JavaEE Technologies

Student Guide

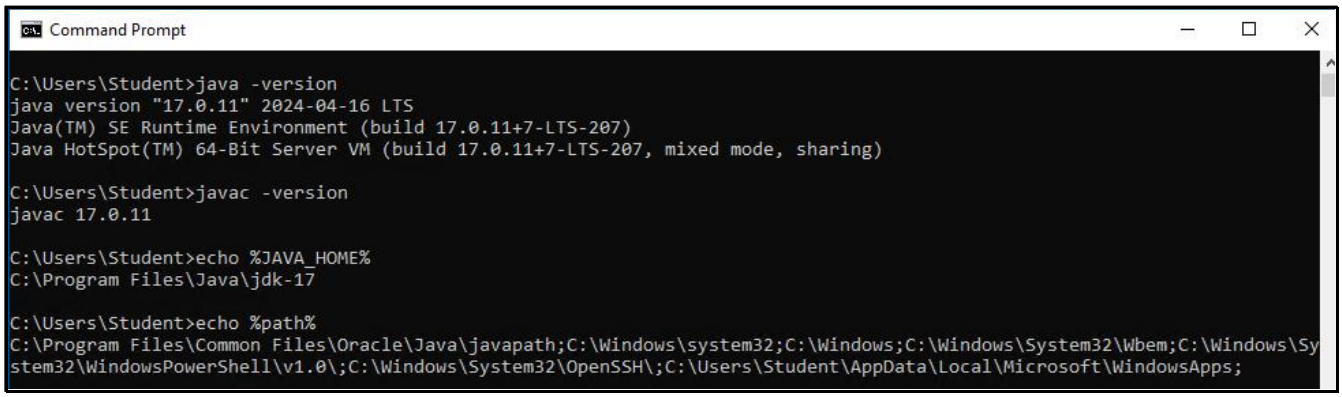
Prepared by Łukasz Laszko

Lab.1

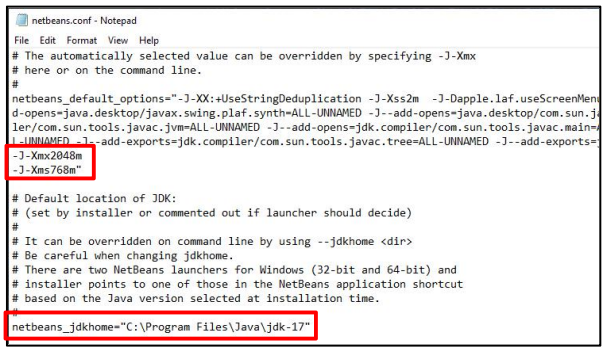
Topic scope: Building a JavaSE application with GUI
for database management using H2DB, Maven, JUnit and Git in NetBeans IDE

Part 1: Setting up the environment

1. **Verify required tools:**
   1.1. Java configuration:
      a) cmd → java -version
      b) cmd → javac -version
      c) cmd → echo %JAVA_HOME%
      d) cmd → echo %path%



   1.2. NetBeans IDE:
      a) netbeans.conf



This file should be in the following directory:

C:\Program Files\NetBeans-22\netbeans\etc\

      b) Tools → Java Platforms

Check if the property is set correctly

      c) Team → Git

Check if this functionality is available (not grayed out)

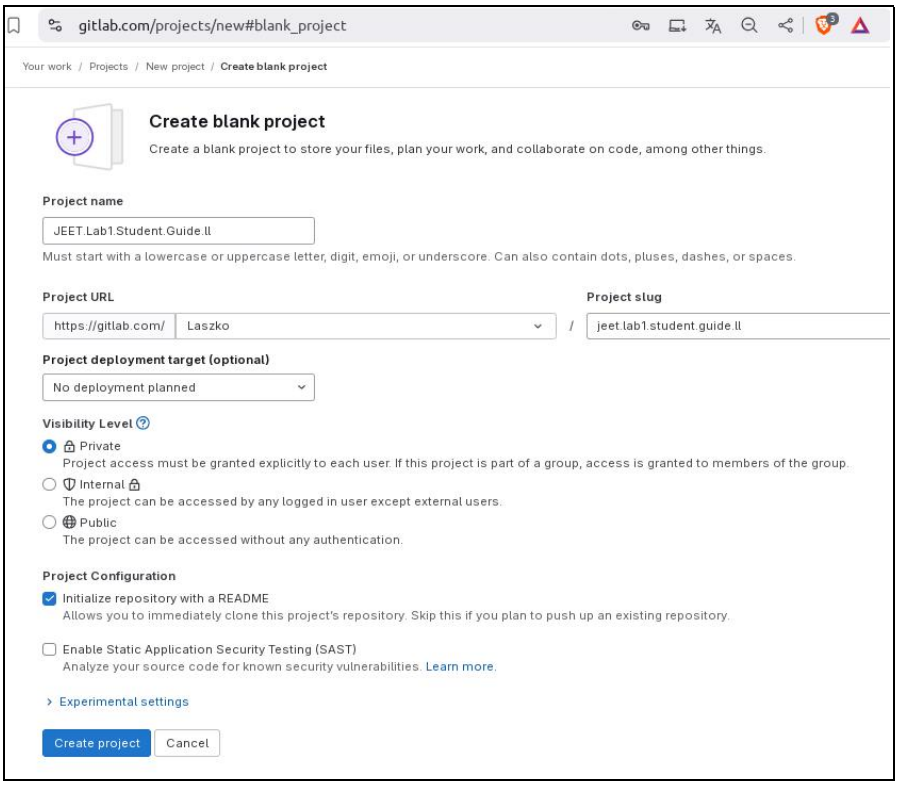
d) Tools → Plugins → Installed

Check if JavaSE is set to active and if Maven feature is supported

1.3. Git repository:

a) Create an account and login to public Git repository (e.g. GitLab.com)
- https://gitlab.com/users/sign_in
- Create new blank project of private visibility.

Consider adding your initials to the project name (e.g. LL)



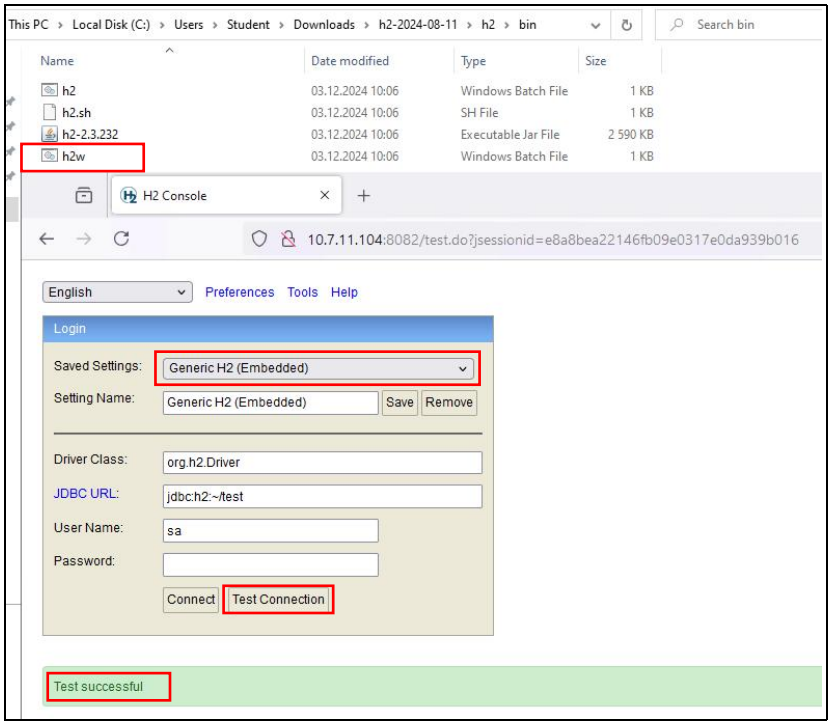
b) Save the link to the project for future use:
- e.g. Code → Clone with HTTPS:
  https://gitlab.com/Laszko/jeet.lab1.Student.Guide.LL.git

1.4. H2DB database management system availability:

Note that H2DB DBMS can be used in a Java project both via Maven and via explicit installation.
The latter option will be discussed below

a) Download H2DB binaries from (*zip archive is suggested*):
https://h2database.com/html/download.html

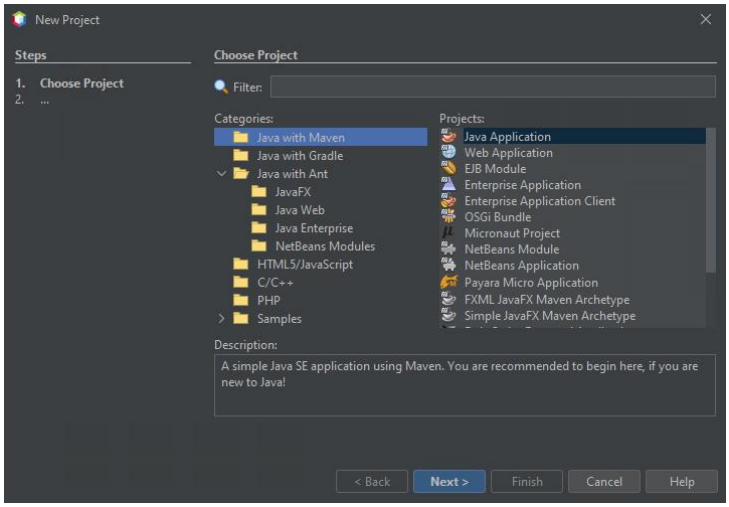
b) Extact and test-run it using the web client
c) Optionally create test database



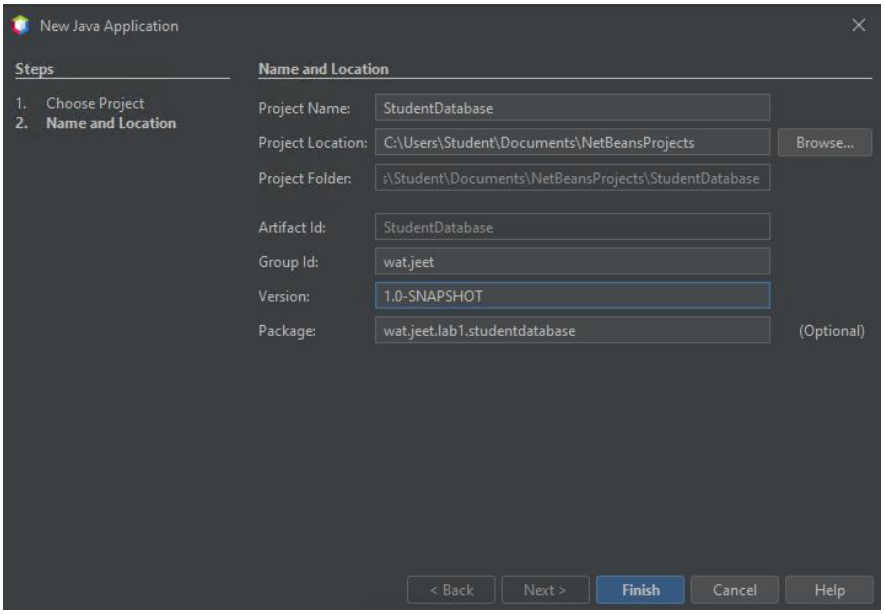
2. Preliminary application Implementation:

2.1. Create new project:

a) File → New Project → Java with Maven → Java Application

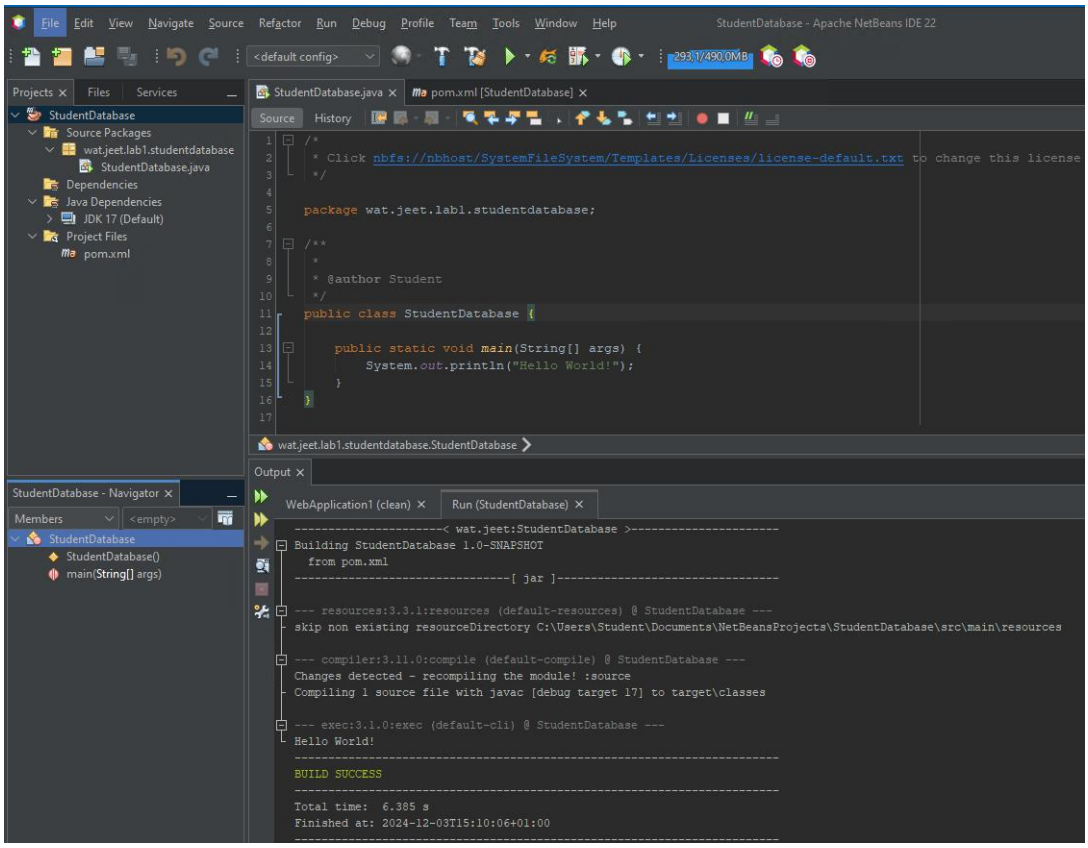


2.2. Create a basic project configuration. Decide on your own or similarly to the image below.



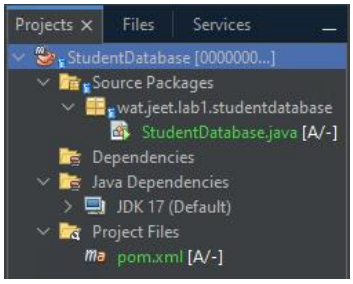
2.3. Review the project structure and in particular the Maven project build dependency diagram (pom.xml)

2.4. Test-run generated template code using Run Project function (or F6, Run → Run Project)



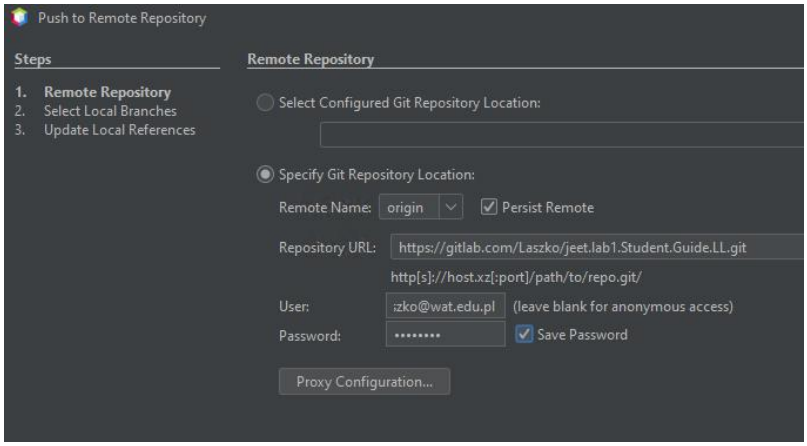
2.5. Initialize Git repository

a) Using the project context menu go to versioning and choose to create a local Git repository (Versioning → Initialize Git Repository ...)
b) Observe the changes to project structure
c) Add project directory to change tracking list (Git → Add)



d) Do first commit to the local repository (Git → Commit)
e) Send initial commit to remote Git repository (Git → Remote → Push...)

Consider disabling destructive actions



f) check whether changes have been uploaded

3. Implement data access:

The tasks below include sample code and independent work by students on the application. It is suggested to write your own code on the basis of the samples

3.1. Database Initialization

a) Create a helper class to set up and initialize the H2 database

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class DatabaseInitializer {
    private static final String DB_URL = "jdbc:h2:~/testdb";

    public static void initialize() {
        try (Connection connection = DriverManager.getConnection(DB_URL, "sa", "");
             Statement statement = connection.createStatement()) {
            String createTableSQL = """
                CREATE TABLE IF NOT EXISTS users (
                    id BIGINT AUTO_INCREMENT PRIMARY KEY,
                    name VARCHAR(255) NOT NULL,
                    email VARCHAR(255) NOT NULL
                )
            """;
            statement.execute(createTableSQL);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3.2. DAO Layer

a) Create a UserDAO class to handle database operations using JDBC

b) Think of adding and getting records, rather than all CRUD methods

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class UserDAO {
    private static final String DB_URL = "jdbc:h2:~/testdb";

    public void addUser(String name, String email) {
        String sql = "INSERT INTO users (name, email) VALUES (?, ?)";
        try (Connection connection = DriverManager.getConnection(DB_URL, "sa",
"");
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setString(1, name);
            statement.setString(2, email);
            statement.executeUpdate();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public List<String> getAllUsers() {
        List<String> users = new ArrayList<>();
        String sql = "SELECT * FROM users";
```

```
        try (Connection connection = DriverManager.getConnection(DB_URL, "sa",
"");
             Statement statement = connection.createStatement();
             ResultSet resultSet = statement.executeQuery(sql)) {
            while (resultSet.next()) {
                String user = resultSet.getString("name") + " - " +
resultSet.getString("email");
                users.add(user);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return users;
    }
}
```

3.3. Write a GUI using Swing

a) Create a Swing-based user interface for adding and viewing users.

b) Think of a simple GUI with text fields for getting data and a panel for presenting data from DB, as well as a appropriate buttons.

```
import javax.swing.*;
import java.awt.*;
import java.util.List;

public class UserApp {
    private final UserDAO userDAO;

    public UserApp(UserDAO userDAO) {
        this.userDAO = userDAO;
    }

    public void launch() {
        JFrame frame = new JFrame("User Management");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);

        JTextField nameField = new JTextField(20);
        JTextField emailField = new JTextField(20);
        JButton addButton = new JButton("Add User");
        JTextArea userList = new JTextArea(10, 30);

        addButton.addActionListener(e -> {
            String name = nameField.getText();
            String email = emailField.getText();
            userDAO.addUser(name, email);
            updateUserList(userList);
        });

        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());
        panel.add(nameField, BorderLayout.NORTH);
        panel.add(emailField, BorderLayout.CENTER);
        panel.add(addButton, BorderLayout.SOUTH);
```

```
        frame.getContentPane().add(panel, BorderLayout.NORTH);
        frame.getContentPane().add(new JScrollPane(userList), BorderLayout.CENTER);

        frame.setVisible(true);
    }

    private void updateUserList(JTextArea userList) {
        List<String> users = userDAO.getAllUsers();
        userList.setText("");
        users.forEach(user -> userList.append(user + "\n"));
    }

    public static void main(String[] args) {
        DatabaseInitializer.initialize();
        UserDAO userDAO = new UserDAO();
        new UserApp(userDAO).launch();
    }
}
```

3.4. Send current changes to Git repository
   a) Git → Add (to add newly created files)
   b) Git → Commit (to save local change set)
   c) Git → Remote → Push... (to send changes to the remote change set)

4. **Test application:**
   4.1. Prepare any kind of tests using JUnit
   4.2. The example below shows the concepts of testing the DAO Layer

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

public class UserDAOTest {
    private UserDAO userDAO;

    @BeforeEach
    public void setup() {
        DatabaseInitializer.initialize();
        userDAO = new UserDAO();
    }

    @Test
    public void testAddAndRetrieveUsers() {
        userDAO.addUser("John", "john@example.com");
        userDAO.addUser("Alice", "alice@example.com");

        List<String> users = userDAO.getAllUsers();
        assertEquals(2, users.size());
        assertTrue(users.contains("John - john@example.com"));
        assertTrue(users.contains("Alice - alice@example.com"));
    }
}
```