MySQL Week 3 Exercises

Background

In the homework exercises, you are creating an application that will perform CRUD (Create, Read, Update, and Delete) operations on a MySQL database. This application connects to a DIY Project database and demonstrates many features of SQL and JDBC. Learning these skills will help prepare you for using these skills in the workplace.

As a reminder, in the last two weeks you used JDBC to connect to a MySQL database. Then, you diagrammed the project tables using Draw.io. Lastly, you wrote the CREATE TABLE statements for the five tables and created the tables in DBeaver.

In this week's exercises, you will begin development of the menu-driven application. You will use proper exception handling to gracefully manage any errors. You will write code to add project details to the project tables. This will involve properly creating and managing JDBC resources as well as database transactions.

In future exercises you will write code to read from a single table as well as from joined tables. Finally, you will write code to update and delete table rows.

Objectives

In these exercises, you will:

- Learn to write a menu-driven application with correct exception handling.
- Implement a scanner to gather user input from the console.
- Learn how to work with `BigDecimal` objects.
- Use JDBC to correctly handle resources (Connections and PreparedStatements) ensuring that they are closed properly.
- Implement JDBC methods to insert a Project object into the project table.

Important

In the exercises below, you will see this icon: . This means to take a screen shot or snip showing the results of the action or the code in the editor.

Exercises

In these exercises you will write code to create a menu-driven application. You will display menu selections to the user and will write the code to add project details to the DIY project tables. This will demonstrate the use of the INSERT statement (the Create part of CRUD).

Complete these exercises as directed. If you get hopelessly stuck, please see the "Solutions" section below.

In these exercises, you will often be told to call a method prior to creating it. This is a good approach. You set up the return type by assigning to a variable and set up the parameters. Then, Eclipse can correctly create the method.

Cleanup

In this section, you are working with `ProjectsApp.java` in the `projects` package.

1. Delete the debugging line (`DbConnection.getConnection();`) in the main method. The method should now be empty.

2. Remove the import statement: `import projects.dao.DbConnection;`

Build the Menu Application

The exercises in this section build a menu-driven application. This application displays a list of available operations. The user selects which operation to perform. A switch statement then routes the selection to the appropriate method. Along the way you will add in proper exception handling. This is an important step to get right when building any application.

The purpose of the menu application is to perform CRUD operations on a relational database that holds information on DIY projects. Throughout the coming weeks you will add to this application to insert project rows, then materials, steps, and categories. You will fetch projects as a list and fetch an individual project with all the details. You will modify rows and delete an entire project with all associated detail (child) rows.

In this section, you are working with `ProjectsApp.java` in the `projects` package.

1. In order to display a list of menu options you must store them somewhere. In this step you will write the code that holds the list of operations.

   a. Add a private instance variable named "`operations`". The type is `List<String>`. Initialize it using `List.of` with the following value: "`1) Add a project`". To prevent the Eclipse formatter from reformatting the list, surround the variable declaration with `// @formatter:off` and `// @formatter:on` so that it looks like this:

   ```
   // @formatter:off
   private List<String> operations = List.of(
       "1) Add a project"
   );
   // @formatter:on
   ```

   This list of operations will be printed on the console so that the user will be reminded which selection to make.

2. In this step you will use a `Scanner` to obtain input from a user from the Java console. A `Scanner` is a Java object that can be used to read from a variety of sources. When you create the `Scanner`, you will set its input source to `System.in`, which is the opposite of `System.out`. You use `System.out` to print to the console. You will use the `Scanner` to read from the console. So, the user types in selections and the `Scanner` reads the input and gives it to the application.

Add a `private` instance variable named `scanner`. It is of type `java.util.Scanner`. Initialize it to a `new Scanner` object. Pass `System.in` to the constructor. This will set the scanner so that it accepts user input from the Java console. It should look like this:

```
private Scanner scanner = new Scanner(System.in);
```

3. In this step you will call the method that processes the menu. In the `main()` method, create a new `ProjectsApp` object and call the method: `processUserSelections()` method. The method takes zero parameters and returns nothing.

```
new ProjectsApp().processUserSelections();
```

4. Now you can create the `processUserSelections()` method as an instance method. This method displays the menu selections, gets a selection from the user, and then acts on the selection. Let Eclipse create the method for you by waving your mouse over the compiler error in the `main()` method (over the red squigglies). Eclipse will pop up a menu. Select "`Create method processUserSelections()`".

In method `processUserSelections()`:

    a. Add a local variable:

```
boolean done = false;
```

    b. Add a `while` loop below the local variable. Loop until the variable `done` is `true`.

```
boolean done = false;

while(!done) {
}
```

    c. Inside the `while` loop, add a `try/catch` block. The `catch` block should catch `Exception`. Inside the `catch` block print the `Exception` message. Call the `toString()` method on the `Exception` object provided to the `catch` block. This is done by simply concatenating the `Exception` object onto a `String` literal. When you do this Java implicitly calls the `toString()` method behind the scenes.

    d. Inside the `try` block, assign an `int` variable named `selection` to the return value from the method `getUserSelection()`. The method should now look like this:

```java
private void processUserSelections() {
    boolean done = false;

    while(!done) {
        try {
            int selection = getUserSelection();

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

5. Create the method `getUserSelection()`. It takes no parameters and returns an `int`. This method will print the `operations` and then accept user input as an `Integer`. In the `getUserSelection()` method:

    a. Make a method call to the method `printOperations()`. This method takes no parameters and returns nothing.

    b. Add a method call to `getIntInput()`. Assign the results of the method call to a variable named `input` of type `Integer`. The method `getIntInput()`, which you haven't written yet. It will return the user's menu selection. The value may be `null`. Pass the `String` literal `"Enter a menu selection"` as a parameter to the method.

    c. Add a return statement that checks to see if the value in local variable `input` is `null`. If so, return `-1`. (The value `-1` will signal the menu processing method to exit the application.) Otherwise, return the value of `input`. The method should look like this:

```java
private int getUserSelection() {
    printOperations();

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

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

6. Create the method `printOperations()`. It takes no parameters and returns nothing. This method does just what it says, it prints each available selection on a separate line in the console. In the `printOperations()` method:

    a. Print a line to the console:

```java
"\nThese are the available selections. Press the Enter key to quit:");
```

    b. Print all the available menu selections, one on each line. Each line should be indented slightly (2 or 3 spaces). Use any strategy that you choose to print the instructions. If you use a Lambda expression as shown in the video, it should look like this:

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

Every List object must implement the `forEach()` method. `forEach()` takes a `Consumer` interface object as a parameter. `Consumer` has a single abstract method, `accept()`. The `accept()` method takes a single parameter and returns nothing. The Lambda expression has a single parameter and `System.out.println` returns nothing. The Lambda expression thus matches the requirements for the `accept()` method.

If you don't want to use a Lambda expression, you can use an enhanced for loop to print the instructions.

7. There will be several user input methods that return different types of objects. Due to the way the `java.util.Scanner` object was implemented, the safest way to get an input line from the user is to input it as a String and then convert it to the appropriate type. With this design, all the input methods will ultimately call the `String` input method, which actually prints the prompt and uses the Scanner to get the user's input. In this step, you will write a method that returns an `Integer` value.

Create the method `getIntInput`. It takes a single parameter of type `String` named `prompt`. This method accepts input from the user and converts it to an `Integer`, which may be `null`. It is called by `getUserSelection()` and will be called by other data collection methods that require an `Integer`. Inside the method body:

   a. Assign a local variable named `input` of type `String` to the results of the method call `getStringInput(prompt)`.

   b. Test the value in the variable `input`. If it is `null`, return `null`. Use `Objects.isNull()` for the `null` check.

   c. Create a `try/catch` block to test that the value returned by `getStringInput()` can be converted to an `Integer`. The `catch` block should accept a parameter of type `NumberFormatException`.

      i. In the `try` block, convert the value of input, which is a `String`, to an `Integer` and return it. If the conversion is not possible, a `NumberFormatException` is thrown. The message in the `NumberFormatException` is totally obscure so it will get fixed in the `catch` block. Here's what the contents of the `try` block should look like:

      ```
      return Integer.valueOf(input);
      ```

      ii. In the `catch` block throw a new `DbException` with the message, `input + " is not a valid number. Try again."`

   d. The method should look like this:

```java
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.");
  }
}
```

8. Now create the method that really prints the prompt and gets the input from the user. Create the method `getStringInput()`. It should have a single parameter of type `String` named `prompt`. This is the lowest level input method. The other input methods call this method and convert the input value to the appropriate type. This will also be called by methods that need to collect `String` data from the user. It should return a `String`. Inside the method:

   a. Print the prompt using `System.out.print(prompt + ": ")` to keep the cursor on the same line as the prompt. (Note: `print` and not `println`!)

   b. Assign a `String` variable named `input` to the results of a method call to `scanner.nextLine()`.

   c. Test the value of `input`. If it is blank return `null`. Otherwise return the trimmed value.

   d. The method should look like this:

   ```java
   private String getStringInput(String prompt) {
     System.out.print(prompt + ": ");
     String input = scanner.nextLine();

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

   e. At this point the file should have no compile errors.

9. Now we want to add code that will process the user's selection. Since the user enters an `Integer` value (the menu selection number) you can use a `switch` statement to process the selection.

   Back in the method `processUserSelections()`:

   a. Add a `switch` statement below the method call to `getUserSelection()`. Create a `switch` statement to switch on the value in the local variable `selection`.

   b. Add the first case of `-1`. Inside this case, call `exitMenu()` and assign the result of the method call to the local variable `done`. Make sure to add the `break` statement.

c. Add the default case. Print a message: `"\n" + selection + " is not a valid selection. Try again."`.

10. Now that the menu code has been written you will need to test it to see if it works. Test the application two ways:

a. This will test that a non-integer selection prints an error message and gracefully recovers. Run the application. Click in the Eclipse console so that input will go to the scanner. Enter "`abc`" (without quotes) and press `Enter`. You should get an error message and be prompted again to enter a valid selection. Now press `Enter` with no input. The application should quit. Take a screenshot to show the application output 📷 . It should look something like this:

```
These are the available selections. Press the Enter key to quit:
  1) Add a project
Enter a menu selection: abc

Error: projects.exception.DbException: abc is not a valid number. Try again.

These are the available selections. Press the Enter key to quit:
  1) Add a project
Enter a menu selection:
Exiting the menu.
```

```
These are the available selections. Press the Enter key to quit:
   1. Add a project
Enter a menu selection: abc

Error: projects.exception.DbException: abc is not a valid number. Try again.

These are the available selections. Press the Enter key to quit:
   1. Add a project
Enter a menu selection:
```

b. This will test that entering a valid `Integer` without a corresponding `case` statement will print an error message and recover gracefully. Run the application. Click in the Eclipse console so that input will go to the scanner. Enter "`5`" (without quotes) and press `Enter`. You should get an error message and be prompted again to enter a valid selection. Now press `Enter` with no input. The application should quit. Take a screen shot to show the application output 📷 . It should look something like this:

```
These are the available selections. Press the Enter key to quit:
  1) Add a project
Enter a menu selection: 5

5 is not a valid selection. Try again.

These are the available selections. Press the Enter key to quit:
  1) Add a project
Enter a menu selection:
Exiting the menu.
```

```
These are the available selections. Press the Enter key to quit:
  1. Add a project
Enter a menu selection: 5

5 is not a valid selection. Try again.

These are the available selections. Press the Enter key to quit:
  1. Add a project
Enter a menu selection:
Exiting the menu.
```

Add project files from student resources

Promineo Tech has provided some resources so that you don't have to write every bit of code. In this section, you will add files into the `mysql-java` project from the student resources.

1. Drag the four files from the student resources `/Homework/entity` folder and drop them onto the `projects.entity` package in the Eclipse project. You may need to expand some folders in the package explorer to make the `projects.entity` package visible. When done you should see `Category.java`, `Material.java`, `Project.java`, and `Step.java` in the `projects.entity` package. There should be no errors visible in those files.

2. Drag the directory named "`provided`" from the student resources `/Homework` folder and drop it onto `src/main/java` in the package explorer. When done, there should be a new package named `provided.util` with a single file in it named `DaoBase.java`.

Add a new project to the project table

You will now write the code to collect project information, create the project entities and insert the project row into the project table.

Modifications to the main application file

This section will collect project information from the user and call the service class (not written yet) to store the project row. In this section you will be introduced to the `BigDecimal` class if you haven't seen it before.

The `BigDecimal` class exactly represents decimal numbers (numbers with decimal places). In this, the decimal numbers act like `Integer`s with a known number of decimal places. Money is a good example of this. Your bank may perform operations on money that results in fractional pennies, but at the end of

the day your account is credited or debited with an exact dollar and penny amount. Fractional pennies are transitive and are not persisted to your account.

The `BigDecimal` object is perfectly suited to handle the SQL `DECIMAL` data type. `DECIMAL` values have a fixed number of digits (precision) and a fixed number of decimal places (scale). In the `CREATE TABLE` statement, a column definition of `DECIMAL (5, 2)` means that the value can range from -999.99 to 999.99. There are a maximum of five digits (precision) with two decimal places (scale).

`BigDecimal` contains immutable values – once created they cannot be changed. Any operation performed on `BigDecimal` results in a new `BigDecimal` object. `BigDecimal`s can be created using a constructor, then the scale can be set by calling the `setScale()` method. To set the scale to 2 (two decimal places) do something like this:

```
BigDecimal bd = new BigDecimal("1234.5678").setScale(2);
```

The JDBC driver has methods to natively handle `BigDecimal`. The driver automatically converts from Java `BigDecimal` to SQL `DECIMAL` and vice versa.

In this section you will be working in `ProjectsApp.java`.

1. At the top of the class, add a private instance variable of type `ProjectService` named `projectService` and call the zero-argument constructor to initialize it. Let Eclipse create the `ProjectService` class. Make sure the class is created in the `projects.service` package. (Hint: wave the mouse over `ProjectService`, which should have red squigglies under it. When the menu pops up, click "`Create class 'ProjectService'`". When the Java Class wizard pops up, change the value in the field "`Package`" from "`projects`" to "`projects.service`".) The editor will switch over to `ProjectService.java`. Switch back to `ProjectsApp.java`.

2. In this step, you will add code in the `switch` statement to handle user selection "`1`", which will call a method to collect project details and save them in the project table.

   In the method `processUserSelections()`, add `case 1` to the switch statement. Inside the case, call the method `createProject()`. This method takes no parameters and returns nothing. Remember to add the `break` statement.

3. Now write the method to gather the project details from the user. Once collected, they will be put into a `Project` object. Then, another method will be called to save the project details.

   Create the method `createProject()`. It is `private`, takes no parameters, and returns nothing. In this method:

   a. Add local variable `String projectName`. Assign the value to the result of calling `getStringInput("Enter the project name")`.

   b. Add local variable `BigDecimal estimatedHours`. Assign the value to the result of calling `getDecimalInput("Enter the estimated hours")`. You may need to add the import statement for `BigDecimal`. It is in the `java.math` package.

   c. Add local variable `BigDecimal actualHours`. Assign the value to the result of calling `getDecimalInput("Enter the actual hours")`.

d. Add local variable `Integer difficulty`. Assign the value to the result of calling `getIntInput("Enter the project difficulty (1-5)")`. Note that the instructions don't include code to validate that the input is valid. You can do this if you want.

e. Add local variable `String notes`. Assign the value to the result of calling `getStringInput("Enter the project notes")`.

f. Create a new variable of type `Project` named `project`. Initialize it to a `new Project` object by calling the zero-argument constructor. Import the `Project` class from the `projects.entity` package. The `Project` class should have been added to the Eclipse project in the section "Add project files from student resources." If Eclipse can't find the import, follow the instructions in that section.

g. Call the appropriate setters on the `Project` object to set `projectName`, `estimatedHours`, `actualHours`, `difficulty` and `notes`. For example, to add the project name on the Project object, call `setProjectName()` and pass it `projectName`.

h. Call the `addProject()` method on the `projectService` object. Pass it the `Project` object. This method will be created shortly. This method should return an object of type `Project`. Assign it to variable `dbProject`.

i. Print a success message to the console `"You have successfully created project: " + dbProject`. The value returned from `projectService.addProject()` is different from the `Project` object passed to the method. It contains the project ID that was added by MySQL.

The method should look like this. (There will be an error on the line that calls the `projectService` object and the lines that call `getDecimalInput()`).

```java
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);
}
```

4. To get rid of the compilation errors, you will need to create two methods. In this step you will create the method `getDecimalInput()`.

Create the method `getDecimalInput()`. The easiest way to do this is to create the method body, then copy the method contents from `getIntInput()` and paste it into the method body. Fix the following lines:

    a.   The line in the `try` block. Change it to:

```
return new BigDecimal(input).setScale(2);
```

       This will create a new `BigDecimal` object and set the number of decimal places (the scale) to 2.

    b.   The message in `DbException`. Change it to:

```
input + " is not a valid decimal number."
```

5.   Now create the second method that will fix the compilation errors. Wave the mouse over "`projectService.addProject()`". When the menu pops up, select "`Add method 'addProject(project)' in in type 'ProjectService'`".

6.   Save all files. All compiler errors should now be gone.

Modifications to project service

The service layer in this small application is implemented by a single file, `ProjectService.java`. *Mostly* this file acts as a pass-through between the main application file that runs the menu (`ProjectsApp.java`) and the DAO file in the data layer (`ProjectDao.java`).

In this section you will be working in `ProjectService.java`.

1.   In this step, you will create the DAO class and initialize a variable of that type. At the top of the class, add a private instance variable of type `ProjectDao` named `projectDao`. Assign the variable to a new `ProjectDao` object by calling the constructor with no parameters. If possible, let Eclipse create the class for you. In any event, create a `ProjectDao` class in the `projects.dao` package. Make sure that `ProjectDao` extends `DaoBase` from the `provided.util` package. Save all files. You should have no compile errors. The editor will probably change to the `ProjectDao` class. Change back to the `ProjectService` class.

2.   In method `addProject()`, call the method `insertProject()` on the `projectDao` object. The method should take a single parameter. Pass it the `Project` parameter and return the value from the method. The `addProject()` method should look like this:

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

3.   Wave the mouse over `insertProject()` (with the red squigglies) and select "`Create method 'insertProject(Project)' in type 'ProjectDao'`". Save all files. You should have no compile errors.

Modifications to project DAO

Now you want to create the class that will read and write to the MySQL database. In this section you will write the values that were collected from the user and that are contained in a `Project` object to the project table using JDBC method calls.

In this section you will be working in file `ProjectDao.java` in the `projects.dao` package. If you followed the steps above, `ProjectDao` should extend `DaoBase`. If it doesn't, do that now or you will run into problems later.

Add constants

First, you should add some constants with the table names. It's a good idea to add constants for values that are used over and over again in a class. The table names are used by all the methods that write to or read from the tables.

In this section, you will be adding constants into the `ProjectDao` class. These are placed at the top of the class just inside the class body. Java does not have a "constant" keyword. Instead, a constant is specified using `static final`. Constants can either be `public` or `private`. In this file all the constants should be `private`.

1. Add the constant for the category table named `CATEGORY_TABLE`. Set the value to `"category"`.

2. Add the constant for the material table named `MATERIAL_TABLE`. Set the value to `"material"`.

3. Add the constant for the project table named `PROJECT_TABLE`. Set the value to `"project"`.

4. Add the constant for the project-category table named `PROJECT_CATEGORY_TABLE`. Set the value to `"project_category"`.

5. Add the constant for the step table named `STEP_TABLE`. Set the value to `"step"`.

The constants should look like this:

```java
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";
```

Save the project details

There are several steps that must be taken to save the project details. First, you must create the SQL statement. Then you will obtain a `Connection` and start a transaction. Next you will obtain a `PreparedStatement` and set the parameter values from the `Project` object. Finally, you will save the data and commit the transaction. Follow the steps below to save the project details.

In this section, you will be working exclusively in the method `insertProject()` in `ProjectDao.java`.

1. Write the SQL statement that will insert the values from the `Project` object passed to the `insertProject()` method. Remember to use question marks as placeholder values for the parameters passed to the `PreparedStatement`. Add the fields `project_name`,

estimated_hours, actual_hours, difficulty, and notes. Make sure to add the correct blank spaces between words or it won't work. It should look like this:

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

2. Obtain a connection from DbConnection.getConnection(). Assign it a variable of type Connection named conn in a try-with-resource statement. Catch the SQLException in a catch block added to the try-with-resource. From within the catch block, throw a new DbException. The DbException constructor should take the SQLException object passed into the catch block.

```
try(Connection conn = DbConnection.getConnection()) {
}
catch(SQLException e) {
    throw new DbException(e);
}
```

3. Start a transaction. Inside the try block, start a transaction by calling startTransaction() and passing in the Connection object. startTransaction() is a method in the base class, DaoBase.

4. Obtain a PreparedStatement object from the Connection object. Inside the try block and below startTransaction(), add another try-with-resource statement to obtain a PreparedStatement from the Connection object.

    a. Pass the SQL statement as a parameter to conn.prepareStatement().

    b. Add a catch block to the inner try block that catches Exception. In the catch block, roll back the transaction and throw a DbException initialized with the Exception object passed into the catch block. This will ensure that the transaction is rolled back when an exception is thrown.

    c. The method should look like this at this point:

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

Transaction started

Transaction rolled back

5. In this step you will set the project details as parameters in the `PreparedStatement` object. Inside the inner `try` block, set the parameters on the `Statement`. Use the convenience method in DaoBase `setParameter()`. This method handles `null` values correctly. (See the JavaDoc comments on that method for details.) Add these parameters: `projectName`, `estimatedHours`, `actualHours`, `difficulty`, and `notes`. When done it should look like this:

```java
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);
```

6. Now you can save the project details. Perform the insert by calling `executeUpdate()` on the `PreparedStatement` object. Do not pass any parameters to `executeUpdate()` or it will reset all the parameters leading to an obscure error.

7. Obtain the project ID (primary key) by calling the convenience method in `DaoBase`, `getLastInsertId()`. (See the JavaDoc documentation on that method for details.) Pass the `Connection` object and the constant `PROJECT_TABLE` to `getLastInsertId()`. Assign the return value to an `Integer` variable named `projectId`.

8. Commit the transaction by calling the convenience method in `DaoBase`, `commitTransaction()`. Pass the `Connection` object to `commitTransaction()` as a parameter.

9. Set the `projectId` on the `Project` object that was passed into `insertProject` and return it. At this point there should be no compile errors. The method should now look like this:

```java
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);
  }
}
```

Test it

After all that coding, it's a good idea to test that it actually works. You need to ensure that you can add a project row to the project table with no errors.

1. Run the application.

2. Enter the menu selection "1".

3. Enter project name, estimated hours, actual hours, difficulty and notes.

4. Take a screen shot 📷 of the console output showing the data entry and the printed Project object. It should look something like this:

```
These are the available selections. Press the Enter key to quit:
  1) Add a project
Enter a menu selection: 1
Enter the project name: Hang a door
Enter the estimated hours: 4
Enter the actual hours: 3
Enter the project difficulty (1-5): 3
Enter the project notes: Use the door hangers from Home Depot
Connection to schema 'projects' is successful.
You have successfully created project:
   ID=1
   name=Hang a door
   estimatedHours=4.00
   actualHours=3.00
   difficulty=3
   notes=Use the door hangers from Home Depot
   Materials:
   Steps:
   Categories:

These are the available selections. Press the Enter key to quit:
  1) Add a project
Enter a menu selection:
Exiting the menu.
```

```
These are the available selections. Press the Enter key to quit:
  1. Add a project
Enter a menu selection: 1
Enter the project name: grocery list
Enter the estimated hours: 3
Enter the actual hours: 2
Enter the project difficulty (1-5): 2
Enter the project notes: need more bread
Connection to schema projects is successful.
You have successfully created project:
   ID=2
   name=grocery list
   estimatedHours=3.00
   actualHours=2.00
   difficulty=2
   notes=need more bread
   Materials:
   Steps:
   Categories:

These are the available selections. Press the Enter key to quit:
  1. Add a project
Enter a menu selection:
Exiting the menu.
```

Solutions

These solutions are provided as a reference. Please work through the exercises on your own as best you can.

ProjectsApp.java

```java
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;

/**
 * This class is a menu-driven application that accepts user input from the console. It then
 * performs CRUD operations on the project tables.
 *
 * @author Promineo
 *
 */
public class ProjectsApp {
  private Scanner scanner = new Scanner(System.in);
  private ProjectService projectService = new ProjectService();

  // @formatter:off
  private List<String> operations = List.of(
      "1) Add a project"
  );
  // @formatter:on

  /**
   * Entry point for Java application.
   *
   * @param args Unused.
   */
  public static void main(String[] args) {
    new ProjectsApp().processUserSelections();
  }
}
```

```java
/**
 * This method prints the operations, gets a user menu selection, and performs the requested
 * operation. It repeats until the user requests that the application terminate.
 */
private void processUserSelections() {
  boolean done = false;

  while(!done) {
    try {
      int selection = getUserSelection();

      switch(selection) {
        case -1:
          done = exitMenu();
          break;

        case 1:
          createProject();
          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.");
    }
  }
}

/**
 * Gather user input for a project row then call the project service to create the row.
 */
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);
}
```

```java
/**
 * Gets the user's input from the console and converts it to a BigDecimal.
 *
 * @param prompt The prompt to display on the console.
 * @return A BigDecimal value if successful.
 * @throws DbException Thrown if an error occurs converting the number to a BigDecimal.
 */
private BigDecimal getDecimalInput(String prompt) {
  String input = getStringInput(prompt);

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

  try {
    /* Create the BigDecimal object and set it to two decimal places (the scale). */
    return new BigDecimal(input).setScale(2);
  }
  catch(NumberFormatException e) {
    throw new DbException(input + " is not a valid decimal number.");
  }
}

/**
 * Called when the user wants to exit the application. It prints a message and returns
 * {@code true} to terminate the app.
 *
 * @return {@code true}
 */
private boolean exitMenu() {
  System.out.println("Exiting the menu.");
  return true;
}

/**
 * This method prints the available menu selections. It then gets the user's menu selection from
 * the console and converts it to an int.
 *
 * @return The menu selection as an int or -1 if nothing is selected.
 */
private int getUserSelection() {
  printOperations();

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

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

```java
/**
 * Prints a prompt on the console and then gets the user's input from the console. It then
 * converts the input to an Integer.
 *
 * @param prompt The prompt to print.
 * @return If the user enters nothing, {@code null} is returned. Otherwise, the input is converted
 *         to an Integer.
 * @throws DbException Thrown if the input is not a valid Integer.
 */
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.");
  }
}

/**
 * Prints a prompt on the console and then gets the user's input from the console. If the user
 * enters nothing, {@code null} is returned. Otherwise, the trimmed input is returned.
 *
 * @param prompt The prompt to print.
 * @return The user's input or {@code null}.
 */
private String getStringInput(String prompt) {
  System.out.print(prompt + ": ");
  String input = scanner.nextLine();

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

/**
 * Print the menu selections, one per line.
 */
private void printOperations() {
  System.out.println("\nThese are the available selections. Press the Enter key to quit:");

  /* With Lambda expression */
  operations.forEach(line -> System.out.println("   " + line));

  /* With enhanced for loop */
  // for(String line : operations) {
  // System.out.println("  " + line);
  // }
}
}
```

ProjectService.java

```java
package projects.service;

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

/**
 * @author Promineo
 *
 */
public class ProjectService {
  private ProjectDao projectDao = new ProjectDao();

  /**
   * This method simply calls the DAO class to insert a project row.
   *
   * @param project The {@link Project} object.
   * @return The Project object with the newly generated primary key value.
   */
  public Project addProject(Project project) {
    return projectDao.insertProject(project);
  }
}
```

ProjectDao.java

```java
package projects.dao;

import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import projects.entity.Project;
import projects.exception.DbException;
import provided.util.DaoBase;

/**
 * This class uses JDBC to perform CRUD operations on the project tables.
 *
 * @author Promineo
 *
 */
@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";
```

```java
    /**
     * Insert a project row into the project table.
     *
     * @param project The project object to insert.
     * @return The Project object with the primary key.
     * @throws DbException Thrown if an error occurs inserting the row.
     */
    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);
      }
    }

}
```