ProjectsApp.java

```
/**
 *
 */
package projects;

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

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

/**
 * @author Candace Samuels
 */

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.");
				}
```

```
                } catch (Exception e) {
                        System.out.println("\nError: " + e + " Try again.");
                }
        }
    }
    //WEEK 11
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.nonNull(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.fetchProjectByProjectId(curProject.getProjectId());
	}


// WEEK 10: START
	private void selectProject() {
		listProjects();
		Integer projectId= getIntInput("Enter a project ID to select a project");


		/* Unselect the current project. */
		curProject = null;


		/* This will throw an exception if an invalid project ID is entered. */
		curProject = projectService.fetchProjectByProjectId(projectId);
}
```

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

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

			projects.forEach (
			  project -> System.out.println(" " + project.getProjectId() + ":" +
project.getProjectName()));
			    // Lambda expression used
		}

// WEEK 9 START
	@SuppressWarnings("unused")
	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 project 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 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 + ": "); // test the application
            String input = scanner.nextLine();

            return input.isBlank() ? null : input.trim();
        }

        private void printOperations() {
            System.out.println("These are the avaliable menu sections. Press the Enter key to
quit:");
        // WEEK 10 START
            if(Objects.isNull(curProject)) {
                System.out.println("\nYou are not working with a project");
            }
            else {
                System.out.println("\nYou are working with project: " + curProject);
            }
        // WEEK 10 END
            operations.forEach(line -> System.out.println("  " + line));
        }
}
```

ProjectService.java

```
package projects.service;

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

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

```
@SuppressWarnings("unused")
public class ProjectService {
	private ProjectDao projectDao = new ProjectDao();

	public Project addProject(Project project) {
		return ProjectDao.insertProject(project);
	}

	public ProjectDao getProjectDao() {
		return projectDao;
	}

	public void setProjectDao(ProjectDao projectDao) {
		this.projectDao = projectDao;
	}
// WEEK 10 START :

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

	}

	public Project fetchProjectByProjectId(Integer projectId) {
		Optional<Project> op = projectDao.fetchProjectById(projectId);
		return projectDao.fetchProjectById(projectId).orElseThrow(() -> new
NoSuchElementException(
				"Project with Project ID=" + projectId + "does not exist."));
	}
//WEEK 10 END:

//WEEK 11 START:
	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.");
		}
	}
//WEEL 11 END:
}
```

ProjectDao.java

```
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 java.util.Optional;

import projects.dao.exception.DbException;
import projects.entity.Category;
import projects.entity.Material;
import projects.entity.Project;
import projects.entity.Step;
import provided.util.DaoBase;

/**
 * This class uses JDBC to perform CRUD opeartions on the project tables.
 *
 * @author Candace Samuels
```

```
 *
 */
@SuppressWarnings("unused")
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 static 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);
        }
    }

// WEEK 10: START
    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 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 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;
                }
        }
    }
//WEEK 11 START:
    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);
        }
    }
//WEEK 11 END:
```