String dbPassword = dbc.getDbPassword();

            Class.forName(className: jdbcDriver);
            Connection connection = DriverManager.getConnection(url:dbConnectionURL,
                    user: dbUsername,password: dbPassword);

            PreparedStatement statement = connection.prepareStatement(string: query);
            statement.setInt(i: 1, i1: currentRecord); //added for next record

            ResultSet resultSet = statement.executeQuery();

            if(resultSet.next()) {

                System.out.println(x: "Connection established!");
```

Lastly, let's increment our currentRecord variable inside the if statement block, so we can move to the next record once its value is incremented. Check the image below:



```
System.out.println(x: "Connection established!");
String record_id = resultSet.getString(string: "id");
String firstName = resultSet.getString(string: "firstname");
String lastName = resultSet.getString(string: "lastname");
String school_id = resultSet.getString(string: "school_id");



System.out.println("Record ID: " + record_id);
System.out.println("First Name: " + firstName);
System.out.println("Last Name: " + lastName);
System.out.println("School ID: " + school_id);

//display the output in the UI
recordNumJL.setText(text: record_id);
firstNameTF.setText(t: firstName);
lastNameTF.setText(t: lastName);
school_numTF.setText(t: school_id);

currentRecord++; //added as counter
```

If in any case the application finds no more record in the database to pull or display, let's have something that will prompt our user. Let's put it inside our **else block.**

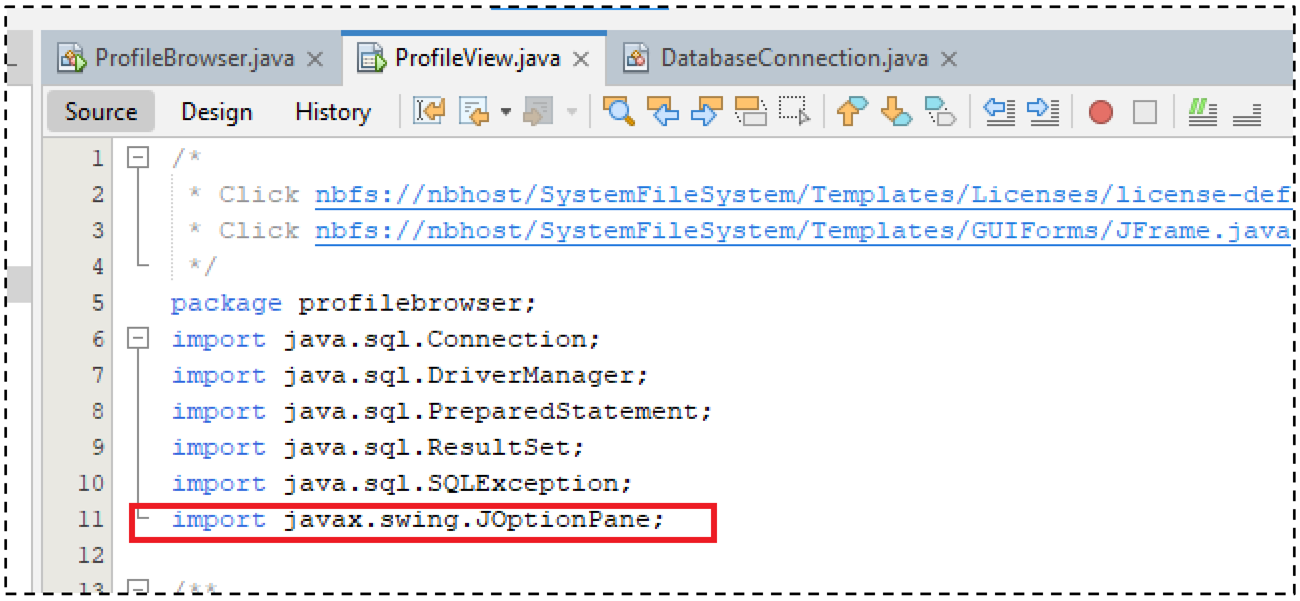
Just outside your if block, add the following code:

```
else {System.out.println("No more record found!");

        JOptionPane.showMessageDialog(null, "No more records found.");
}
```

The above code uses the method inside JOptionPane class. We need to import JOptionPane to make use of it. Let's import it by added the following line just below our import statements as seen below:

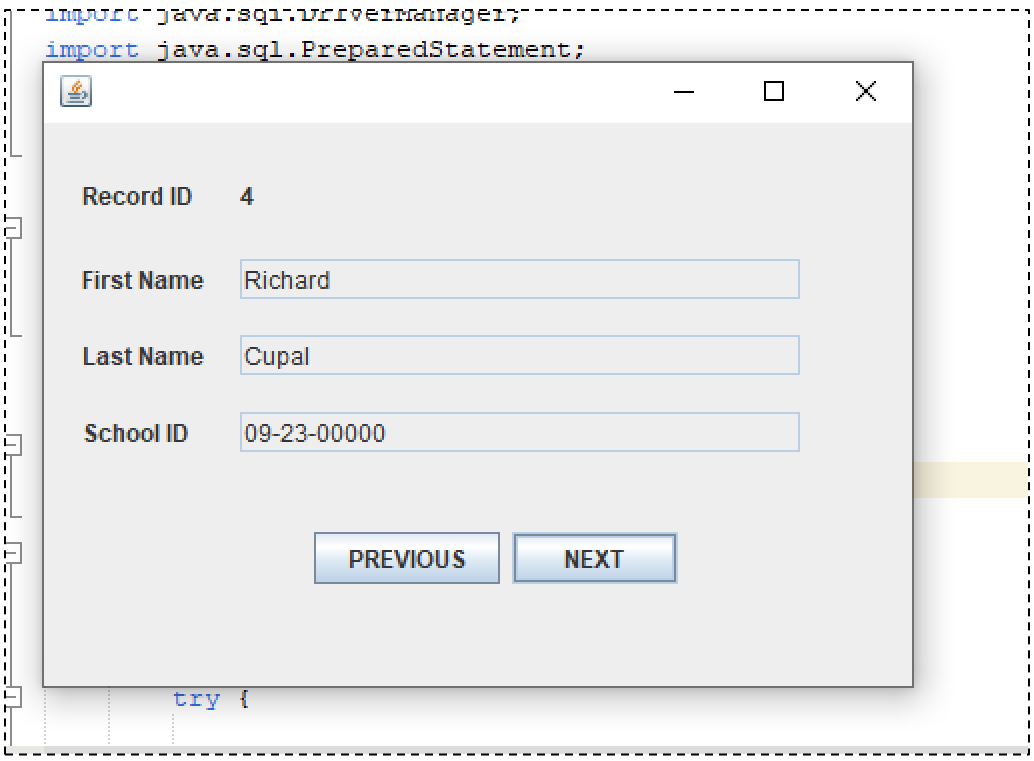
```
import javax.swing.JOptionPane;
```

See the image below for reference:



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-def
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java
 */
package profilebrowser;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.swing.JOptionPane;

/**
```

Now, lets **BUILD** and **RUN** our application!



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

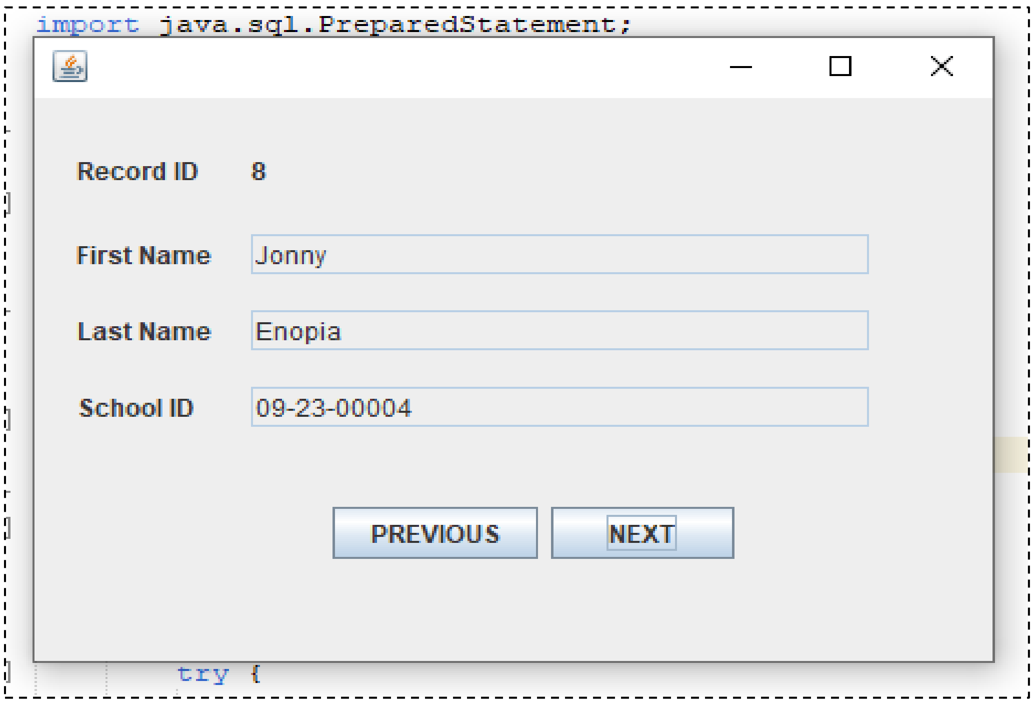
Record ID 4

First Name: Richard

Last Name: Cupal

School ID: 09-23-00000

```
try {
```



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

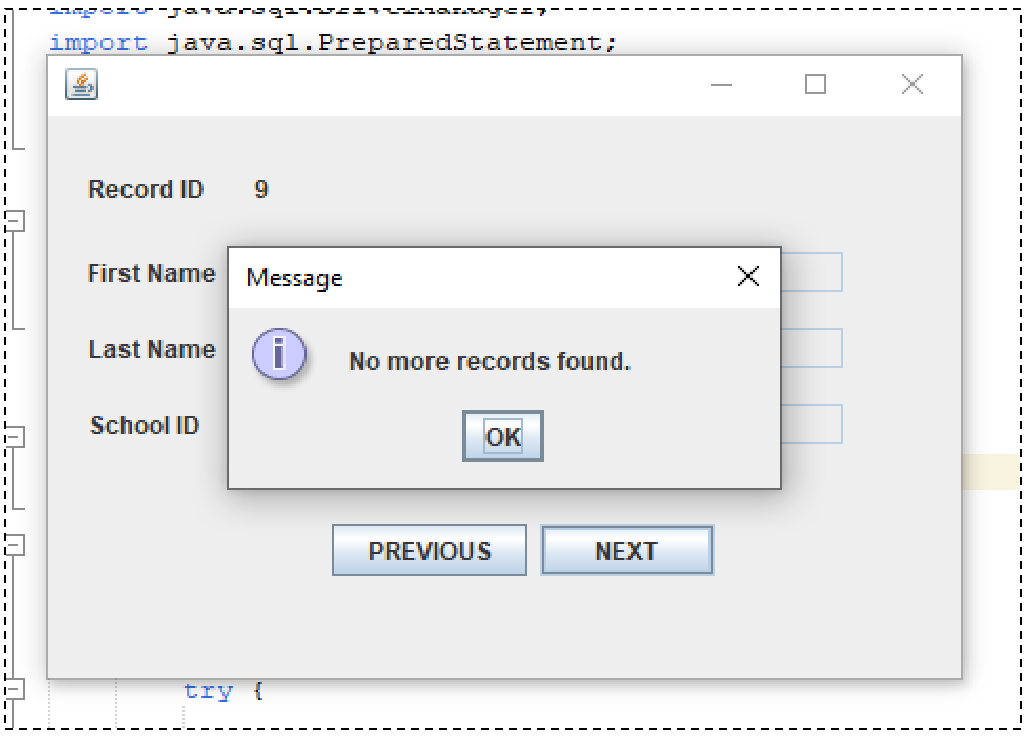
Record ID 8

First Name: Jonny

Last Name: Enopia

School ID: 09-23-00004

```
try {
```



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

Record ID 9

First Name

Last Name

School ID

Message

No more records found.

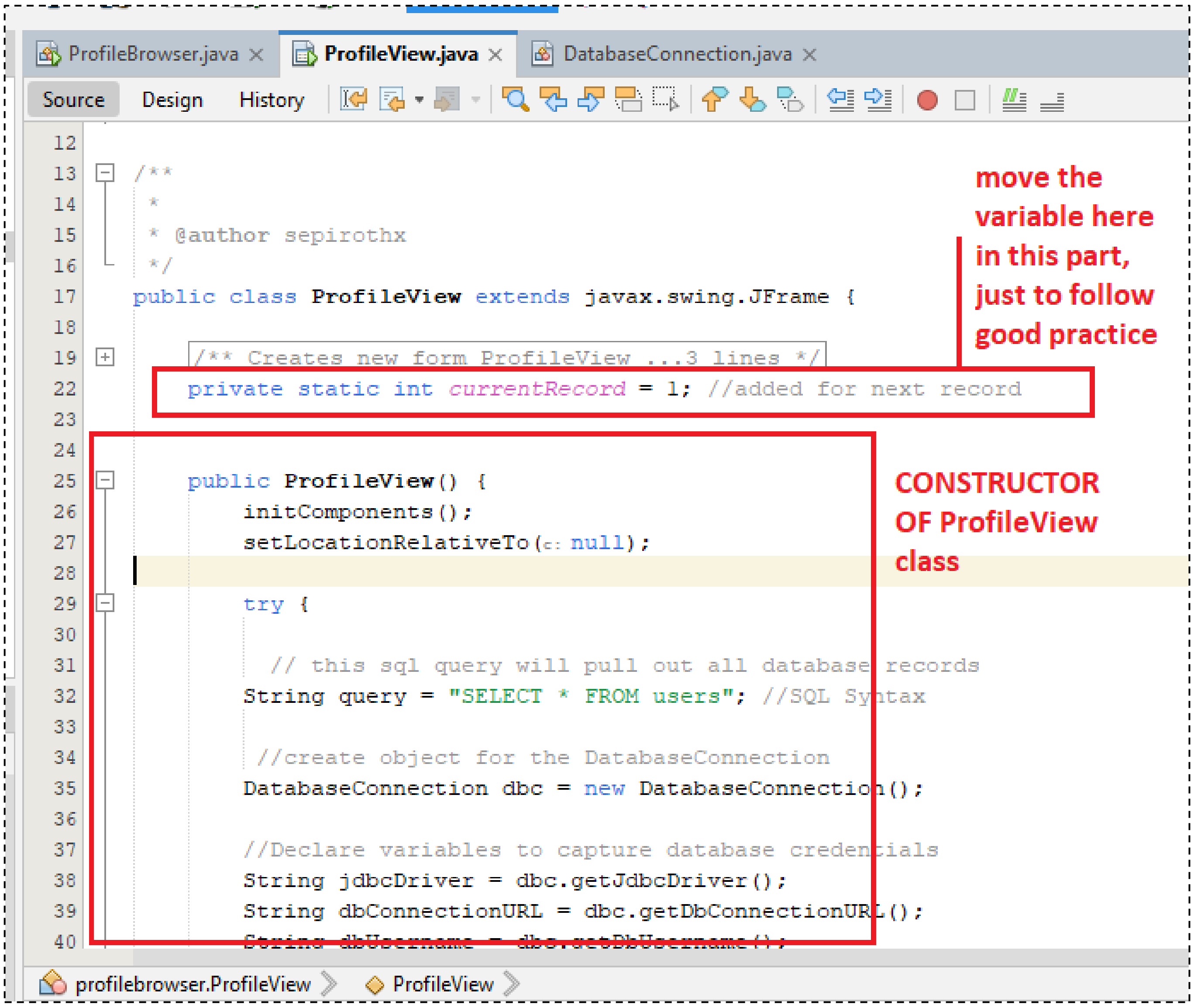
```
try {
```

If the displayed data changes once you clicked on the next, **CONGRATULATIONS!** You just made another break-through!

Now, we have **ONE LAST BUTTON** to work on! You're right, we still have the **PREVIOUS BUTTON**! What code do you think we should do or add to make it work? Let's find out!

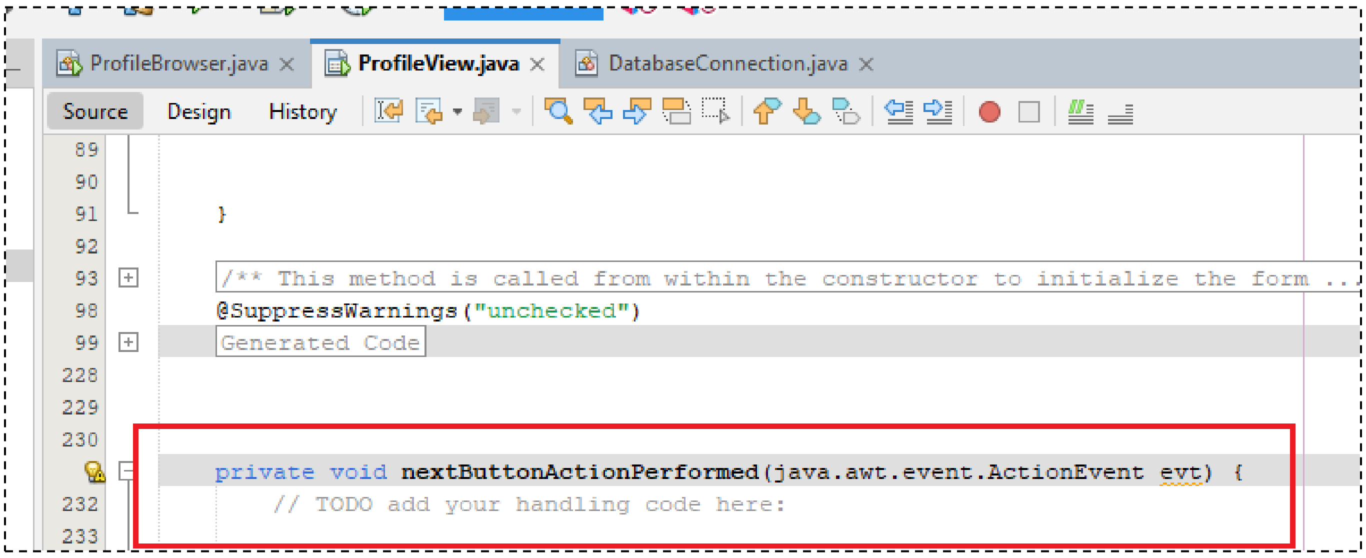
Click again on **ProfileView.java** tab, click on Design and double click on the PREVIOUS button!

Remember the private static variable we declared a while ago with an identifier **currentRecord**? Let's cut that one and move just above of our source code, just before the declaration of ProfileView constructor as seen below.


ProfileBrowser.java
ProfileView.java
DatabaseConnection.java
Source
Design
History
move the variable here in this part, just to follow good practice
CONSTRUCTOR OF ProfileView class
profilebrowser.ProfileView
ProfileView

```
/**
 *
 * @author sepirothx
 */
public class ProfileView extends javax.swing.JFrame {

    /** Creates new form ProfileView ...3 lines */
    private static int currentRecord = 1; //added for next record


    public ProfileView() {
        initComponents();
        setLocationRelativeTo(c: null);

        try {

          // this sql query will pull out all database records
        String query = "SELECT * FROM users"; //SQL Syntax

         //create object for the DatabaseConnection
        DatabaseConnection dbc = new DatabaseConnection();

        //Declare variables to capture database credentials
        String jdbcDriver = dbc.getJdbcDriver();
        String dbConnectionURL = dbc.getDbConnectionURL();
        String dbUsername = dbc.getDbUsername();
```

Now, let's go again to ProfileView.java, Design and double click on Previous Button so you'll be directed right away to the part of the source code where the definition of the previous button resides as seen below:



```
    }

    /** This method is called from within the constructor to initialize the form ...
    @SuppressWarnings("unchecked")
    Generated Code


    private void nextButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
```

PREVIOUS BUTTON SOURCE CODE: For the source code for previous button, just copy the source code below

```
try {
  if(currentRecord > 0){
       currentRecord--; //added as counter
       String query = "SELECT * FROM users LIMIT ?, 1"; //added to limit result to 1
       //create object for the DatabaseConnection DatabaseConnection dbc = new DatabaseConnection();

       //Declare variables to capture database credentials
       String jdbcDriver = dbc.getJdbcDriver();
       String dbConnectionURL = dbc.getDbConnectionURL();
       String dbUsername = dbc.getDbUsername();
       String dbPassword = dbc.getDbPassword();

       Class.forName(jdbcDriver);
       Connection connection = DriverManager.getConnection(dbConnectionURL,
                             dbUsername,dbPassword);
       PreparedStatement statement = connection.prepareStatement(query);
       statement.setInt(1, currentRecord); //added for next record
       ResultSet resultSet = statement.executeQuery();
              if(resultSet.next()) {
                 System.out.println("Connection established!");
                 String record_id = resultSet.getString("id");
                 String firstName = resultSet.getString("firstname");
                 String lastName = resultSet.getString("lastname");
                 String school_id = resultSet.getString("school_id");

                 System.out.println("Record ID: " + record_id);
                 System.out.println("First Name: " + firstName);
                 System.out.println("Last Name: " + lastName);
                 System.out.println("School ID: " + school_id);
                 //display the output in the UI
                 recordNumJL.setText(record_id);
                 firstNameTF.setText(firstName);
                 lastNameTF.setText(lastName);
                 school_numTF.setText(school_id);
             } else {
                JOptionPane.showMessageDialog(null, "Beginning of record!"); }
           connection.close();
           resultSet.close();
           statement.close();
       } else {System.out.println("No more records found.");
       JOptionPane.showMessageDialog(null, "No more records found."); }
     } catch (ClassNotFoundException | SQLException e  ) {
           e.printStackTrace(); }
```

Please use common sense for the above source code. You are paste it inside function definition of Previous Button. You can browse the function definition of the previous button by double-clicking on the button object itself. It is there where you are going to paste the source code.

Now, **BUILD** and **RUN**!

If The record changes by clicking either the NEXT or PREVIOUS button, then that means you've successfully followed the entire course and can proudly tell yourself that you're indeed a **PROGRAMMER**!

GIVE YOURSELF A ROUND OF APPLAUSE FOR YOU DESERVE IT! YOU MADE ME PROUD!

The source codes for this tutorial have been uploaded on my github page!

Here's the link: **https://github.com/sepiroth-x/Java-Projects-Source-Codes/tree/main**

Follow my socials:

Facebook: https://facebook.com/tsadiqz3

Instagram: https://instagram.com/sepiroth.x/

Twitter: https://twitter.com/sepirothx000

Github: https://github.com/sepiroth-x

Patreon: https://www.patreon.com/vajrayogiii

Gcash: +639150388448

Email: richard.cupal@ifamsocial.com , chardy.tsadiq@gmail.com

---