Mortgages Calculator

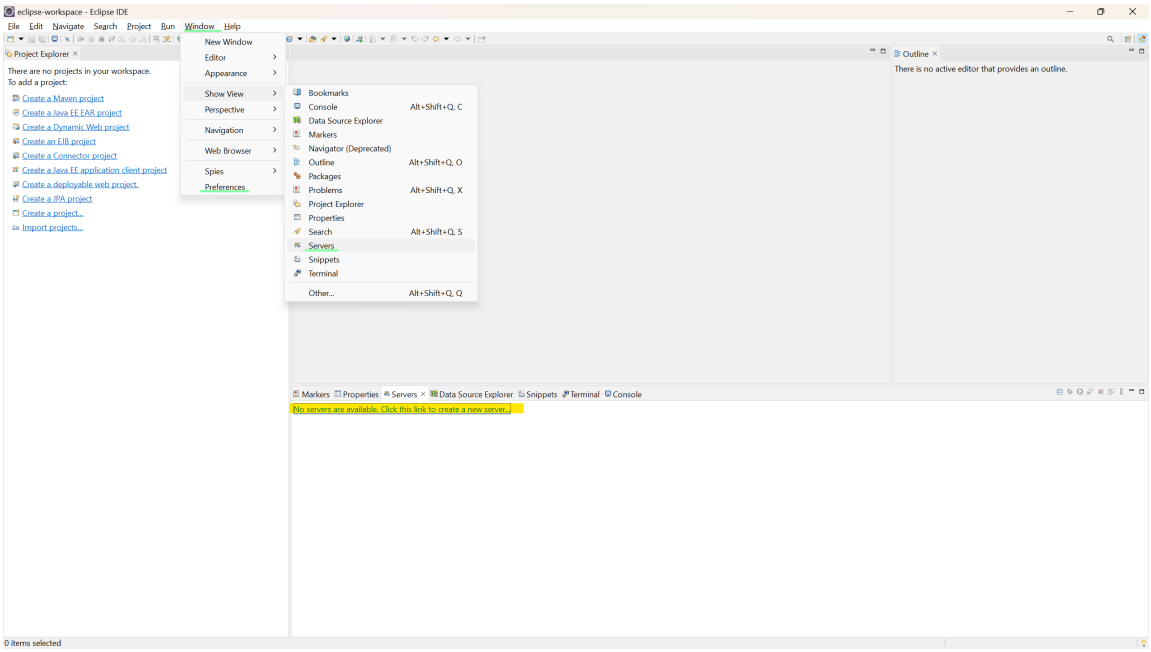
This is a dynamic web project for calculating mortgages, developed using Servlets and MySql. It was created as a web application deployment assignment for DAW dual.

Development Environment

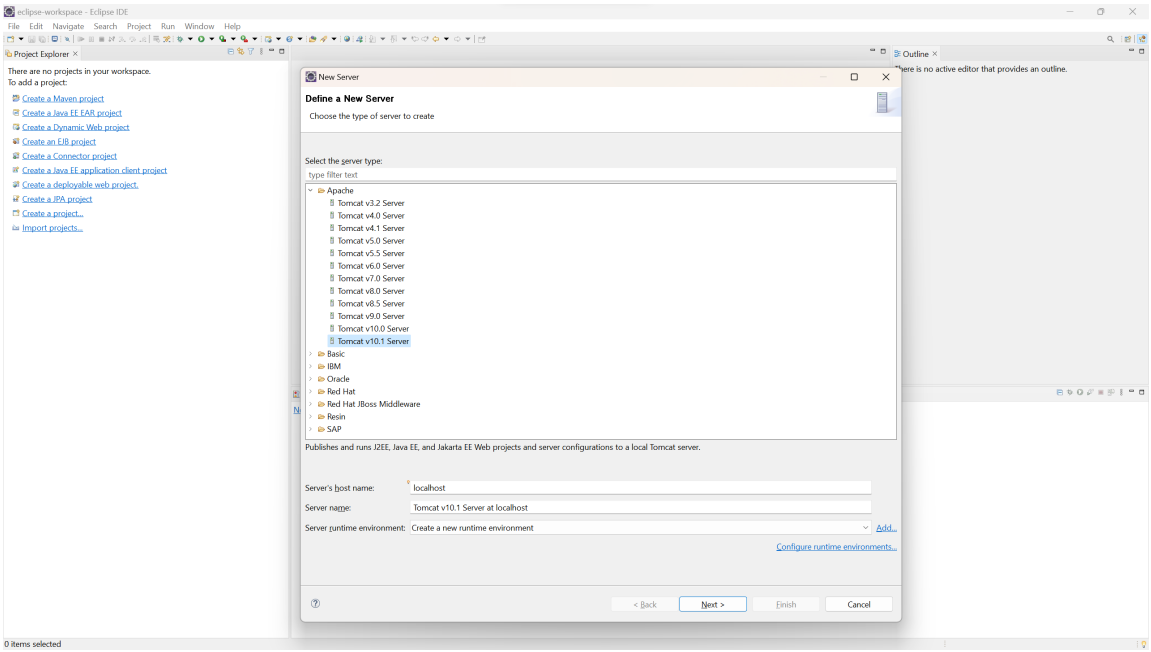
- [ ] Eclipse IDE for Enterprise Java and Web Developers
- [ ] JDBC MySQL Connector jar 8.0.32
- [ ] MySQL installer Community (developer option or full | Tutorial recommended)
- [ ] Apache tomcat 10 (Core's zip)

How to configure Tomcat in Eclipse?

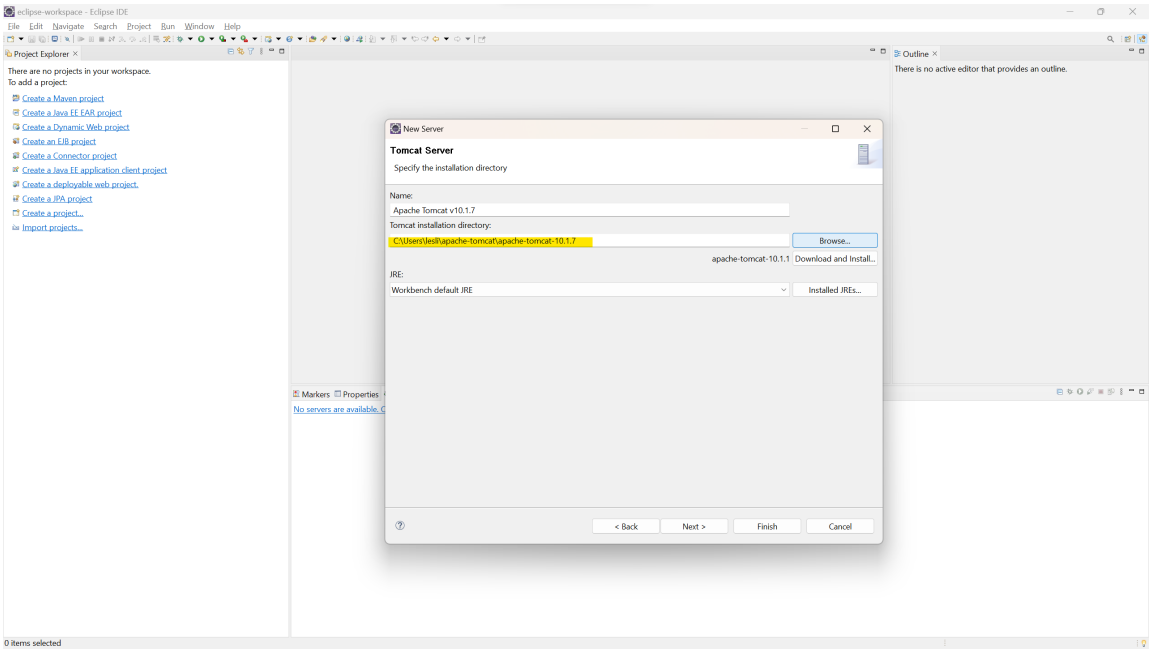
1. Open Eclipse IDE and click on *“Click this link to create a new Server” (yellow).* If you dont see it, go to *Window>Show view>Servers (green)*



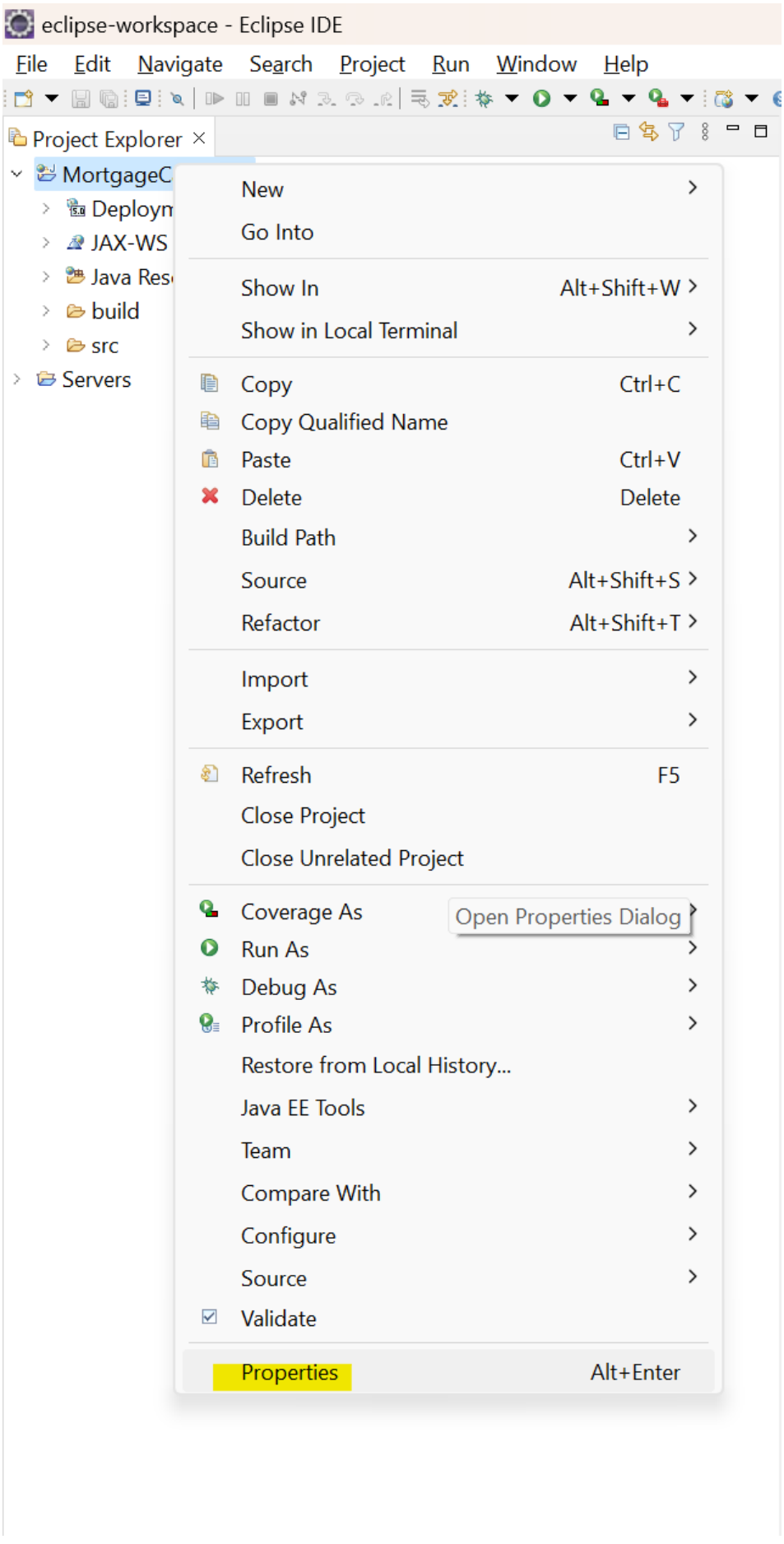
2. Select the latest Tomcat available, then Next



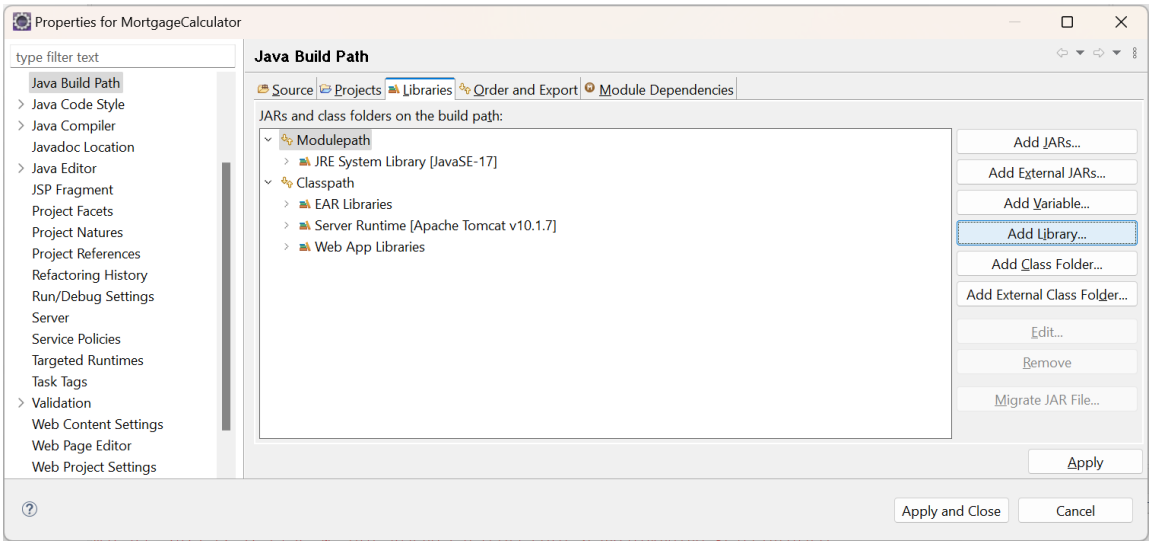
3. Afterward, click on the "Browse" button and select the path where Tomcat is installed. Finally, click on the "Finish" button.



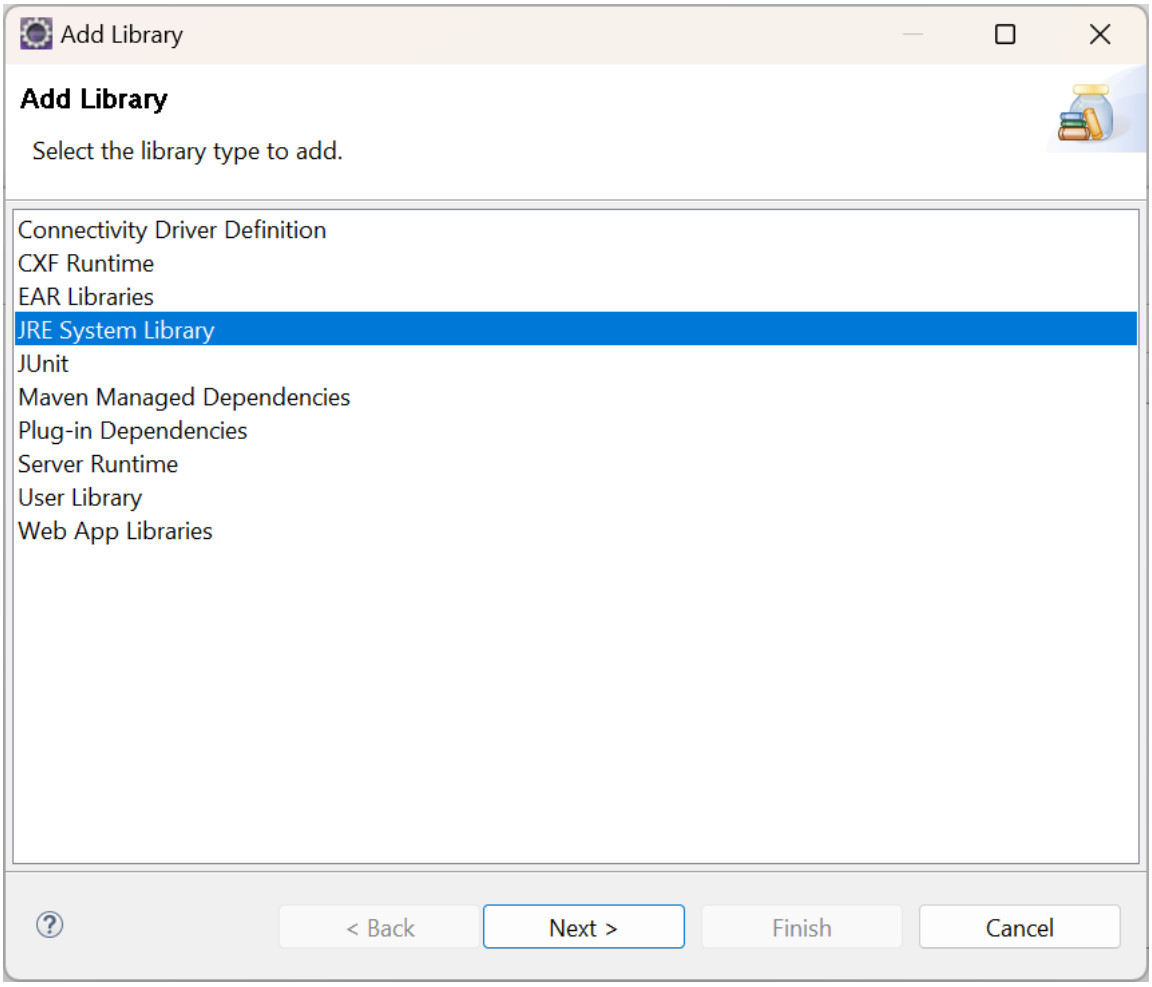
4. To create a dynamic web project for the MortgageCalculator activity, go to **File > New > Dynamic Web Project**. In the configuration window, select the previous Tomcat server.
5. Right-click on the MortgageCalculator project and select Properties.


eclipse-workspace - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Project Explorer
MortgageC
Deployn
JAX-WS
Java Res
build
src
Servers
New
Go Into
Show In Alt+Shift+W
Show in Local Terminal
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import
Export
Refresh F5
Close Project
Close Unrelated Project
Coverage As
Open Properties Dialog
Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
Team
Compare With
Configure
Source
Validate
Properties Alt+Enter

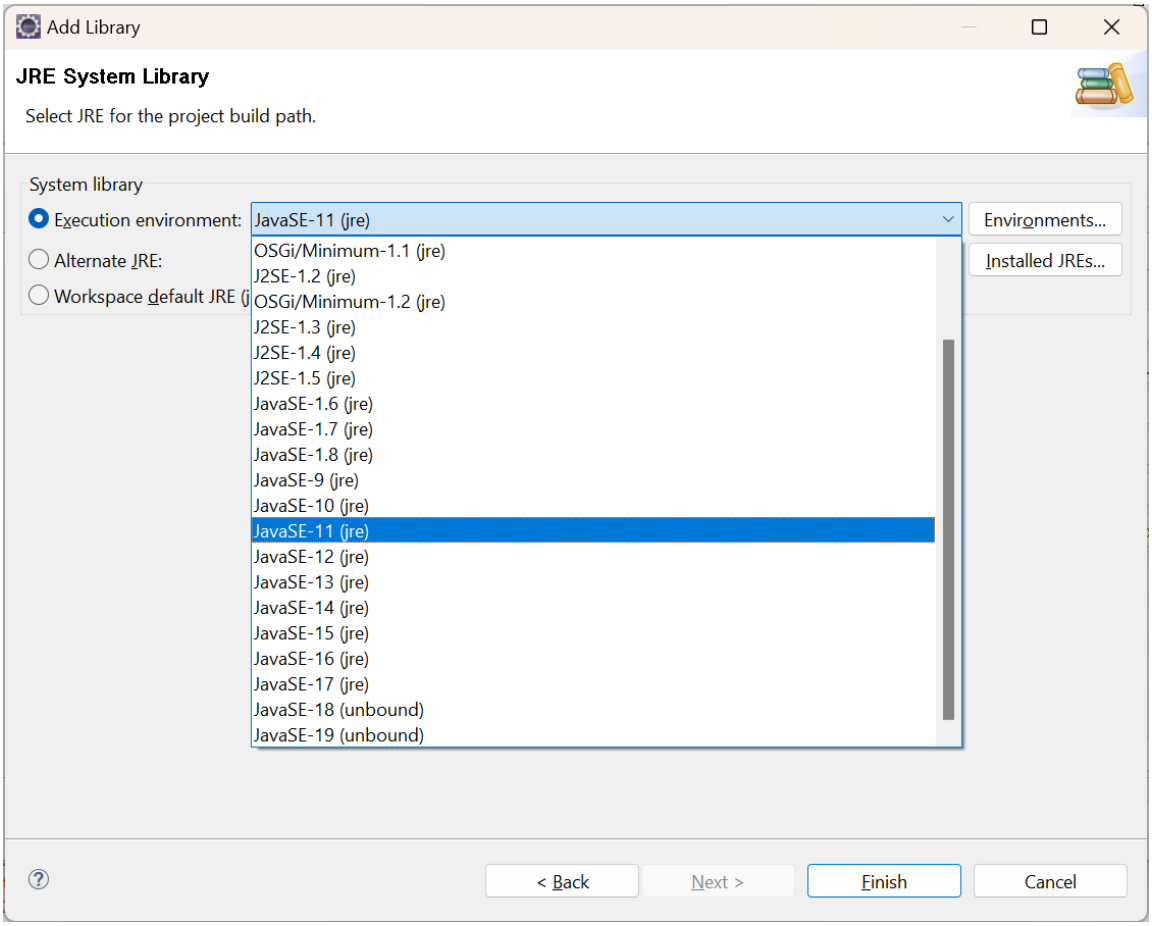
6. Navigate to Java Build Path > Libraries, click on "Module Path", and then select "Add Library".



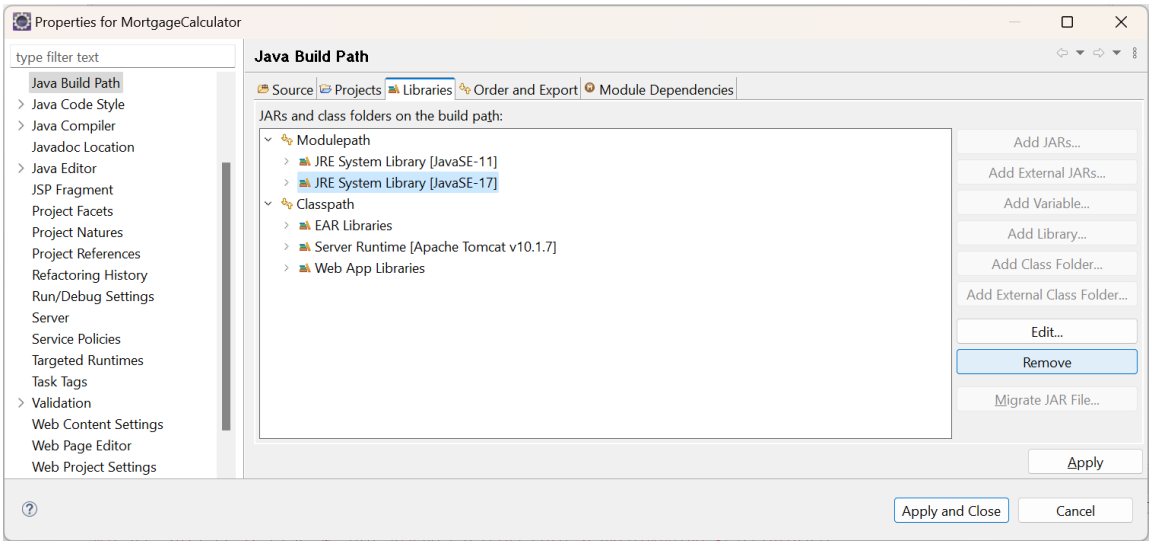
7. Click on "JRE System Library", then click "Next".



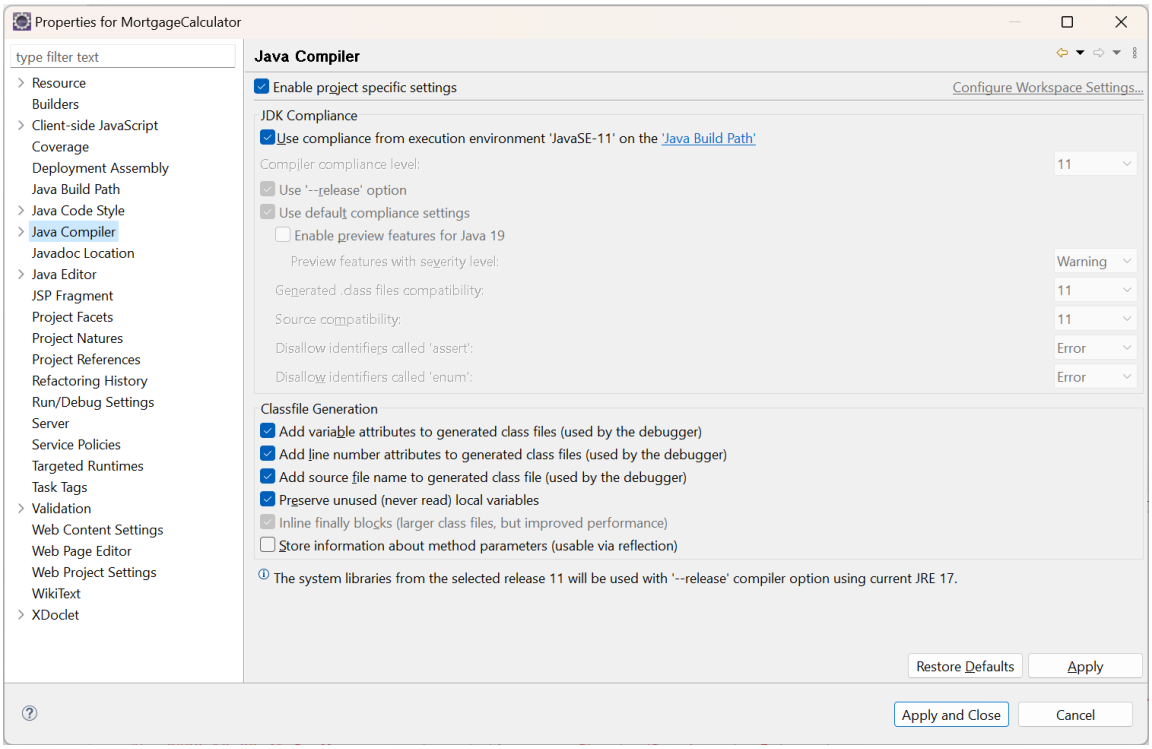
8. Select the "Execution Environment" option, and choose "JavaSE-11" from the dropdown menu. Finally, click "Finish."



9. Remove the previous JavaSE-17 and click on "apply".



10. Check the Java Compiler section to ensure that the version is set to Java 11. After that, click on ***"Apply and Close"*** button.



11. For windows systems, navigate to ***"Environment Variables"*** and add:
    - **JAVA_HOME:** path where the JDK 11 is installed
    - Edit the **PATH** and add %JAVA_HOME%\bin
    - **CATALINA_HOME:** parth where the Tomcat 10 is installed
12. Restart the computer.

How to do a database connection from JDBC?

▼ What is JDBC?

Java Database Connectivity (JDBC) is an API that allows access to relational databases in a standard way. Each database vendor provides the specific JDCB driver to allow connection to the database.

A **JDBC drive** or controller is an implementation of the JDBC API for a specific database management system (Oracle, MySQL, PostgreSQL). There are 4 types of drives:

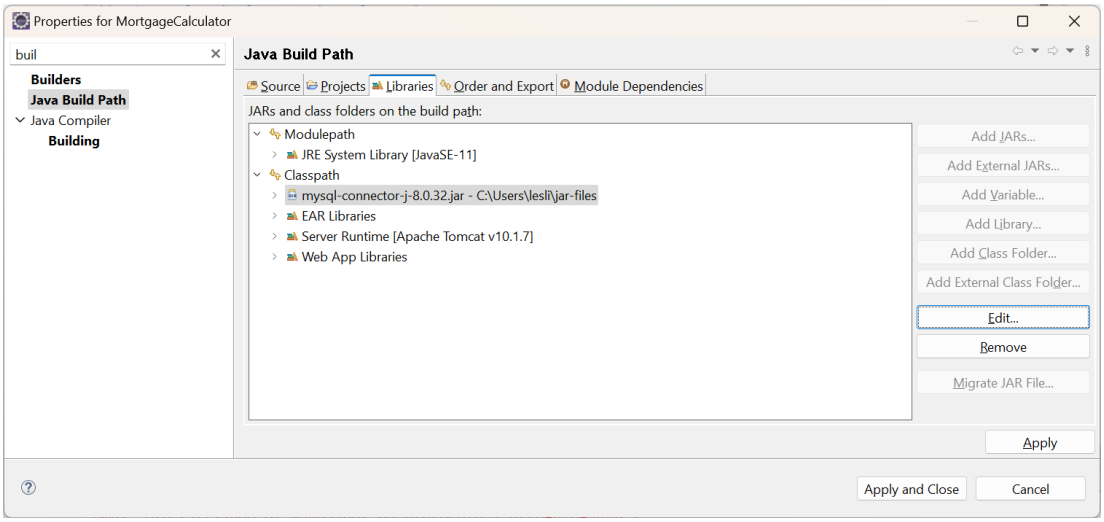
- JDBC-ODBC
- Native JDBC-API

- JDBC 100% Java in network
- 100% Java

▼ Steps:

The connection to Java databases through JDBC is done in 4 steps:

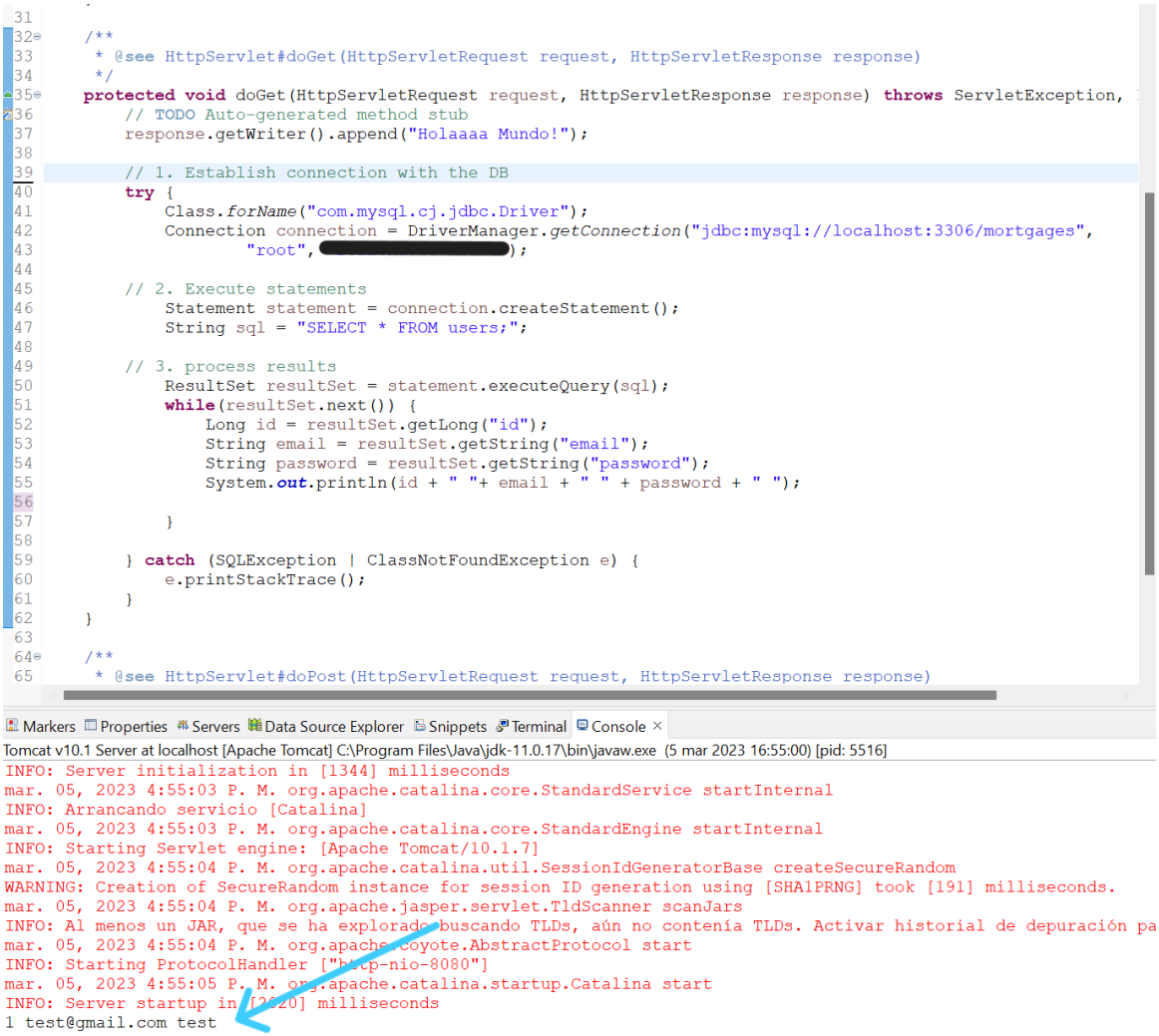
1. Right-click on the MortgageCalculator project and select Properties. Navigate to "***Java Build Path"> "Libraries" >*** click on ***"Class path"*** > then ***"Add External Jars.." >*** select the path where the **jar** is installed. > ***"Apply and Close"***



> *Recomended video tutorial →*
> https://www.youtube.com/watch?v=lX0FhmpHGrc

2. Establish a connection
3. Execute SQL statement
4. Process results: mostly this procedure consists of mapping the columns with attributes of java objects

Example code in a servlet.



```
31
32  /**
33   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
34   */
35  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
36      // TODO Auto-generated method stub
37      response.getWriter().append("Holaaaa Mundo!");
38
39      // 1. Establish connection with the DB
40      try {
41          Class.forName("com.mysql.cj.jdbc.Driver");
42          Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/mortgages",
43                  "root", [redacted]);
44
45      // 2. Execute statements
46          Statement statement = connection.createStatement();
47          String sql = "SELECT * FROM users;";
48
49      // 3. process results
50          ResultSet resultSet = statement.executeQuery(sql);
51          while(resultSet.next()) {
52              Long id = resultSet.getLong("id");
53              String email = resultSet.getString("email");
54              String password = resultSet.getString("password");
55              System.out.println(id + " "+ email + " " + password + " ");
56
57          }
58
59      } catch (SQLException | ClassNotFoundException e) {
60          e.printStackTrace();
61      }
62  }
63
64  /**
65   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
```

```
Markers  Properties  Servers  Data Source Explorer  Snippets  Terminal  Console
Tomcat v10.1 Server at localhost [Apache Tomcat] C:\Program Files\Java\jdk-11.0.17\bin\javaw.exe  (5 mar 2023 16:55:00) [pid: 5516]
INFO: Server initialization in [1344] milliseconds
mar. 05, 2023 4:55:03 P. M. org.apache.catalina.core.StandardService startInternal
INFO: Arrancando servicio [Catalina]
mar. 05, 2023 4:55:03 P. M. org.apache.catalina.core.StandardEngine startInternal
INFO: Starting Servlet engine: [Apache Tomcat/10.1.7]
mar. 05, 2023 4:55:04 P. M. org.apache.catalina.util.SessionIdGeneratorBase createSecureRandom
WARNING: Creation of SecureRandom instance for session ID generation using [SHA1PRNG] took [191] milliseconds.
mar. 05, 2023 4:55:04 P. M. org.apache.jasper.servlet.TldScanner scanJars
INFO: Al menos un JAR, que se ha explorado buscando TLDs, aún no contenía TLDs. Activar historial de depuración pa
mar. 05, 2023 4:55:04 P. M. org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-8080"]
mar. 05, 2023 4:55:05 P. M. org.apache.catalina.startup.Catalina start
INFO: Server startup in [2020] milliseconds
1 test@gmail.com test
```

DAO design pattern

The DAO pattern (Data Access Object) is a design pattern that allows for separation of concerns between the application logic and the persistence layer. This is achieved by abstracting away the details of the database implementation behind a common interface (the DAO interface), which declares the methods necessary for performing database operations.

By using the DAO pattern, the application can interact with the persistence layer through a set of standard interfaces, rather than directly with the underlying database. This promotes loose coupling between the application and the database, making it easier to change the database implementation or switch to a different database entirely without having to modify the application code.

As mentioned, the implementation of the DAO interface involves overriding each CRUD method (create, read, update, delete) and following a standard 3-step scheme. The first step is to establish a connection to the database, the second step is to execute the appropriate SQL statement, and the third step is to process the results of the query. This implementation ensures that the code for interacting with the database is consistent across the application, and that any changes to the database schema can be easily accommodated in the DAO implementation.

How Servlets work?

Servlets are Java classes that extend the functionality of web servers by processing HTTP requests and generating responses. When a user sends a request to a web server, such as submitting a form or clicking a link, the web server forwards the request to a servlet that is configured to handle that particular URL pattern.

In the Jakarta CLass there are two main methods:

1. **doGet()** →is invoked every time an HTTP GET request is received at the URL for which the servlet is registered to handle or listen. A servlet can redirect to another component or return any type of response, such as
    a. JSON :

```
response.setContentType("aplication/json");
response.getWriter().println("{\"message\": \"Hola Mundo! JSON ej.\"}");
```

    b. HTML :

```
response.setContentType("text/html");
response.getWriter().println("<h1>Hola Mundo</h1>");
```

2. **doPost()** → This method is triggered every time an HTTP POST request arrives at the URL that the servlet is listening to. A servlet can extract the data that was sent in the request. This data is usually transmitted to the server through HTML forms using the HTTP POST method, with the 'action' attribute in the form tag pointing to the URL that the servlet is listening to

```
<div class="form-inner">
   <form action="http://localhost:8080/MortgageCalculator/login" method= "POST" class="login">
      <div class="field">
         <input type="text" placeholder="Email Address" name="email" id="email" required>
      </div>
      <div class="field">
         <input type="password" placeholder="Password" name="password" id="password" required>
      </div>
      <div class="pass-link">
         <a href="#">Forgot password?</a>
      </div>
      <div class="field btn">
         <div class="btn-layer"></div>
         <input type="submit" value="Login">
      </div>
   </form>
```

How to connect Servlets with DAO?

One option is to add a services layer, which acts as an intermediary between the Servlets and the DAOs. This layer provides an additional level of abstraction, allowing for greater flexibility and separation of concerns.

Implementing the View Layer with JSP/ JSTL in Java Web Applications

JSP pages are files with the **.jsp** extension that contain html and xml tags and embedded java code

They are created inside the webapp folder. When the web container receives aHTTP request to a JSP uses a JSP engine to convert internally the JSP to servlet and process the request. The JSP is one more layer of abstraction and they serve to be able to work better with html.

JSP tags allow you to use java code:

- Directives: <%@ … %>

- Java code: <% … %>
- Comments: <% // … %>
- Declaration: <%! … %>
- Expressions: <%= … %>

Within the JSP it is possible to access the request and other implicit objects, such as:

- HttpServletRequest request
- HttpServletResponse response
- HttpSession session