URL to GitHup Repository: https://github.com/aliustunyer/Menu-Driven-Application-Project-CRUD-Java-MySQL-Week-11-.git

URL to Youtube Video : https://youtu.be/OVF63mfHlJ0

```
package projects.dao;


import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Collection;
import java.util.LinkedList;
import java.util.List;
import java.util.Objects;

import projects.entity.Category;
import projects.entity.Material;
import projects.entity.Project;
import projects.entity.Step;
import projects.exception.DbException;

import provided.util.DaoBase;


public class ProjectDao extends DaoBase {

	private static final String CATEGORY_TABLE = "category";
	private static final String MATERIAL_TABLE = "material";
	private static final String PROJECT_TABLE = "project";
	private static final String PROJECT_CATEGORY_TABLE = "project_category";
	private static final String STEP_TABLE = "step";



	public Project insertProject(Project project) {
		//@formatter : off
		String sql = ""
		+ "INSERT INTO " + PROJECT_TABLE + " "
		+ "(project_name, estimated_hours, actual_hours, difficulty, notes) "
	    + "VALUES "
		+ "(?, ?, ?, ?, ?)";
		//@formatter : on

		try (Connection conn = DbConnection.getConnection()) {
		startTransaction(conn);

	   try (PreparedStatement stmt = conn.prepareStatement(sql)){
		setParameter (stmt, 1, project.getProjectName(), String.class);
		setParameter (stmt, 2, project.getEstimatedHours(), BigDecimal.class);
		setParameter (stmt, 3, project.getActualHours(), BigDecimal.class);
```

```
            setParameter (stmt, 4, project.getDifficulty(), Integer.class);
            setParameter (stmt, 5, project.getNotes(), String.class);

            stmt.executeUpdate();
            Integer projectId = getLastInsertId (conn, PROJECT_TABLE);
            commitTransaction(conn);
            project.setProjectId(projectId);
            return project;

         } catch (Exception e) {
             rollbackTransaction (conn);
             throw new DbException (e);
         }

        } catch (SQLException e) {
            throw new DbException(e);
        }

      }



      public List<Project> fetchAllProjects() {
            String sql = "SELECT * FROM " + PROJECT_TABLE + " ORDER BY
project_name";

            try (Connection conn = DbConnection.getConnection()) {
            startTransaction(conn);

            try (PreparedStatement stmt = conn.prepareStatement(sql)){
                  try (ResultSet rs = stmt.executeQuery()){
                        List <Project> projects = new LinkedList<>();

                        while (rs.next()) {
                              projects.add(extract (rs,Project.class));

                        }
                  return projects;
                  }
              }

            catch (Exception e) {
                  rollbackTransaction(conn);
                  throw new DbException(e);
            }
            }
                catch (SQLException e) {
                      throw new DbException(e);
            }
        }



      public java.util.Optional<Project> fetchProjectById(Integer projectId) {
            String sql = "SELECT * FROM " + PROJECT_TABLE + " WHERE project_id
= ?";
            try (Connection conn = DbConnection.getConnection()) {
            startTransaction(conn);
```

```
            try {
            Project project = null;

             try (PreparedStatement stmt = conn.prepareStatement(sql)){
                  setParameter (stmt, 1, projectId, Integer.class);

               try (ResultSet rs = stmt.executeQuery()) {
                     if (rs.next()) {
                           project = extract (rs, Project.class);
                     }
               }

            }

             if (Objects.nonNull(project)) {
                   project.getMaterials().addAll(fetchMaterialsForProject(conn,
projectId));
                   project.getSteps().addAll(fetchStepsForProject(conn,
projectId));
                   project.getCategories().addAll(fetchCategoriesForProject(conn,
projectId));

             }
            commitTransaction(conn);

            return java.util.Optional.ofNullable(project);
            }
            catch (Exception e) {
                  rollbackTransaction(conn);
                  throw new DbException(e);
            }
            }
                catch (SQLException e) {
                      throw new DbException(e);
            }
        }
```

```
      private List <Category> fetchCategoriesForProject(Connection conn, Integer
projectId) throws SQLException {

            //@formatter : off
            String sql = ""
                        + "SELECT c.* FROM " + CATEGORY_TABLE + " c "
                        + "JOIN " + PROJECT_CATEGORY_TABLE + " pc USING
(category_id) "
                        + "WHERE project_id = ?";
            //@formatter : on

            try (PreparedStatement stmt = conn.prepareStatement(sql)){
                  setParameter (stmt, 1, projectId, Integer.class);

                  try (ResultSet rs = stmt.executeQuery()){
                        List<Category> categories = new LinkedList <>();
```

```
                        while (rs.next()) {
                              categories.add(extract(rs, Category.class));
                        }
                        return categories;
                  }

            }


      }



      private List <Step> fetchStepsForProject(Connection conn, Integer
projectId)throws SQLException {
            String sql = "SELECT * FROM " + STEP_TABLE + " WHERE project_id = ?";

            try (PreparedStatement stmt = conn.prepareStatement(sql)){
                  setParameter (stmt, 1, projectId, Integer.class);

                  try (ResultSet rs = stmt.executeQuery()){
                        List<Step> steps = new LinkedList <>();

                        while (rs.next()) {
                              steps.add(extract(rs,Step.class));
                        }
                        return steps;
                  }

            }


      }

      private List  <Material> fetchMaterialsForProject(Connection conn, Integer
projectId) throws SQLException {
            String sql = "SELECT * FROM " + MATERIAL_TABLE + " WHERE project_id
= ?";

            try (PreparedStatement stmt = conn.prepareStatement(sql)){
                  setParameter (stmt, 1, projectId, Integer.class);

                  try (ResultSet rs = stmt.executeQuery()){
                        List<Material> materials = new LinkedList <>();

                        while (rs.next()) {
                              materials.add(extract(rs, Material.class));
                        }
                        return materials;
                  }
            }
        }



      public boolean modifyProjectDetails(Project project) {
            //@formatter : off
                        String sql = ""
```

```
                        + "UPDATE " + PROJECT_TABLE + " SET "
                        + "project_name = ?, estimated_hours = ?, actual_hours = ?,
"
                    + "difficulty = ?, notes = ? "
                    + "WHERE project_id = ?";
            //@formatter : on

                        try (Connection conn = DbConnection.getConnection()) {
                        startTransaction(conn);

                     try (PreparedStatement stmt = conn.prepareStatement(sql)){
                        setParameter (stmt, 1, project.getProjectName(),
String.class);
                        setParameter (stmt, 2, project.getEstimatedHours(),
BigDecimal.class);
                        setParameter (stmt, 3, project.getActualHours(),
BigDecimal.class);
                        setParameter (stmt, 4, project.getDifficulty(),
Integer.class);
                        setParameter (stmt, 5, project.getNotes(), String.class);
                        setParameter (stmt, 6, project.getProjectId(),
Integer.class);

                        boolean modified = stmt.executeUpdate() == 1;
                        commitTransaction(conn);



            return modified;
      }
                         catch (Exception e) {
                        rollbackTransaction (conn);
                        throw new DbException (e);
                     }

                    } catch (SQLException e) {
                        throw new DbException(e);
                    }
      }



      public boolean deleteProject(Integer projectId) {

            String sql = "DELETE FROM " + PROJECT_TABLE + " WHERE project_id = ?";

            try (Connection conn = DbConnection.getConnection()) {
            startTransaction(conn);

         try (PreparedStatement stmt = conn.prepareStatement(sql)){
            setParameter (stmt, 1, projectId, Integer.class);


            boolean deleted = stmt.executeUpdate() == 1;
            commitTransaction(conn);
            return deleted;
         }
              catch (Exception e) {
             rollbackTransaction (conn);
```

```
            throw new DbException (e);
         }

        } catch (SQLException e) {
            throw new DbException(e);
        }
         }
```

```
package projects.service;

import java.util.List;
import java.util.NoSuchElementException;
import java.util.Optional;

import projects.dao.ProjectDao;
import projects.entity.Project;
import projects.exception.DbException;

public class ProjectService {

      private ProjectDao projectDao = new ProjectDao();

      public Project addProject(Project project) {

            return projectDao.insertProject(project);
      }

      public List<Project> fetchAllProjects() {

            return projectDao.fetchAllProjects();


      }

      public Project fetchProjectById(Integer projectId) {

            return projectDao.fetchProjectById(projectId)
            .orElseThrow(() -> new NoSuchElementException("Project with project
ID=" + projectId + " does not exist."));
      }

      public void modifyProjectDetails(Project project) {
            if (!projectDao.modifyProjectDetails(project)) {
                  throw new DbException("Project with ID = " +
project.getProjectId() + " does not exist.");

            }

      }

      public void deleteProject(Integer projectId) {
            if (!projectDao.deleteProject(projectId)) {
                  throw new DbException("Project with ID = " + projectId + " does
not exist.");

      }
  }
}
```

```
package projects;

import java.math.BigDecimal;
import java.util.List;
import java.util.Objects;
import java.util.Scanner;

import projects.entity.Project;
import projects.exception.DbException;
import projects.service.ProjectService;

public class ProjectsApp {

          private Scanner scanner = new Scanner (System.in);
            private ProjectService projectService = new ProjectService();
            private Project curProject;


            //@formatter : off
      private List<String> operations = List.of (
                  "1) Add a project",
                  "2) List projects",
                  "3) Select a project",
                  "4) Update project details",
                  "5) Delete a project"
            );
            //@formatter : on


      public static void main(String[] args) {

            new ProjectsApp().processUserSelections();

}

      private void processUserSelections() {
            boolean done = false ;

            while (!done) {
                  try {
                        int selection = getUserSelection();
                        switch (selection) {
                        case -1 :
                              done = exitMenu();
                              break;
                        case 1 :
                              createProject();
                              break;
                        case 2 :
                              listProjects();
                              break;
                        case 3 :
                              selectProject();
                              break;
                        case 4 :
                              updateProjectDetails();
                              break;
```

```
                        case 5 :
                              deleteProject();
                              break;

                        default :
                              System.out.println("\n" + selection + " is not a
valid selection. Try again.");
                              break;

                        }
                  }
                  catch (Exception e) {
                        System.out.println ("\nError: "+ e + ". Try again.");
                  }
            }
      }

      private void deleteProject() {
            listProjects();
            Integer projectId = getIntInput("Enter the ID of the project to
delete");

            projectService.deleteProject(projectId);

            System.out.println("Project " + projectId + " was deleted
successfully.");

            if (Objects.nonNull(curProject) &&
curProject.getProjectId().equals((projectId))){
                  curProject = null;
            }
      }

      private void updateProjectDetails() {
            if (Objects.isNull(curProject)) {
                  System.out.println("\nPlease Select a project.");
                  return;
            }
            String projectName = getStringInput ("Enter the project name [" +
curProject.getProjectName()+ "]");
            BigDecimal estimatedHours = getDecimalInput ("Enter the estimated hours
[" + curProject.getEstimatedHours()+ "]");
            BigDecimal actualHours = getDecimalInput ("Enter the actual hours [" +
curProject.getActualHours()+ "]");
            Integer difficulty = getIntInput ("Enter the project difficulty (1-5)
[" + curProject.getDifficulty()+ "]");
            String notes = getStringInput ("Enter the project notes [" +
curProject.getNotes()+ "]");

            Project project = new Project();

            project.setProjectId(curProject.getProjectId());
            project.setProjectName (Objects.isNull(projectName) ?
curProject.getProjectName(): projectName);
            project.setEstimatedHours (Objects.isNull(estimatedHours) ?
curProject.getEstimatedHours(): estimatedHours);
            project.setActualHours (Objects.isNull(actualHours) ?
curProject.getActualHours(): actualHours);
```

```
            project.setDifficulty (Objects.isNull(difficulty) ?
curProject.getDifficulty(): difficulty);
            project.setNotes (Objects.isNull(notes) ? curProject.getNotes():
notes);

            projectService.modifyProjectDetails(project);

            curProject =
projectService.fetchProjectById(curProject.getProjectId());

      }

      private void selectProject() {
       listProjects();
       Integer projectId = getIntInput ("Enter a project ID to select a project");

       curProject= null;

       curProject= projectService.fetchProjectById (projectId);


      }

      private void listProjects() {
            List <Project> projects = projectService.fetchAllProjects();

            System.out.println("\nProjects:");

            projects.forEach(project -> System.out.println ("  " +
project.getProjectId()+ ": " + project.getProjectName()));



      }

      private void createProject() {

            String projectName = getStringInput ("Enter the project name");
            BigDecimal estimatedHours = getDecimalInput("Enter the estimated
hours");
            BigDecimal actualHours = getDecimalInput("Enter the actual hours");
            Integer difficulty = getIntInput("Enter the project difficulty (1-5)");
            String notes = getStringInput ("Enter the projects notes");

            Project project = new Project();

            project.setProjectName(projectName);
            project.setEstimatedHours(estimatedHours);
            project.setActualHours(actualHours);
            project.setDifficulty(difficulty);
            project.setNotes(notes);

            Project dbProject = projectService.addProject(project);
            System.out.println("You have successfully created project: " +
dbProject);
```

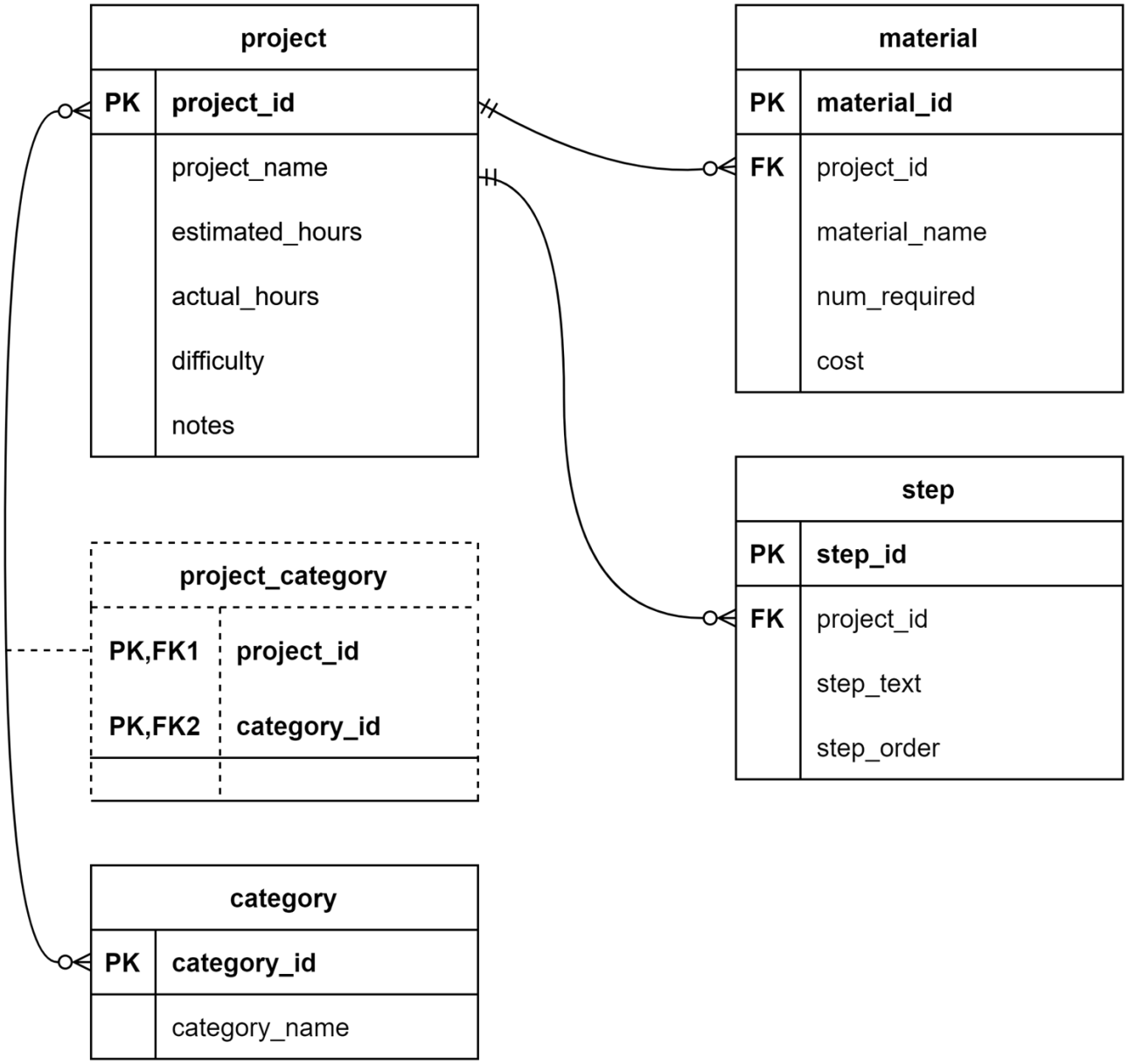
```
	}

	private BigDecimal getDecimalInput(String prompt) {
		String input = getStringInput(prompt);

		if (Objects.isNull(input)) {
			return null;
		}
		try {
			return new BigDecimal(input).setScale(2);

		}
		catch (NumberFormatException e) {
			throw new DbException(input + " is not a valid decimal number");

		}
	}

	private boolean exitMenu() {
		System.out.println("Exiting the menu.");
		return true;
	}

	private int getUserSelection() {
		printoperations ();

		Integer input = getIntInput ("Enter a menu selection");

		return Objects.isNull(input)? -1 : input;
	}

	private Integer getIntInput(String prompt) {
		String input = getStringInput(prompt);

		if (Objects.isNull(input)) {
			return null;
		}
		try {
			return Integer.valueOf(input);

		}
		catch (NumberFormatException e) {
			throw new DbException(input + " is not a valid number");

		}
	}

	private String getStringInput(String prompt) {
		System.out.print(prompt + ": ");
		String input = scanner.nextLine();
		return input.isBlank() ? null: input.trim();
	}

	private void printoperations() {
		System.out.println( "\nThese are the available selections. Press the
Enter key to quit:" );

		operations.forEach (line -> System.out.println("  "+ line));
```

```
            if (Objects.isNull(curProject)){
                  System.out.println("\nYou are not working with a project.");
            }
                  else {
                        System.out.println("\nYou are working with project: "
+curProject );

                  }
      }
```

```
CREATE TABLE project (
  project_id INT AUTO_INCREMENT NOT NULL,
  project_name VARCHAR(128) NOT NULL,
  estimated_hours DECIMAL (7,2),
  actual_hours DECIMAL (7,2),
  difficulty INT,
  notes TEXT,
  PRIMARY KEY (project_id)
);

CREATE TABLE category (
  category_id INT AUTO_INCREMENT NOT NULL,
  category_name VARCHAR(128) NOT NULL,
  PRIMARY KEY (category_id)
);

CREATE TABLE project_category (
  project_id INT NOT NULL,
  category_id INT NOT NULL,
  FOREIGN KEY (project_id) REFERENCES project (project_id) ON DELETE CASCADE,
  FOREIGN KEY (category_id) REFERENCES category (category_id) ON DELETE CASCADE,
  UNIQUE KEY (project_id, category_id)
);

CREATE TABLE step (
  step_id INT AUTO_INCREMENT NOT NULL,
  project_id INT NOT NULL,
  step_text TEXT NOT NULL,
  step_order INT NOT NULL,
  PRIMARY KEY (step_id),
  FOREIGN KEY (project_id) REFERENCES project (project_id) ON DELETE CASCADE
);

CREATE TABLE material (
  material_id INT AUTO_INCREMENT NOT NULL,
  project_id INT NOT NULL,
  material_name VARCHAR(128) NOT NULL,
  num_required INT,
  cost DECIMAL(7, 2),
  PRIMARY KEY (material_id),
  FOREIGN KEY (project_id) REFERENCES project (project_id) ON DELETE CASCADE
  );

insert into category (category_name)
values ('Doors and Windows');
select * from project;
```

```sql
insert into material (project_id, material_name, num_required)
values
(1, '2-inch screws', 20);

insert into step (project_id, step_text, step_order)
values
(1, 'Screw door hangers on the top and bottom of each side of the door frame', 1);

insert into project_category (project_id, category_id)
values
(1, 1);

insert into category (category_id, category_name) values(2, 'Repairs');
insert into category (category_id, category_name) values(3, 'Gardening');

insert into material (project_id, material_name, num_required, cost) values (2,
'faucet', 30, 4.55);
insert into material (project_id, material_name, num_required, cost) values (3,
'power point', 225, 19.99);

insert into step (project_id, step_text, step_order) values (2, 'Repair all of the
old parts', 2);
insert into step (project_id, step_text, step_order) values (3, 'Clear the garbege
in the garden', 3);

insert into project_category (project_id, category_id)
values
(2, 2);

insert into project_category (project_id, category_id)
values
(3, 3);
```

ERD ;


project
PK
project_id
project_name
estimated_hours
actual_hours
difficulty
notes
material
PK
material_id
FK
project_id
material_name
num_required
cost
step
PK
step_id
FK
project_id
step_text
step_order
project_category
PK,FK1
project_id
PK,FK2
category_id
category
PK
category_id
category_name

Main Class (ProjectApp)

```
package projects;

import projects.dao.DbConnection;

public class ProjectsApp {

	public static void main(String[] args) {

		DbConnection.getConnection();

	}

}
```

Class (Db Connection)

```
package projects.dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import projects.exception.DbException;

public class DbConnection {

	private static final String HOST = "localhost";
	private static final String PASSWORD = "projects";
	private static final int PORT = 3306;
	private static final String SCHEMA = "projects";
	private static final String USER = "projects";


	public static Connection getConnection() {

		String url =
String.format("jdbc:mysql://%s:%d/%s?user=%s&password=%s&useSSL=false", HOST, PORT,
SCHEMA, USER,
				PASSWORD);

		System.out.println("Connecting with url =" + url);

		try {
			Connection conn =DriverManager.getConnection(url);
```

```
                System.out.println("Successfully obtained connection!");
                return conn;
            } catch (SQLException e) {

                throw new DbException(e);
            }

        }
}
```

Class (Db Connection)

```
package projects.exception;

@SuppressWarnings("serial")
public class DbException extends RuntimeException {

        public DbException() {

        }

        public DbException(String message) {
            super(message);

        }

        public DbException(Throwable cause) {
            super(cause);

        }

        public DbException(String message, Throwable cause) {
            super(message, cause);

        }

}
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">

 <modelVersion>4.0.0</modelVersion>
  <groupId>com.promineotech</groupId>
  <artifactId>mysql-java</artifactId>
  <version>0.0.1-SNAPSHOT</version>


  <properties>
<java.version>17</java.version>
</properties>

<dependencies>
        <dependency>
                <groupId>com.mysql</groupId>
                <artifactId>mysql-connector-j</artifactId>
                <version>8.0.32</version>
        </dependency>
</dependencies>

<build>
   <pluginManagement>
     <plugins>
       <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-compiler-plugin</artifactId>
         <version>3.10.1</version>
         <configuration>
                        <source>${java.version}</source>
                        <target>${java.version}</target>
         </configuration>
       </plugin>
     </plugins>
   </pluginManagement>
  </build>

</project>
```