```
//Projects

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 Projects {
  private Scanner scanner = new Scanner(System.in);
  private ProjectService projectService = new ProjectService();
  private Project currentProject;

  //@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 Projects().displayUserOptions();
  }

  private void displayUserOptions() {
    boolean done = false;

    while (!done) {
      try {
        int operation = getOperation();
        switch (operation) {
        case -1:
          done = exitOptionsMenu();
          break;

        case 1:
          addProject();
          break;

        case 2:
          listProjects();
          break;

        case 3:
          selectProject();
          break;

        case 4:
          udpdateProjectDetails();
          break;

        case 5:
          deleteProject();
          break;

        default:
          System.out.println("\n" + operation + " is not valid. Please try again.");
          break;

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


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

```
    if(Objects.nonNull(projectId)) {
      projectService.deleteProject(projectId);

      System.out.println("You have deleted project with id of : " + projectId);

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

  private void udpdateProjectDetails() {
     if (Objects.isNull(currentProject)) {
       System.out.println("\nPlease select a project.");
       return;
   }

   String projectName = getStringInput("Enter the prject name " + currentProject.getProjectName());

   BigDecimal estimatedHours = getBigDecimalInput(
           "Enter the estimated hours " + currentProject.getEstimatedHours());

   BigDecimal actualHours = getBigDecimalInput("Enter the actual hours" + currentProject.getActualHours());

   Integer difficulty = getIntInput("Enter the difficulty (1-5)"  + currentProject.getDifficulty());

   String notes = getStringInput("Enter the project notes " + currentProject.getNotes());

   //update the project from inputs
   Project project = new Project();

   project.setProjectId(currentProject.getProjectId());
   project.setProjectName(Objects.isNull(projectName) ? currentProject.getProjectName() : projectName);
   project.setEstimatedHours(Objects.isNull(estimatedHours) ? currentProject.getEstimatedHours() : estimatedHours);
   project.setActualHours(Objects.isNull(actualHours) ? currentProject.getActualHours() : actualHours);
   project.setDifficulty(Objects.isNull(difficulty) ? currentProject.getDifficulty() : difficulty);
   project.setNotes(Objects.isNull(notes) ? currentProject.getNotes() : notes);
   projectService.modifyProjectDetails(project);
   currentProject = projectService.fetchProjectById(currentProject.getProjectId());
}

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

  }


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

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

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

    return projects;

  }

  private void addProject() {
    String name = getStringInput("Enter the project name ");
    BigDecimal estimatedHours = getBigDecimalInput("Enter the estimated hours ");
    BigDecimal actualHours = getBigDecimalInput("Enter the actual hours ");
    Integer difficulty = getIntInput("Enter the difficulty ");
    String notes = getStringInput("Enter the project notes ");

    Project project = new Project();

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

```
    Project dbProject = projectService.addProject(project);
    System.out.println("You added the following project: \n" + dbProject);

    currentProject = projectService.fetchProjectById(dbProject.getProjectId());

  }

  private boolean exitOptionsMenu() {
    System.out.println("\nYou have now EXITED the menu.");
    return true;
  }

  private int getOperation() {
    printOperations();
    Integer operation = getIntInput("\nEnter a operation number (Press ENTER to quit)");

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

  @SuppressWarnings("unused")
  private Integer getIntInput(String prompt) {
    String input = getStringInput(prompt);

    if (Objects.isNull(input)) {
      return null;
    }
    try {
      return Integer.valueOf(input);
    } catch (NumberFormatException exception) {
      throw new DbException(input + " is not a valid number.");
    }
  }

  @SuppressWarnings("unused")
  private BigDecimal getBigDecimalInput(String prompt) {
    String input = getStringInput(prompt);

    if (Objects.isNull(input)) {
      return null;
    }
    try {
      return new BigDecimal(input).setScale(2);
    } catch (NumberFormatException exception) {
      throw new DbException(input + " is not a valid decimal number.");
    }
  }

  private void printOperations() {
    System.out.println();
    System.out.println("Here's what you can do:");

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

    if (Objects.isNull(currentProject)) {
      System.out.println("\nYou are currently not working with a project!");
    } else {
      System.out.println("\nYou are working with project " + currentProject);
    }
  }

  private String getStringInput(String prompt) {
    System.out.println(prompt + ": ");
    String line = scanner.nextLine();

    return line.isBlank() ? null : line.trim();
  }
}
```

```
//DbConnection

package projects.dao;

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

```
import java.sql.SQLException;

import projects.exception.DbException;

public class DbConnection {
  private static String HOST = "localhost";
  private static String PASSWORD = "projects";
  private static int PORT = 3306;
  private static String SCHEMA = "projects";
  private static 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);
    try {
    Connection connection = DriverManager.getConnection(url);
    System.out.println("The connection succeeded!");
    return connection;
    }catch(SQLException exception) {
      System.out.println("The connection failed.");
      throw new DbException(exception);
    }

  }
}



//ProjectDao
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.sql.Statement;
import java.util.LinkedList;
import java.util.List;
import java.util.Objects;
import java.util.Optional;

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

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

      try (PreparedStatement statement = connection.prepareStatement(sql)) {
        try (ResultSet resultset = statement.executeQuery()) {
          List<Project> projects = new LinkedList<>();

          while (resultset.next()) {
            projects.add(extract(resultset, Project.class));
          }
          return projects;
        }
      } catch (Exception exception) {
        rollbackTransaction(connection);
        throw new DbException(exception);
      }
    } catch (SQLException exception) {
```

```
      throw new DbException(exception);
    }
  }

  public Optional<Project> fetchProjectById(Integer projectId) {
    String sql = "SELECT * FROM " + PROJECT_TABLE + " WHERE project_id = ?";

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

      try {
        Project project = null;

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

          try (ResultSet resultset = statement.executeQuery()) {
            if (resultset.next()) {
              project = extract(resultset, Project.class);
            }
          }
        }
        if (Objects.nonNull(project)) {
          project.getMaterials().addAll(fetchProjectMaterials(connection, projectId));

          project.getSteps().addAll(fetchProjectSteps(connection, projectId));
          project.getCategories().addAll(fetchProjectCategories(connection, projectId));
        }
        // first missing line of code
        commitTransaction(connection);
        return Optional.ofNullable(project);

      } catch (Exception exception) {
        rollbackTransaction(connection);
        throw new DbException(exception);
      }

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

  public void executeBatch(List<String> sqlBatch) {
    try (Connection connection = DbConnection.getConnection()) {
      startTransaction(connection);

      try (Statement statement = connection.createStatement()) {
        for (String sql : sqlBatch) {
          statement.addBatch(sql);
        }

        statement.executeBatch();
        commitTransaction(connection);

      } catch (Exception exception) {
        rollbackTransaction(connection);
        throw new DbException(exception);
      }
    } catch (SQLException exception) {
      throw new DbException(exception);
    }
  }

  private List<Category> fetchProjectCategories(Connection connection, Integer projectId) throws SQLException {
    String sql = "" + "SELECT ct.* " + "FROM " + CATEGORY_TABLE + " ct " + "JOIN " + PROJECT_CATEGORY_TABLE + " pct

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

      try (ResultSet resultSet = statement.executeQuery()) {
        List<Category> categories = new LinkedList<Category>();

        while (resultSet.next()) {
          categories.add(extract(resultSet, Category.class));
        }

        return categories;
```

```
      }
    }
  }

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

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

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

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

        return steps;
      }
    }
  }

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

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

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

        while (resultset.next()) {
          Material material = extract(resultset, Material.class);

          materials.add(material);
        }

        return materials;
      }
    }
  }

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

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

      try (PreparedStatement statement = connection.prepareStatement(sql)) {
        setParameter(statement, 1, project.getProjectName(), String.class);
        setParameter(statement, 2, project.getEstimatedHours(), BigDecimal.class);
        setParameter(statement, 3, project.getActualHours(), BigDecimal.class);
        setParameter(statement, 4, project.getDifficulty(), Integer.class);
        setParameter(statement, 5, project.getNotes(), String.class);
        statement.executeUpdate();

        Integer projectId = getLastInsertId(connection, PROJECT_TABLE);
        commitTransaction(connection);

        project.setProjectId(projectId);
        return project;

      } catch (Exception exception) {
        rollbackTransaction(connection);
        throw new DbException(exception);
      }
    } catch (SQLException exception) {
      throw new DbException(exception);
    }
  }

//  public void addMaterialToProject(Material material) {
//    String sql = "INSERT INTO " + MATERIAL_TABLE + " (material_id, project_id, material_name, num_required, cost)
//        + " VALUES (?, ?, ?, ?, ?)";
//
//    try (Connection connection = DbConnection.getConnection()) {
```

```
//      startTransaction(connection);
//
//      try {
//        Integer order = getNextSequenceNumber(connection, material.getProjectId(), MATERIAL_TABLE, "project_id");
//        try (PreparedStatement statement = connection.prepareStatement(sql)) {
//          setParameter(statement, 1, material.getMaterialId(), Integer.class);
//          setParameter(statement, 2, material.getProjectId(), Integer.class);
//          setParameter(statement, 3, material.getMaterialName(), String.class);
//          setParameter(statement, 4, material.getNumRequired(), BigDecimal.class);
//          setParameter(statement, 5, material.getCost(), BigDecimal.class);
//
//          statement.executeUpdate();
//          commitTransaction(connection);
//
//        }
//      } catch (Exception exception) {
//        rollbackTransaction(connection);
//        throw new DbException(exception);
//      }
//    } catch (SQLException exception) {
//      throw new DbException(exception);
//    }
//
//  }

//  public void addStepToProject(Step step) {
//    String sql = "INSERT INTO " + STEP_TABLE + " (project_id, step_order, step_text)" + " VALUES (?, ?, ?)";
//
//    try (Connection connection = DbConnection.getConnection()) {
//      startTransaction(connection);
//
//      Integer order = getNextSequenceNumber(connection, step.getProjectId(), STEP_TABLE, "project_id");
//
//      try (PreparedStatement statement = connection.prepareStatement(sql)) {
//        setParameter(statement, 1, step.getProjectId(), Integer.class);
//        setParameter(statement, 2, order, Integer.class);
//        setParameter(statement, 3, step.getStepText(), String.class);
//
//        statement.executeUpdate();
//        commitTransaction(connection);
//      } catch (Exception exception) {
//        rollbackTransaction(connection);
//        throw new DbException(exception);
//      }
//    } catch (SQLException exception) {
//      throw new DbException(exception);
//    }
//  }

//  public List<Category> fetchAllCategories() {
//    String sql = "SELECT * FROM " + CATEGORY_TABLE + " ORDER BY category_name";
//    try (Connection connection = DbConnection.getConnection()) {
//      startTransaction(connection);
//
//      try (PreparedStatement statement = connection.prepareStatement(sql)) {
//        try (ResultSet resultSet = statement.executeQuery()) {
//          List<Category> categories = new LinkedList<>();
//
//          while (resultSet.next()) {
//            categories.add(extract(resultSet, Category.class));
//          }
//          return categories;
//        }
//      } catch (Exception exception) {
//        rollbackTransaction(connection);
//        throw new DbException(exception);
//      }
//    } catch (SQLException exception) {
//      throw new DbException(exception);
//    }
//  }

//  public void addCategoryToProject(Integer projectId, String category) {
//    String subQuery = "(SELECT category_id FROM " + CATEGORY_TABLE + " WHERE category_name = ?)";
//
//    String sql = "INSERT INTO " + PROJECT_CATEGORY_TABLE + " (project_id, category_id) VALUES (?, " + subQuery +
//
```

```
//    try (Connection connection = DbConnection.getConnection()) {
//      startTransaction(connection);
//
//      try (PreparedStatement statement = connection.prepareStatement(sql)) {
//        setParameter(statement, 1, projectId, Integer.class);
//        setParameter(statement, 2, category, String.class);
//
//        statement.executeUpdate();
//        commitTransaction(connection);
//
//      } catch (Exception exception) {
//        rollbackTransaction(connection);
//        throw new DbException(exception);
//      }
//
//    } catch (SQLException exception) {
//      throw new DbException(exception);
//    }
//  }

  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: off

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

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

        boolean updated = statement.executeUpdate() == 1;
        commitTransaction(connection);

        return updated;

      } catch (Exception exception) {
        rollbackTransaction(connection);
        throw new DbException(exception);
      }

    } catch (Exception exception) {
      throw new DbException(exception);
    }
  }

  public boolean deleteProject(Integer projectId) {
    String sql = "DELETE FROM " + PROJECT_TABLE + " WHERE project_id = ?";

    try (Connection connection = DbConnection.getConnection()) {
      startTransaction(connection);
      try (PreparedStatement statement = connection.prepareStatement(sql)) {
        setParameter(statement, 1, projectId, Integer.class);

        boolean deleted = statement.executeUpdate() == 1;

        commitTransaction(connection);
        return deleted;

      } catch (Exception exception) {
        rollbackTransaction(connection);
        throw new DbException(exception);
      }
    } catch (SQLException exception) {
      throw new DbException(exception);
```

```
    }
  }
}


//Category

package projects.entity;

public class Category {
  private Integer categoryId;
  private String categoryName;

  public String getCategoryName() {
    return categoryName;
  }
  public void setCategoryName(String categoryName) {
    this.categoryName = categoryName;
  }
  public Integer getCategoryId() {
    return categoryId;
  }
  public void setCategoryId(Integer categoryId) {
    this.categoryId = categoryId;
  }
  @Override
  public String toString() {
    return "ID = " + categoryId + ", categoryName = " + categoryName;
  }

}


//Material

package projects.entity;

import java.math.BigDecimal;

public class Material {
  private Integer materialId;
  private Integer projectId;
  private String materialName;
  private Integer numRequired;
  private BigDecimal cost;

  public Integer getMaterialId() {
    return materialId;
  }
  public void setMaterialId(Integer materialId) {
    this.materialId = materialId;
  }
  public Integer getProjectId() {
    return projectId;
  }
  public void setProjectId(Integer projectId) {
    this.projectId = projectId;
  }
  public String getMaterialName() {
    return materialName;
  }
  public void setMaterialName(String materialName) {
    this.materialName = materialName;
  }
  public Integer getNumRequired() {
    return numRequired;
  }
  public void setNumRequired(Integer numRequired) {
    this.numRequired = numRequired;
  }
  public BigDecimal getCost() {
    return cost;
  }
  public void setCost(BigDecimal cost) {
    this.cost = cost;
  }
}
```

```java
//Project

package projects.entity;

import java.math.BigDecimal;
import java.util.LinkedList;
import java.util.List;

public class Project {
  private Integer projectId;
  private String projectName;
  private BigDecimal estimatedHours;
  private BigDecimal actualHours;
  private Integer difficulty;
  private String notes;

  private List<Material> materials = new LinkedList<>();
  private List<Step> steps = new LinkedList<>();
  private List<Category> categories = new LinkedList<>();

  @Override
  public String toString() {
    String project = " ";

    project += "\n ID = " + projectId;
    project += "\n Project Name = " + projectName;
    project += "\n Estimated hours = " + estimatedHours;
    project += "\n Actual Hours = " + actualHours;
    project += "\n Difficulty = " + difficulty;
    project += "\n Notes = " + notes;

    project += "\n Materials: ";

    for (Material material : materials) {
      project += "\n    " + material;
    }

    project += "\n  Steps";

    for (Step step : steps) {
      project += "\n    " + step;
    }

    project += "\n  Categories";

    for(Category category : categories) {
      project+= "\n   "+ category;
    }

    return project;
  }

  public Integer getProjectId() {
    return projectId;
  }

  public void setProjectId(Integer projectId) {
    this.projectId = projectId;
  }

  public String getProjectName() {
    return projectName;
  }

  public void setProjectName(String projectName) {
    this.projectName = projectName;
  }

  public String getNotes() {
    return notes;
  }

  public void setNotes(String notes) {
    this.notes = notes;
```

```
  }

  public BigDecimal getEstimatedHours() {
    return estimatedHours;
  }

  public void setEstimatedHours(BigDecimal estimatedHours) {
    this.estimatedHours = estimatedHours;
  }

  public BigDecimal getActualHours() {
    return actualHours;
  }

  public void setActualHours(BigDecimal actualHours) {
    this.actualHours = actualHours;
  }

  public List<Material> getMaterials() {
    return materials;
  }

  public void setMaterials(List<Material> materials) {
    this.materials = materials;
  }

  public List<Step> getSteps() {
    return steps;
  }

  public void setSteps(List<Step> steps) {
    this.steps = steps;
  }

  public List<Category> getCategories() {
    return categories;
  }

  public void setCategories(List<Category> categories) {
    this.categories = categories;
  }
  public int getDifficulty() {
    return difficulty;
  }

  public void setDifficulty(Integer difficulty) {
    this.difficulty = difficulty;
  }

}



//projects

package projects.entity;

public class Step {
  private Integer stepId;
  private Integer projectId;
  private String stepText;
  private Integer stepOrder;



  public Integer getStepId() {
    return stepId;
  }
  public void setStepId(Integer stepId) {
    this.stepId = stepId;
  }
  public Integer getProjectId() {
    return projectId;
  }
  public void setProjectId(Integer projectId) {
    this.projectId = projectId;
  }
```

```
  public Integer getStepOrder() {
    return stepOrder;
  }
  public void setStepOrder(Integer stepOrder) {
    this.stepOrder = stepOrder;
  }
  public String getStepText() {
    return stepText;
  }
  public void setStepText(String stepText) {
    this.stepText = stepText;
  }

}



//DbException
package projects.exception;

public class DbException extends RuntimeException {

  public DbException() {
    // TODO Auto-generated constructor stub
  }

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

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

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

}


//ProjectService
package projects.service;

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

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

  }

  public List<Project> fetchAllProjects() {
    return projectDao.fetchAllProjects();
  }

  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.");
    }
  }
}
```

```
//RESOURCES
//GitHub: https://github.com/fmd5045/Week07-11SQLProjectRedo/tree/main
//YouTube: https://youtu.be/rcuK02JZFmU
```