Introduction to Java Servlet

Overview of Application Server

- For hosting template text / static pages / hard coded (html) and server side application (cgi / servlet)
- Most of the time integrated with a simple web-server
- Can be plugged to a more powerful web-server
- Ranging from million of dollars in pricing to the open source project which is free

Overview of Application Server

- Existing application server in the market
  - Apache TOMCAT
  - Xampp
  - WAMP
  - LAMP
  - JRUN
  - Jboss
  - GlassFish

What is Servlet

- a web component,
- managed by a container (Application Server),
- generates dynamic content. (HTML TAG)
    - serving a different page according to client data submitted via a form
    - Or a GET method
- small, platform independent Java classes compiled to a bytecode that can be loaded dynamically into and run by a web server.
- interact with web clients via a request response paradigm implemented by the servlet container.
- request-response model is based on the behavior of the Hypertext Transfer Protocol (HTTP).

Application Example

- Web Information Systems
- Distributed Computing
- E-Commerce systems
- Dynamic information systems – weather reports, stock quotes, search engines etc.

Advantage of Servlet over CGI

- The most important factor – Server Process
  - CGI, new process for every http request
  - overhead of starting the process – dominate execution time
  - Servlets, JVM stay running and handle each request using a lightweight Java thread
  - CGI : N simultaneous request – CGI program load N times
  - Servlet : N copy of thread but only one copy of the servlet class

Advantage of Servlet

- Convenient
  - for a Java programmer – no need to learn a new language
- Powerful.
  - Java servlets is a Java program and can do whatever Java program can do in a local machine. This simplifies operations that need to look up images and other data stored in standard places.
  - Servlets can also share data among each other, making useful things like database connection pools easy to implement.
  - They can also maintain information from request to request, simplifying things like session tracking and caching of previous computations.

Advantage of Servlet

- Portable.
  - Servlets are written in Java and follow a well-standardized API – WORA. Consequently, servlets written for, say I-Planet Enterprise Server can run virtually unchanged on Apache, tomcat etc.
  - Servlets are supported directly or via a plugin on almost every major Web server.
- Inexpensive.
  - There are a number of free or very inexpensive Web servers available that are good for "personal" use or low-volume Web sites.
  - However, with the major exception of Apache, which is free, most commercial-quality Web servers are relatively expensive.

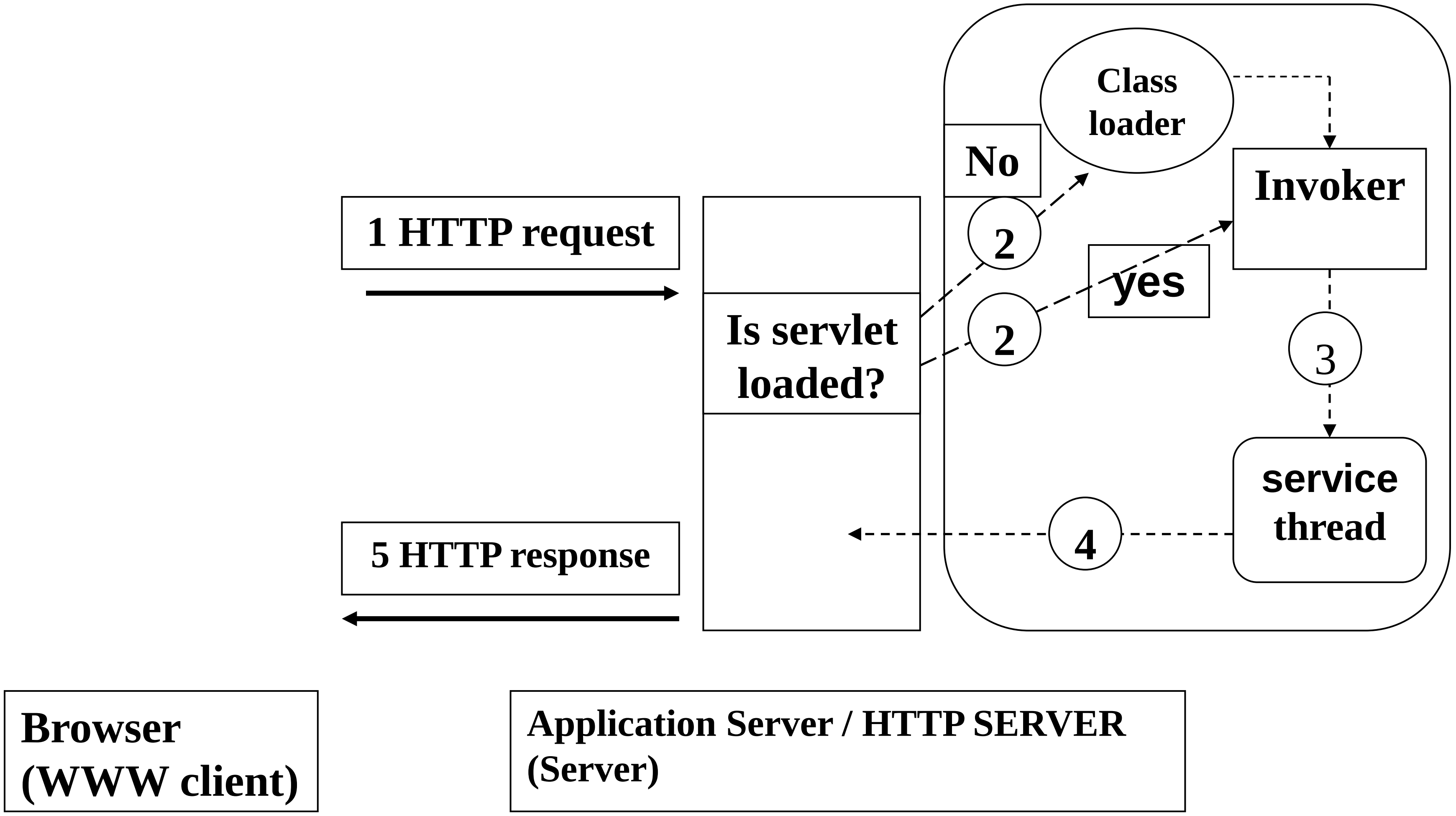
Servlet Tasks

- Read sent user data –
  - via form (POST protocol) or
  - embedded URL (GET protocol)
- Look up info on http request – usually form/POST data or URL/GET data
- Generate result (connect to db etc.),
- Format the html result
- Set the appropriate http response parameters – set content type html/text etc.
- Send document (HTML page) back to client browser

Client Interaction

- When a servlet accepts a call from a client, it receives two objects:
- A ServletRequest, which encapsulates the communication from the client to the server. – getParameter()
- A ServletResponse, which encapsulates the communication from the servlet back to the client – out.println()
- ServletRequest and ServletResponse are interfaces defined by the javax.servlet package

Request & Response Overview



Normal Servlet Operation

- extends HttpServlet interface
- implement one or more service methods
  - doGet, doPost, etc
- Setting the content type
- Data processing
- Formatting presentation HTML
- Returning a response

Intro to Netbeans – HelloWorld servlet
<IMG> tag
<A> tag
<TABLE> tag

Data Transmission

- there are two ways on how browser can send data to a servlet via HTTP protocol
  - GET method
  - POST method

GET method

- The body of the message (the data) is appended to the servlet URL,
  - `http://localhost/servlet/HelloWorld`
- Separated by a question mark
  - `http://localhost/servlet/HelloWorld?`
- Followed by name-value pair which separated by equals sign
- If value consist of more than one word, separate it using plus sign which the servlet will convert it to space character after parsing
  - `name=ajune+ismail`
- Every consecutive name-value pair will be separated using ampersand sign (&)
  - `name=ajune+ismail&ic=h0803907`

Hello Get
Sum / Sum Oper GET
Table GET

POST method

- The body of the message is sent as a stream of data (HTML form data)
- Separated with the servlet URL
- Client send data to servlet using HTML form element

HTML Form element

- **Form tag**

```
<FORM METHOD="post"
        ACTION="/servlet/HelloWorld"
        TARGET="frameName">
```

- Fill the **`TARGET`** value if form result have to display in a different frame
- After coding all the form element (button, textfield, etc) FORM tag must be close using the equivalent end tag – </**FORM**>
- If you have multiple form in a single page every separate every form using the end tag

HTML Form element

- **Textfield element**
  - Single line textbox
  - Code example:
    - `<INPUT NAME="name" TYPE="text" SIZE="25">`
- **Password element**
  - Single line textbox – actual text hidden
  - Code example:
    - `<INPUT NAME="password" TYPE="password" SIZE="25">`
- **TextArea element**
  - Multiline textbox
  - Code example:
    - `<TEXTAREA NAME="address" ROWS="5" COLS="23"></TEXTAREA>`

HTML Form element

- Combo Box
  - Single item selection permitted

```
<SELECT NAME=”creditCardType”>
<OPTION SELECTED VALUE=”mc”>MasterCard
<OPTION VALUE=”visa”>VISA
<OPTION VALUE=”amex”>American Express
</SELECT>
```

- List Box
  - Multiple item selection permitted

```
<SELECT NAME=”language” MULTIPLE>
<OPTION SELECTED VALUE=”c”>C
<OPTION VALUE=”c++”>C++
<OPTION VALUE=”java”>Java
</SELECT>
```

HTML Form element

- **Radio Button**
- Only one item selection permitted

```
<INPUT TYPE="RADIO" NAME="creditCard"
                    VALUE="mc"
  CHECKED>MasterCard
<INPUT TYPE="RADIO" NAME="creditCard"
                    VALUE="visa">VISA
<INPUT TYPE="RADIO" NAME="creditCard"
                    VALUE="amex">American
  Express
```

HTML Form element

- CheckBox
  - **Name** & **Value** attribute are only sent to the server (servlet) if the check box is checked
  - Usually servlet/CGI programs often check only for the existence of the checkbox name, ignoring its value
  - Multiple item selection permitted

```
<P>
<INPUT TYPE="CHECKBOX" NAME="mailMe" CHECKED>
Check here if you want to get our email newsletter
```

HTML Form element

- Push Buttons
  - **Submit Buttons**

```
<INPUT NAME="name" TYPE="submit"
                   VALUE="Submit">
```

  - Change value of attribute VALUE if you want to change the button label

  - **Reset Buttons**

```
<INPUT NAME="name" TYPE="reset"
                        VALUE="Reset">
```

Form Servlet
Calc POST

Java Database Connectivity – JDBC API

- Java API for accessing virtually any kind of tabular data
- Consists of
  - a set of classes and interfaces
  - written in the Java programming language that
  - provide a standard API for tool/database developers
- Guarantee that an application can access virtually any data source and run on any platform with a Java Virtual Machine

4 types of JDBC Driver

- **JDBC-ODBC Bridge plus ODBC driver**
  - provides JDBC access via ODBC drivers
  - Windows platform only
  - Cannot be used directly from browser (HTTP protocol)
  - Can be solved by using a middleware such as RMI / CORBA
- **Native-API partly-Java driver**
  - driver converts JDBC calls into calls on the client API
  - requires that some operating system-specific binary code be loaded on each client machine

4 types of JDBC Driver

- **JDBC-Net pure Java driver**
  - driver translates JDBC calls into a DBMS-independent net protocol,
  - act as middleware server
  - able to connect its pure Java clients to many different databases – the **most flexible** JDBC alternative
- **Native-protocol pure Java driver**
  - converts JDBC calls directly into the network protocol used by DBMS
  - This allows a direct call from the client machine to the DBMS server – perfect for Internet JDBC access
  - available only by DBMS vendor – Oracle, Sybase etc.

Using JDBC

- **Install Java and JDBC API** on your machine – JDBC preinstalled with JDK (standard API)
- **Install a JDBC driver** on your machine. – Usually DMBS comes with its own JDBC driver
  - JDBC-ODBC driver pre-installed with the JDK
- **Install your DBMS** if needed (connection can also be done in remote)
- **Setting Up a Database** – creating table, relationships etc.
- **Establishing a Connection**
  - For selecting, adding, modifying and deleting
  - Closing connection

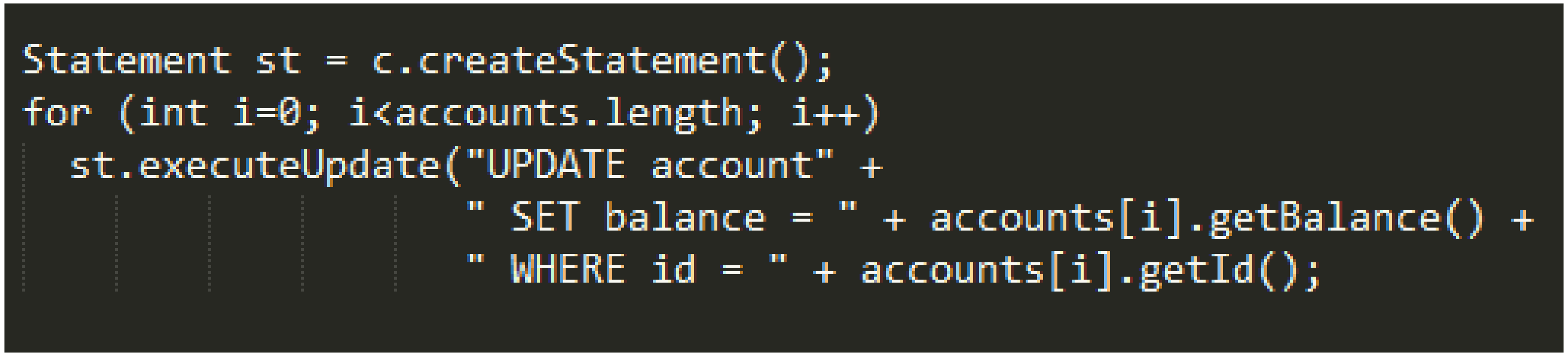
Prepared SQL

- For each SQL statement received, the DB builds a query plan by
  - parsing the SQL statement
  - reading the SQL to determine what to do
  - formulating a plan for executing the SQL
- Repeatedly executing SQL with same query plan is very inefficient

Prepared SQL (contd.)

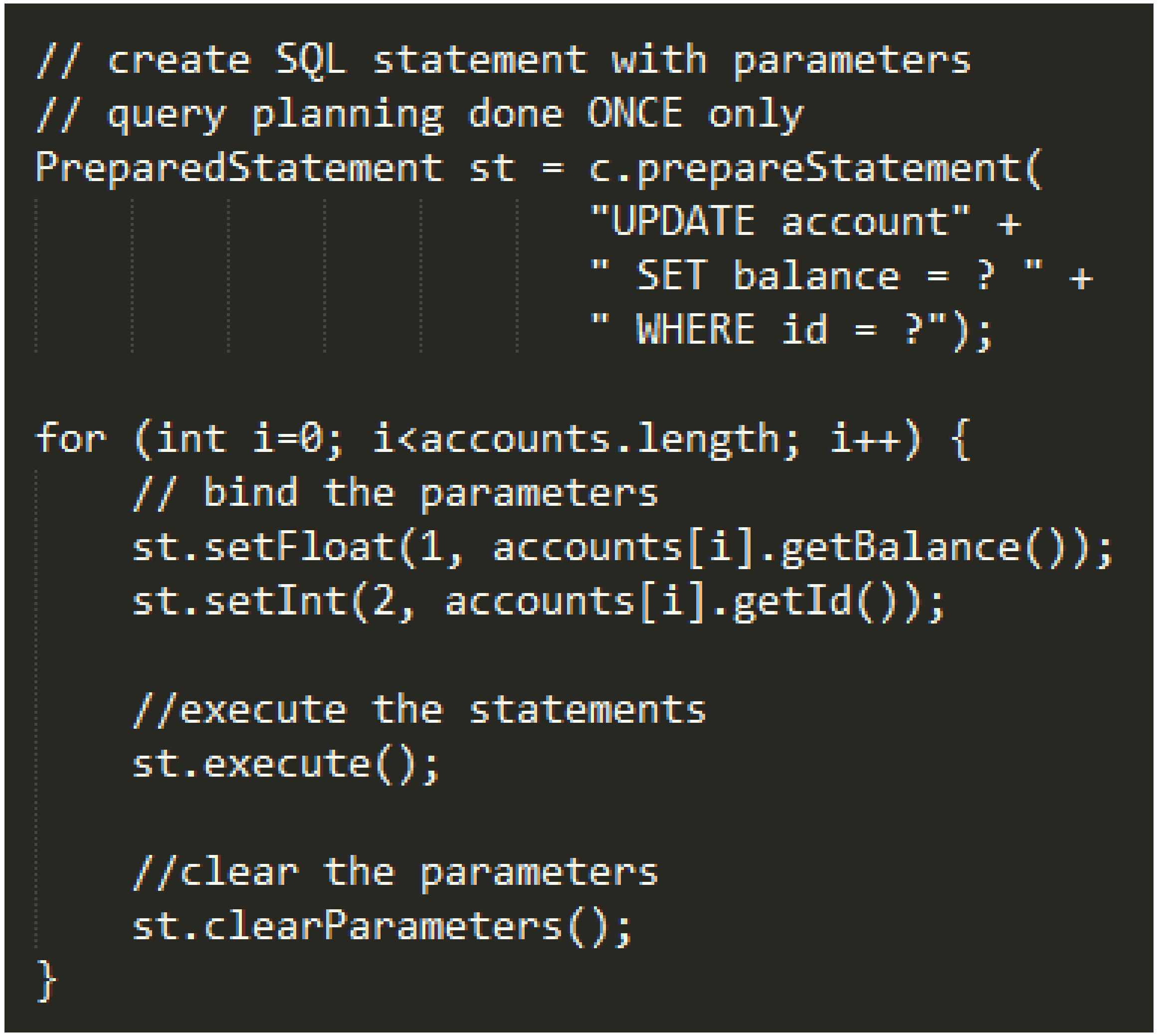
- DBs enable you to optimize repeated calls through prepared SQL.
- Create a Java instance of a prepared statement that notifies the DB of the kind of SQL call it represents.
- DB can then create a query plan for that SQL even before it is actually executed.
- If the same prepared statement is executed more than once, the DB uses the same query plan without rebuilding a new one.

Normal SQL – query planning is repeating each time SQL executed



```
Statement st = c.createStatement();
for (int i=0; i<accounts.length; i++)
  st.executeUpdate("UPDATE account" +
                   " SET balance = " + accounts[i].getBalance() +
                   " WHERE id = " + accounts[i].getId();
```

Using prepared statements Query planning done once Statement can be executed repeatedly



```java
// create SQL statement with parameters
// query planning done ONCE only
PreparedStatement st = c.prepareStatement(
                        "UPDATE account" +
                        " SET balance = ? " +
                        " WHERE id = ?");

for (int i=0; i<accounts.length; i++) {
    // bind the parameters
    st.setFloat(1, accounts[i].getBalance());
    st.setInt(2, accounts[i].getId());

    //execute the statements
    st.execute();

    //clear the parameters
    st.clearParameters();
}
```

Prepared statement operations

1. Prepare the statement

```
// create SQL statement with parameters
PreparedStatement st = c.prepareStatement("UPDATE student" +
                                          " SET name = ?" +
                                          " WHERE id = ?");
```

2. Fill in the statement parameters with data

```
//get the data
String name = request.getParameter("name");

//fill in the statement parameters with the data
st.setString(1, name);
```

3. Execute the statement

```
st.executeUpdate();
```

4. Process the result if there is any

Prepared statement operations

- Operation 1 do ONCE ONLY!
- Operation 2, 3 and 4 can be repeated indefinitely using the prepared statements in 1

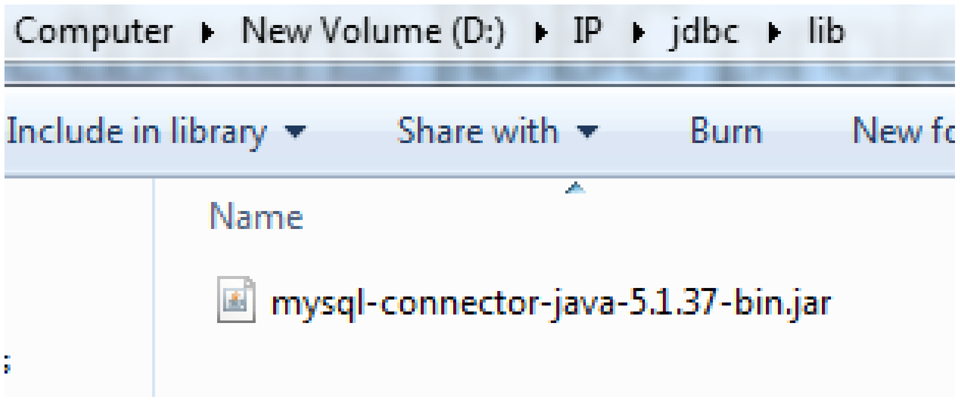
Installing database

- We are using mysql database
- Download xampp and install
- Open phpmyadmin
- Import sql (use the one from dropbox)

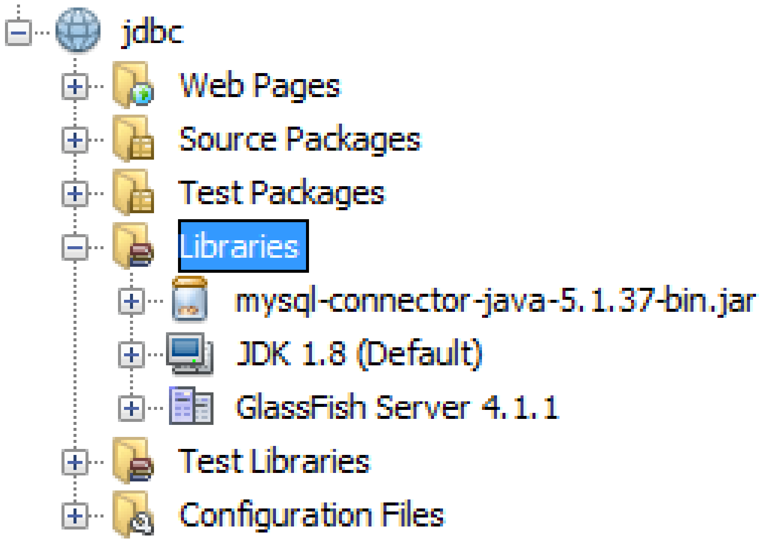
Netbeans JDBC project connect to mysql database

1. Create new Java Web project
2. Using windows explorer locate the project folder
3. Create lib folder in the project folder
4. Download **mysql-connector-java-X.X.X.X-bin.jar**
   - From https://dev.mysql.com/downloads/connector/j/
   - From dropbox
5. Copy the jar files in the lib folder (**task 3**)
6. Right click Libraries in the netbeans project:
   - choose Add Jar/Folder …
   - Add the mysql-connector-java from the lib folder (task 5)

Netbeans JDBC project connect to mysql database



mysql-connector-java-5.1.37-bin.jar



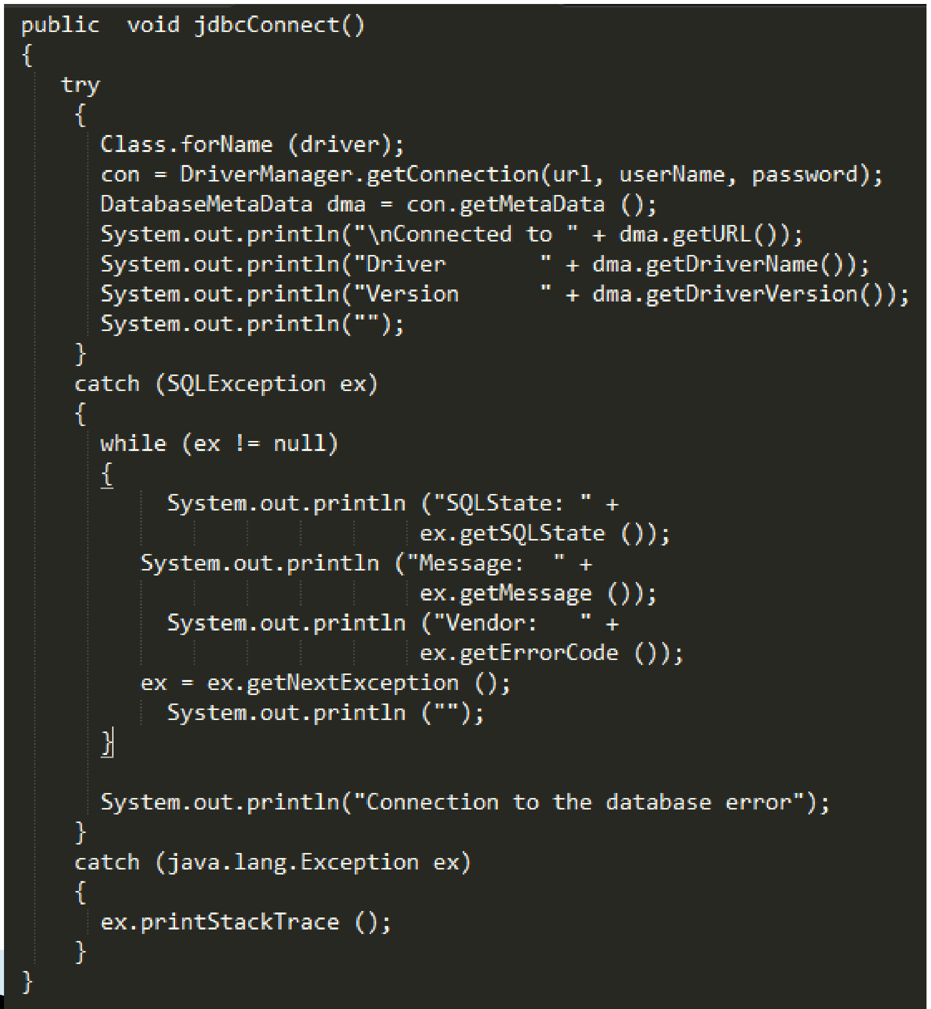
Custom JDBC class – JDBCUtility

- A constructor database connection details

```
public JDBCUtility(String driver,
                   String url,
                   String userName,
                   String password)
  {
     this.driver = driver;
     this.url = url;
     this.userName = userName;
     this.password = password;
  }
```

Custom JDBC class – JDBCUtility

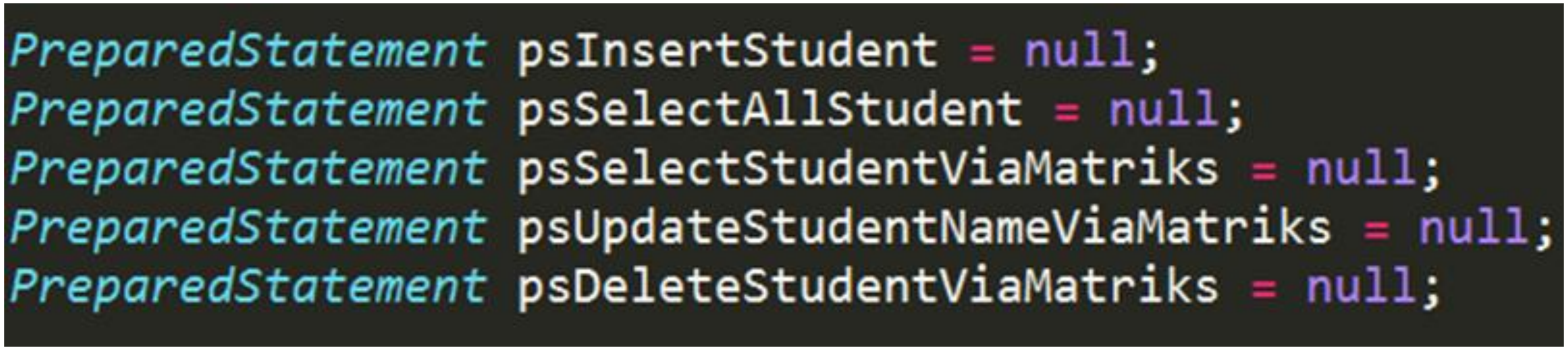
- A method for initiating connection to the database



```
public  void jdbcConnect()
{
   try
    {
      Class.forName (driver);
      con = DriverManager.getConnection(url, userName, password);
      DatabaseMetaData dma = con.getMetaData ();
      System.out.println("\nConnected to " + dma.getURL());
      System.out.println("Driver       " + dma.getDriverName());
      System.out.println("Version      " + dma.getDriverVersion());
      System.out.println("");
    }
    catch (SQLException ex)
    {
      while (ex != null)
      {
           System.out.println ("SQLState: " +
                               ex.getSQLState ());
         System.out.println ("Message:  " +
                               ex.getMessage ());
           System.out.println ("Vendor:   " +
                               ex.getErrorCode ());
         ex = ex.getNextException ();
           System.out.println ("");
      }

      System.out.println("Connection to the database error");
    }
    catch (java.lang.Exception ex)
    {
      ex.printStackTrace ();
    }
}
```

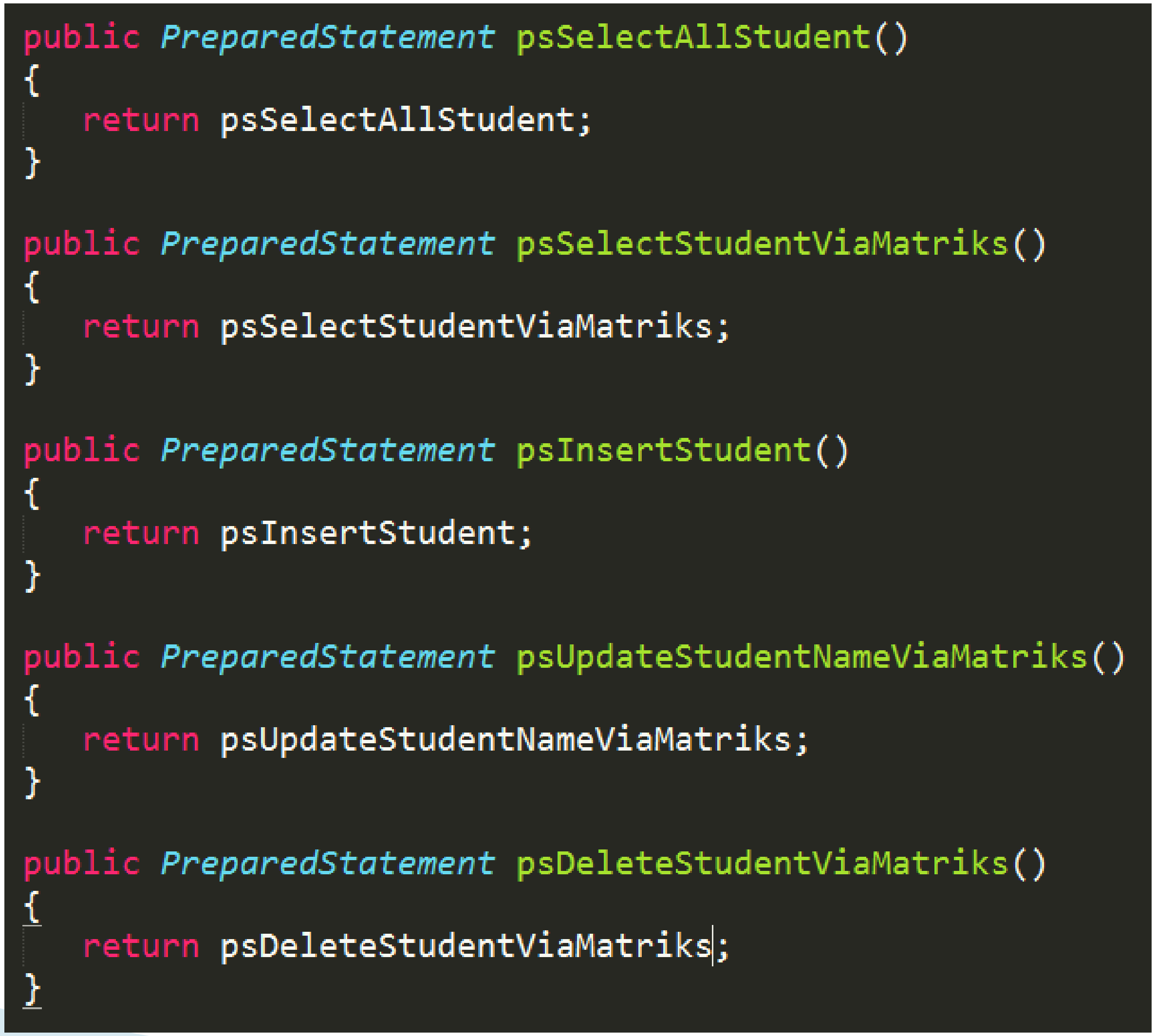
JDBC – initial data

- Create variables for representing all the prepared statements needed



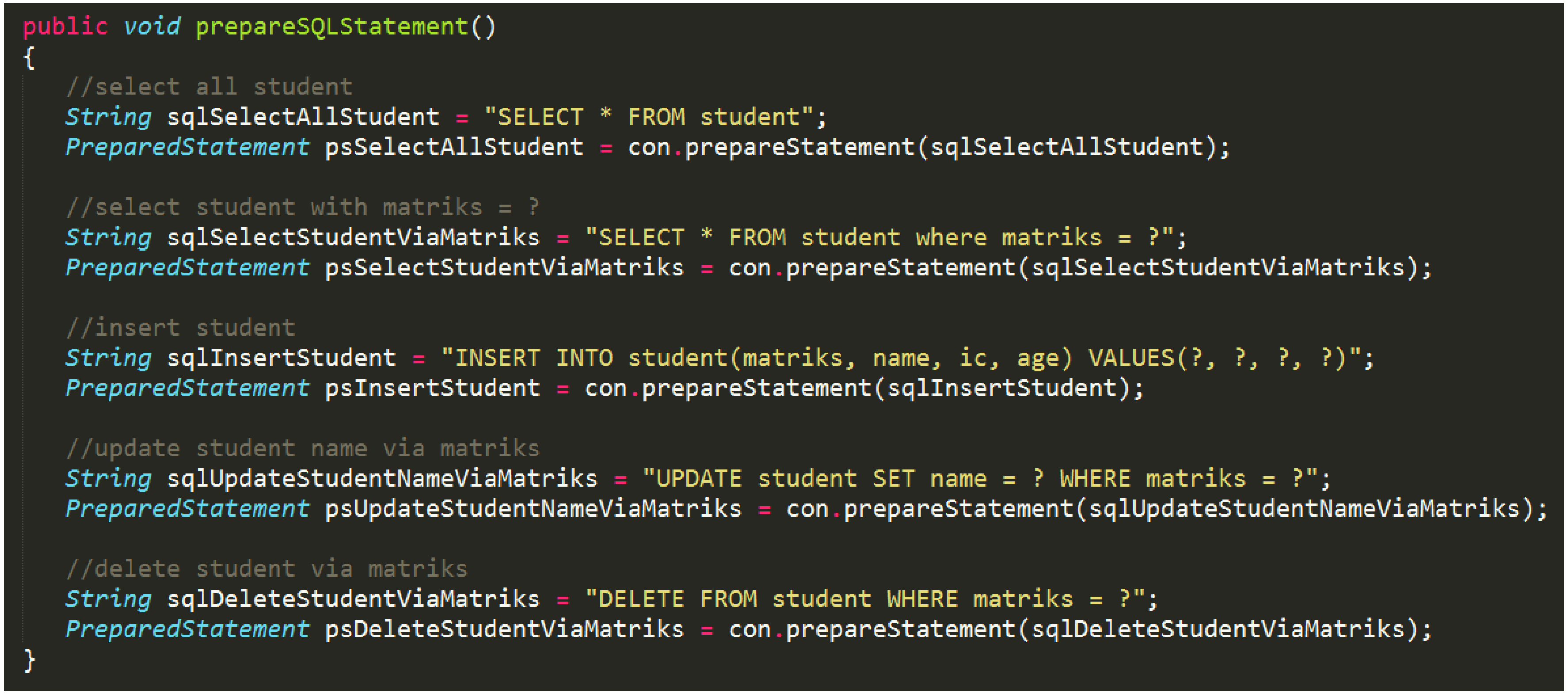
```
PreparedStatement psInsertStudent = null;
PreparedStatement psSelectAllStudent = null;
PreparedStatement psSelectStudentViaMatriks = null;
PreparedStatement psUpdateStudentNameViaMatriks = null;
PreparedStatement psDeleteStudentViaMatriks = null;
```

Prepared Statement – get method to use the statement in servlet for later



```
public PreparedStatement psSelectAllStudent()
{
    return psSelectAllStudent;
}

public PreparedStatement psSelectStudentViaMatriks()
{
    return psSelectStudentViaMatriks;
}

public PreparedStatement psInsertStudent()
{
    return psInsertStudent;
}

public PreparedStatement psUpdateStudentNameViaMatriks()
{
    return psUpdateStudentNameViaMatriks;
}

public PreparedStatement psDeleteStudentViaMatriks()
{
    return psDeleteStudentViaMatriks;
}
```

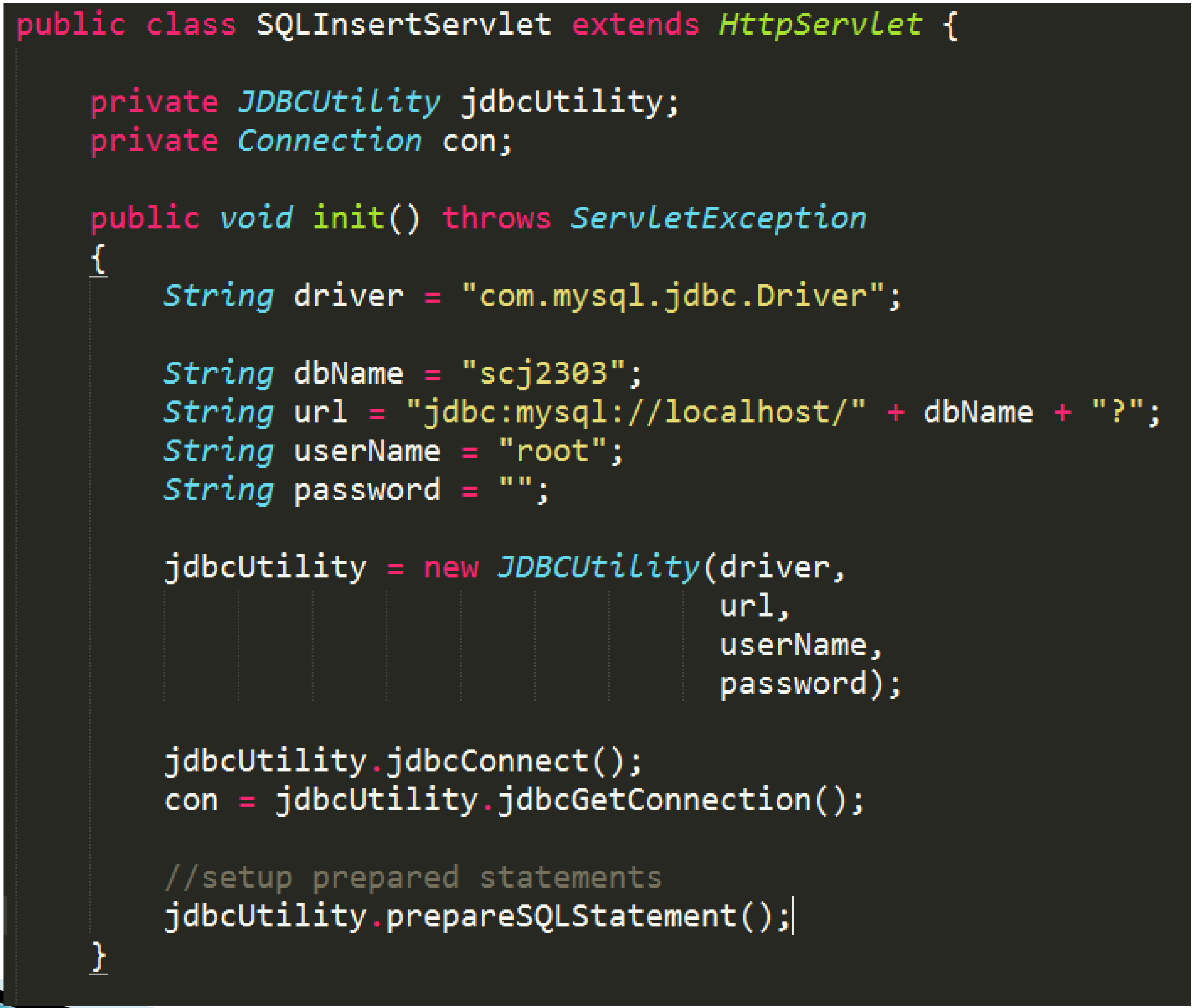
JDBC – method to execute the preparing statements operation – ONCE ONLY!



```java
public void prepareSQLStatement()
{
    //select all student
    String sqlSelectAllStudent = "SELECT * FROM student";
    PreparedStatement psSelectAllStudent = con.prepareStatement(sqlSelectAllStudent);

    //select student with matriks = ?
    String sqlSelectStudentViaMatriks = "SELECT * FROM student where matriks = ?";
    PreparedStatement psSelectStudentViaMatriks = con.prepareStatement(sqlSelectStudentViaMatriks);

    //insert student
    String sqlInsertStudent = "INSERT INTO student(matriks, name, ic, age) VALUES(?, ?, ?, ?)";
    PreparedStatement psInsertStudent = con.prepareStatement(sqlInsertStudent);

    //update student name via matriks
    String sqlUpdateStudentNameViaMatriks = "UPDATE student SET name = ? WHERE matriks = ?";
    PreparedStatement psUpdateStudentNameViaMatriks = con.prepareStatement(sqlUpdateStudentNameViaMatriks);

    //delete student via matriks
    String sqlDeleteStudentViaMatriks = "DELETE FROM student WHERE matriks = ?";
    PreparedStatement psDeleteStudentViaMatriks = con.prepareStatement(sqlDeleteStudentViaMatriks);
}
```

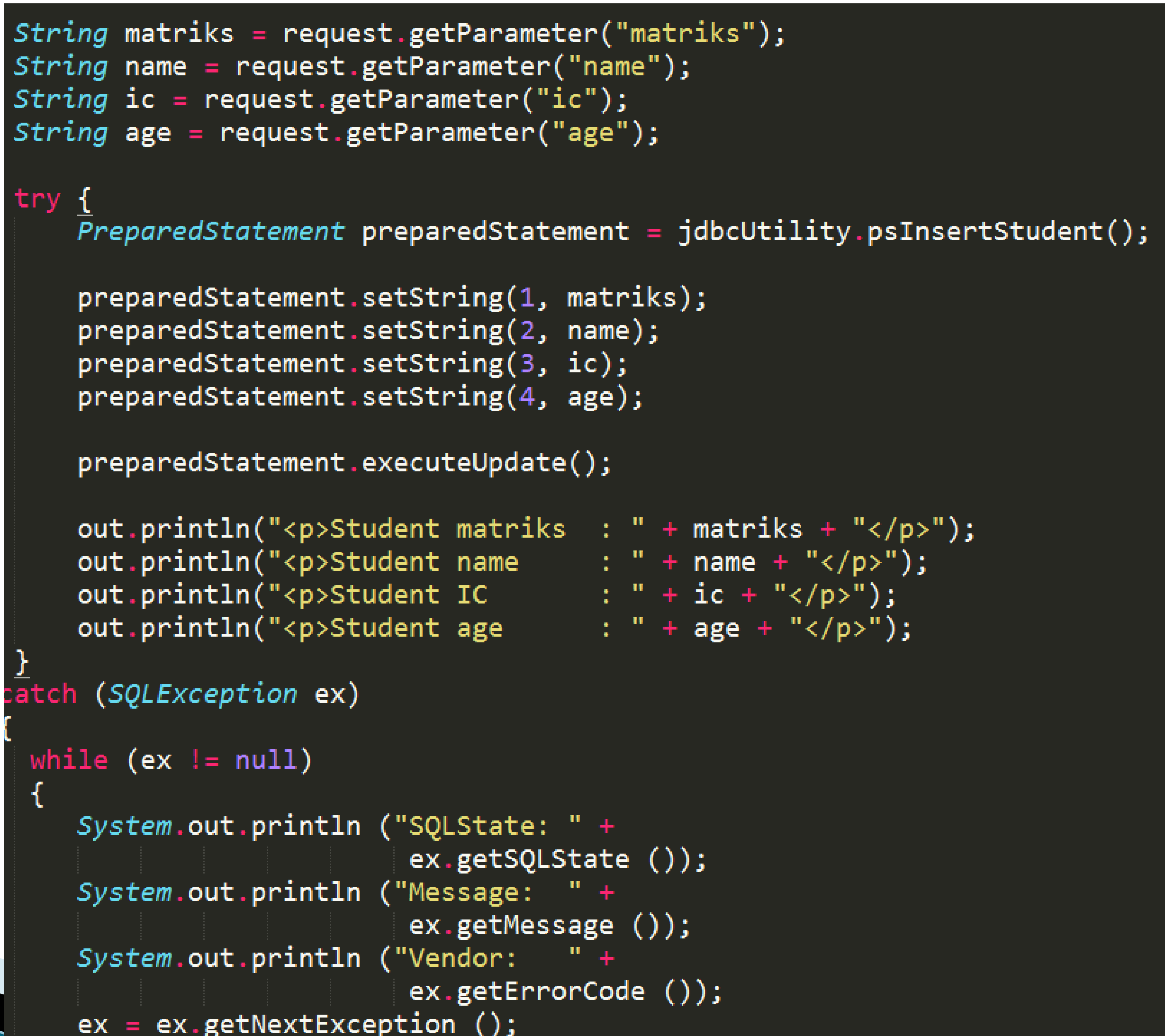
Using the JDBC in servlet

1. Create global variables in servlet
   - **private JDBCUtility jdbcUtility;**
   - **private Connection con;**
2. In **init method (THIS METHOD WILL BE CALLED ONCE ONLY – refer slide 11)**
   - Call the constructor to instantiate the JDBCUtility object
   - Call the connect method to initiate connection to the database
     - **jdbcUtility.jdbcConnect();**
   - Get a working connection
     - **con = jdbcUtility.jdbcGetConnection();**
   - Execute the preparing of statements operation
     - **jdbcUtility.prepareSQLStatement();**



```
public class SQLInsertServlet extends HttpServlet {

    private JDBCUtility jdbcUtility;
    private Connection con;

    public void init() throws ServletException
    {
        String driver = "com.mysql.jdbc.Driver";

        String dbName = "scj2303";
        String url = "jdbc:mysql://localhost/" + dbName + "?";
        String userName = "root";
        String password = "";

        jdbcUtility = new JDBCUtility(driver,
                                      url,
                                      userName,
                                      password);

        jdbcUtility.jdbcConnect();
        con = jdbcUtility.jdbcGetConnection();

        //setup prepared statements
        jdbcUtility.prepareSQLStatement();
    }
```

Get the prepared statement needed, fill the data and execute



```
String matriks = request.getParameter("matriks");
String name = request.getParameter("name");
String ic = request.getParameter("ic");
String age = request.getParameter("age");

try {
    PreparedStatement preparedStatement = jdbcUtility.psInsertStudent();

    preparedStatement.setString(1, matriks);
    preparedStatement.setString(2, name);
    preparedStatement.setString(3, ic);
    preparedStatement.setString(4, age);

    preparedStatement.executeUpdate();

    out.println("<p>Student matriks  : " + matriks + "</p>");
    out.println("<p>Student name     : " + name + "</p>");
    out.println("<p>Student IC       : " + ic + "</p>");
    out.println("<p>Student age      : " + age + "</p>");
}
catch (SQLException ex)
{
 while (ex != null)
 {
    System.out.println ("SQLState: " +
                         ex.getSQLState ());
    System.out.println ("Message:  " +
                         ex.getMessage ());
    System.out.println ("Vendor:   " +
                         ex.getErrorCode ());
    ex = ex.getNextException ();
```

JDBC project

- INSERT record
  - Run the project – (running the index.html)
    - Will display the form
    - Form action directed to **DBInsertStudent** servlet
- SELECT all record (view in table)
  - Run the servlet: **DBSelect**
- UPDATE record
  - Must select the record first using the primary key
    - Servlet name: **DBUpdateForm**
    - GET method data: matriks
  - Put the record into form
  - Form action directed to **DBUpdateStudent** servlet

JDBC project

- DELETE record
  - Servlet name: **DBDeleteStudentViaMatriks**
  - GET method data: matriks
- DBDataTable
  - CRUD operations
    - C-reate – SQL INSERT operation (**DBInsertStudent** servlet)
    - R-ead – SQL SELECT operation (**DBSelect** servlet**)**
    - U-pdate – SQL UPDATE operation (**DBUpdateStudent** servlet**)**
    - D-elete – SQL DELETE operation (**DBDeleteStudentViaMatriks** servlet**)**
  - All operations in one single page

Persistent State in HTTP Servlets

- HTTP transactions are made in isolation of one another
  - do not have a mechanism for keeping track of a request or request data sent using a web browser
  - said to be "stateless"
- **Benefit**
  - Client browsers do not notice when a server goes down and comes up quickly
- **Drawback**
  - difficult to produce groups of pages for collecting information to produce picture of the user's web experience

Session tracking methods

- **Cookies (netbeans example: cookies / cookiesshare**
  - small size of information left by the server at client machine (in browser cookies repository)
  - **misinformation about cookies**
    - Never interpreted or executed
    - browsers generally only accept 20 cookies per site
    - and 300 cookies and limited to 4 kilobytes per size
    - cannot be used to fill up someone's disk or launch other denial of service attack
  - problem
    - user disable browser cookies
    - to protect privacy

Session tracking methods

- **URL Rewriting.**
  - append some extra data on the end of each URL that identifies the session, and the server associate that session identifier with data it has stored about that session.
  - Excellent solution with browsers that don't support cookies or where the user has disabled cookies.
  - However, it has most of the same problems as cookies, namely that the server-side program has a lot of straightforward but tedious processing to do.
  - In addition, you have to be very careful if the user leaves the session and comes back via a bookmark or link, the session information can be lost.

Session tracking methods

- **Hidden form fields.**
  - HTML forms have an entry that looks like the following:

<INPUT TYPE="HIDDEN" NAME="session" VALUE="...">

  - This means that, when the form is submitted, the specified name and value are included in the GET or POST data.
  - This can be used to store information about the session.
  - However, it has the major disadvantage that it only works if every page is dynamically generated, since the whole point is that each session has a unique identifier.

  - Netbeans example: hidden

Session management in Servlet

- **HttpSession API.**
  - high-level interface built on top of cookies or URL-rewriting.
  - use cookies if the browser supports them,
  - automatically revert to URL-rewriting when cookies are unsupported or explicitly disabled.
  - servlet author doesn't need to bother with many of the details,
    - doesn't have to explicitly manipulate cookies
    - or information appended to the URL,
    - automatically given a convenient place to store data that is associated with each session.

HttpSession API

- Provides a way to identify a user across more than one page request
- create a session between an HTTP client and an HTTP server.
- session persists for a specified time period, across more than one connection or page request from the user.
- usually corresponds to one user
- allows servlet to
  - View and manipulate information about a session, such as the **session identifier**, **creation time**, and **last accessed time**
  - **Bind objects** to sessions, allowing user information to persist across multiple user connections

HttpSession API

- Looking up the session object associated with the current request,
  - `HttpSession session = request.getSession();`
    - Returns the current session associated with this request, or if the request does not have a session, creates one.
  - `HttpSession session = request.getSession(boolean param);`
    - **param=true** – to create a new session for this request if necessary;
    - **param=false** to return null if there's no current session

HttpSession API

- Binds an object to this session, using the name specified.
- If an object of the same name is already bound to the session, the object is replaced
  - **void setAttribute(java.lang.String name, java.lang.Object value)**
- Returns the object bound with the specified name in this session, or null if no object is bound under the name
  - **java.lang.Object getAttribute(java.lang.String name)**

HttpSession API

- Specifies the time, in seconds, between client requests before the servlet container will invalidate this session. A negative time indicates the session should never timeout.
  - **void setMaxInactiveInterval(int interval)**
    - **interval** in seconds
    - default 30 minutes
- Invalidates this session and unbinds any objects bound to it (remove current session)
  - **void invalidate()**