Exercise 1-2 Import and run two database projects

This exercise guides you through the process of using Eclipse to import and run two applications that get data from a database. Before you do this exercise, you should make sure to install the database as described in the appendixes.

Import and run the console version of the Product Manager application

1. Start Eclipse. Then, import the project named ch20_ProductManager. This project should be stored in this folder:

   murach/java_eclipse/book_apps

2. Expand all three packages and note how this project uses seven .java files.
3. Expand the Libraries folder and note how it includes a library for the MySQL database driver.
4. Open the Main.java file in the code editor and review its code. For now, don't worry if you don't understand this code! You'll learn how to write code like this later in this book.
5. Run the project.
6. Use the "list" command to display all of the products in the database.
7. Use the "add" command to add a new product. Then, list the products to make sure your product was added.
8. Use the "del" command to delete the product you just added. Then, list the products to make sure the product was deleted.
9. When you are done experimenting, exit the application.

Import and run the GUI version of the Product Manager application

10. Import the project named ch22_ProductManager.
11. Expand all four packages and note how this project uses eight .java files.
12. Expand the Libraries folder and note how it includes a library for the MySQL database driver.
13. Open the Main.java file in the code editor and review its code. For now, don't worry if you don't understand this code! You'll learn how to write code like this later in this book.
14. Run the project. After the application starts, examine the table of products.
15. Click the Add button and add a new product. When you are finished, verify that your new product shows up in the table of products.
16. Click on the row for the product you just added. Then, click the Delete button to delete it. When you are finished, verify that this product has been deleted from the table of products.
17. Click on an existing product. Then, click the Edit button to edit this product. Change one of the values. For example, change the price. When you are finished, verify that the change is reflected in the table of products.
18. Add a new product and enter an invalid value for the price. For example, enter "five" for the price. What happens?

Set the main project and run the applications again

19. Select the ch20_ProductManager project in the Projects window. Then, click the Run Project button to run this application.
20. Select the ch22_ProductManager project in the Projects window. Then, press F6 to run this application.
21. Exit Eclipse. If Eclipse displays a dialog that indicates that these applications are still running, confirm that you want to exit and stop those applications.

Exercise 2-2 Experiment with the Code Tester application

In this exercise, you can experiment with the Code Tester application that's presented at the end of this chapter.

Test the application

1. Start the Eclipse IDE and import the project named ch02_ex2_CodeTester. This project should be in this folder:

   /murach/java_eclipse/extra_ex_starts

2. Expand the com.murach.testing package, open the file named CodeTesterApp.java, and review the code for this file.
3. Run this application. It should print a welcome message and a bye message.

Initialize and print more variables

4. Within the main method, declare and initialize two integer variables named x and y with the values of 3 and 5.
5. Write statements that print the values of the variables to the console. For example. The console should print something like the following:

```
The value of x is: 5
The value of y is: 3
```

6. Run the application to make sure it works correctly.

Work with arithmetic expressions

7. Add another statement that prints the results of dividing x by y.
8. Run the application. The console output should look like this:

```
The value of x is: 5
The value of y is: 3
5 / 3 = 1
```

9. Note that the answer has been rounded down to the nearest whole number.
10. Change the declarations for the x and y variables from the int type to the double type.
11. Run the application.
12. Note that the answer includes decimal places.

Work with escape sequences

13. Modify the first statement in the main method, so that it uses an escape sequence to add a new line to the end of the welcome message.
14. Delete the second statement in the main method. (The one that prints a blank line to the console.)
15. Run the application. It should work as before.
16. Add code that uses escape sequences to print this output to the console:

```
Could not find folder named "C:\Program Files\Test"
```

17. Add code that uses the escape sequence for a new line to add blank lines between each code example.
18. Run the application to make sure it works correctly.

Calculate the area and perimeter of a rectangle

19. Add a statement that declares a double variable for width and initialize it to a value of 4.25.
20. Add a statement that declares a double variable for length and initialize it to a value of 8.5.
21. Calculate the perimeter of a rectangle that has the width and length set in the previous two steps. To do that, you can use this formula:

```
perimeter = 2 * width + 2 * length
```

22. Print the values for the width, length, and perimeter to the console like this:

```
Width:     4.25
Length:    8.5
Perimeter: 25.5
```

23. Run the application to make sure it works correctly.

Code an assignment statement

24. Add a statement that assigns a value of 16.75 to the width. Make sure to code this statement after the statement that initializes the width variable but before the statement that calculates the perimeter.
25. Run the application to make sure it works correctly. The values for the width and perimeter should reflect the new width.

Exercise 3-3 Create the Number Guessing Game

In this exercise, you'll create a simple number guessing game that allows you to guess the value of a random number. Each time you make a guess, the game tells you whether your guess is too high or too low. This process repeats until you guess the correct number. A sample run of the application should look like this:

```
Welcome to the Number Guessing Game

Enter the upper limit for the number: 50
OK, I'm thinking of a number between 0 and 50

Enter your guess: 25
Your guess is too high.

Enter your guess: 20
Your guess is too low.

Enter your guess: 23
Correct!

Bye!
```

Create the game

1. Start Eclipse and import the project named ch03_ex3_GuessingGame in the extra_ex_starts directory.
2. Review the existing code. Note the use of the Random class. This code generates a random integer between 0 and upperLimit. For the purposes of this exercise, you don't need to understand how it works.
3. Write the code that prompts the user for an upper limit for the guess. This code should set the upperLimit variable which is already defined.
4. Write the code that prompts the user for their first guess.
5. Write the while loop that allows the user to guess again if their guess was wrong. This loop should inform the user whether their guess was too high or too low. Then, it should prompt the user for a new guess. The loop should exit when the user's guess is equal to the random number the computer generated.
6. After the while loop, write code that tells the user they guessed correctly.
7. Run the application and make sure it works correctly.

Enhance the game

8. Before the while loop, add a variable to keep track of the number of guesses the user has made. Remember to initialize this variable to 1 instead of 0 since the user will need at least one guess to get the correct number.
9. Inside the while loop, increment the variable by one each time the user guesses incorrectly.
10. After the while loop, add a message that tells the user how many guesses they took.

Exercise 4-3 Create the Circle Calculator

In this exercise, you'll create an application that computes the diameter, circumference, and area of a circle. A sample run of the application should look like this:

```
Welcome to the Circle Calculator

Enter radius:  10
Diameter:      20.0
Circumference: 62.8318
Area:          314.159

Continue? (y/n): n

Bye!
```

Create a new project and a Circle class

1. Start Eclipse and use it to create a new project named ch04_ex3_Circle. Make sure you create a Main class and put it in a package. You can use murach.circle as the name of the package.
2. Create a new class named Circle and store it in the murach.circle package.
3. In the Circle class, add an instance variable for the radius of the circle. If possible, use the IDE to generate get and set methods for this variable.
4. Add a constructor to the Circle class that doesn't take any arguments. This constructor should set the instance variable for the radius to 0.
5. Add a method named getDiameter that returns the diameter of the circle. To do that, this method can multiply the radius by 2.

Use the Circle class

6. Add code to the main method that gets the radius of the circle from the user.
7. Add code that creates a Circle object. Then, add code that uses the setRadius to set the radius in the Circle object.
8. Add code that uses the getDiameter method to get the diameter from the Circle object. Then, add code that displays the diameter.
9. Run the application to make sure it works correctly.

Add the circumference and area to the application

10. Add a method named getCircumference that calculates and returns the circumference of the circle. To do that, you can multiply Pi by 2. You can use 3.14159 as the value of Pi.
11. Add a method named getArea that calculates and returns the area of the circle. To do that, the code for this method can multiply Pi by the radius squared, which is the radius multiplied by the radius.
12. Add code to the main method that uses the getCircumference and getArea methods to display the circumference and area of the Circle object.
13. Run the application and make sure it works correctly.

Add a second constructor to the Circle class and use it

14. Add a second constructor to the Circle class that takes an argument for the radius. This constructor should sets the instance variable for the radius to the value of the argument.
15. Modify the code in the main method so it uses the second constructor to create the Circle object.
16. Comment out the statement that uses the setRadius method since that method is no longer necessary.

Exercise 4-4 Create an object and use it with the Number Guessing Game

In this exercise, you'll convert a number guessing game like the one described in exercise 3-3 so it uses some object-oriented features.

Review the project

1. Start Eclipse and import the project named ch04_ex4_GuessingGame in the extra_ex_starts directory.
2. Run the project and make sure it works correctly. Also, make sure you understand the code in the main method.

Create a class that stores the data for the game

3. In the murach.games package, create a new class named NumberGame.
4. In this class, create one instance variable for storing the upper limit of the number, a second for storing the number, and a third for the number of guesses the user has made.
5. Add a constructor to the class that takes an integer value for the upper limit and uses it to set the upper limit instance variable. Then, generate the number that the user should try to guess and set that instance variable. You can copy and modify the relevant lines of code from the main method to do this. Finally, initialize the instance variable for the number of guesses to 1.
6. Add get methods for all three instance variables. Use your IDE to do this if possible. You don't need to create set methods.
7. Add a method named incrementGuessCount that adds 1 to the instance variable for the number of guesses.

Use the class

8. In the Main class modify the code so it uses the new object. For example, use the getUpperLimit method to display the upper limit to the user. Then, remove any unnecessary code.
9. Run the project again and makes sure it still works correctly.

Add a second constructor and use it

10. In the NumberGame class, add a constructor to the class that takes no arguments. The code for this constructor can call the other constructor in this class and pass it a value of 50 for the upper limit.
11. In the Main class, modify the code so it uses the zero-argument constructor. Then, comment out the statements that get the upper limit from the user. These statements are no longer necessary since the constructor automatically sets the upper limit to 50.
12. Run the project again and makes sure it works correctly. It should set an upper limit of 50 by default.

Exercise 5-3 Add packages to the Number Guessing Game

In this exercise, you'll add packages to the Number Guessing Game like the one described in exercise 3-3.

Review and test the project

1. Start Eclipse and import the project named ch05_ex3_GuessingGame in the extra_ex_starts directory.
2. Review the code for the Main and NumberGame classes. Note that they are currently all in the same package. Run the application and make sure it works correctly.
3. Run the project to make sure it works correctly.

Add packages to the project and move the classes

4. Use Eclipse to create a new package named murach.games.guessing. Then, move the Main class into it.
5. Use Eclipse to create a new package named murach.games.guessing.business. Then, move the NumberGame class into it.
6. Review the code in the NumberGame class and note that Eclipse has automatically updated the package statement for you.
7. Review the code for the Main class and note that Eclipse has automatically updated the package statement. Also, note that Eclipse has automatically added an import statement for the NumberGame class.
8. Run the project to make sure it still works correctly.

Exercise 6-2 Debug the Number Guessing Game

In this exercise, you'll get more practice working with Eclipse debugging capabilities by debugging a version of the Number Guessing Game that has bugs in it.

Review the project

1. import the project named ch06_ex2_GuessingGame in the extra_ex_starts directory.
2. Review the code and note that Eclipse displays error icons for the compile time errors.

Find and fix the compile-time errors

3. Attempt to run the project. This should display an error message that contains links to the errors.
4. Find and fix these errors by clicking on the links to jump to the code for the error. Sometimes the error marker in the left margin may not be on the same line as the actual error.
5. Find and fix any remaining compile-time errors. To do that, you can use the error icons to locate the compile-time errors. Then, you can fix them.

Find and fix the runtime error

6. Run the application to test it. Note that it contains a runtime error. Specifically, the game almost always tells you that your guess is is either too high or too low, no matter which number you pick. For example, even if you chose 0, it might tell you that your guess is too high.
7. Use the Eclipse debugger to find and fix the bug that is causing this problem.

   Hint: As you step through the code, keep an eye on the value of the guess variable.

Exercise 7-3 Modify the Future Value application

In this exercise, you'll use some of the skills that you learned in this chapter to modify the Future Value application.

Work with the increment operator and order or precedence

1. Import the project named ch07_ex3_FutureValue that's in the extra_ex_starts folder. Then, review the code for this project and run it until you understand how it works.
2. Open the Main class. Within the while loop, use the increment operator (++) to increment the counter variable named i.
3. Combine the first three statements in the while loop into a single statement that calculates the future value. To get this to work, you can use parentheses to control the order of precedence of the operations. (First, add the monthly investment. Then, add the monthly interest.)
4. Run the application to make sure it still works correctly.

Add a yearly fee

5. Before the while loop, declare a constant that stores a yearly fee of $50.
6. Within the while loop, add an if statement that uses the modulus operator to check whether the current month is a multiple of 12. If so, subtract the yearly fee from the future value.
7. After the while loop, add a statement that applies currency formatting to the yearly fee and prints it to the console. When you're done, the console should display user input and calculation results like this:

```
Enter monthly investment:   100
Enter yearly interest rate: 3
Enter number of years:      3
Yearly fee:                 $50.00
Future value:               $3,616.85
```

Exercise 7-4 Improve the Area and Perimeter app

In this exercise, you'll modify the Area and Perimeter application so it uses a constant and some methods of the Math class.

Add a constant to the Rectangle class

1. Import the project named ch07_ex4_AreaAndPerimeter that's in the extra_ex_starts folder. Then, review the code for this project.
2. Open the Rectangle class and add a constant that stores the minimum fraction digits for the area and perimeter.
3. Modify the getAreaNumberFormat and getPerimeterNumberFormat methods so they use the constant defined in the previous step to set the minimum fraction digits for the NumberFormat object.
4. Run the application to make sure it still works correctly.
5. Change the minimum fraction digits to 0 by modifying the value that's stored in the constant.
6. Run the application to make sure it still works correctly.

Add another calculation

7. In the Rectangle class, add a method named getDiagonal that calculates the diagonal of a rectangle. Search the Internet to find the formula for calculating the diagonal of a rectangle.

   Hint: To perform this calculation, you can use the pow and sqrt methods of the Math class.
8. In the Rectangle class, add a method named getDiagonalNumberFormat. This method should apply number formatting to the value that's returned by the getDiagonal method. This method should work like the other methods in this class that apply number formatting.
9. In the Main class, modify the code so it displays the diagonal after the area and perimeter.
10. Run the application and make sure it works correctly. The console should display the user input and the results of the calculations like this:

```
Enter length: 100
Enter width:  100
Area:         10,000
Perimeter:    400
Diagonal:     141.421
```

Exercise 7-5 Improve the Circle Calculator

In this exercise, you'll improve the accuracy of the Circle Calculator.

Review the project

1. Import the project named ch07_ex5_Circle in the extra_ex_starts directory.
2. Open the Main and Circle classes and review the code. Note that the Circle class uses a value of 3.14159 for the value of Pi.
3. Run the application to make sure it works correctly.

Use the Math class to get Pi

4. In the Circle class, modify the code so it uses the PI constant that's available from the Math class to get the value of Pi.
5. In the Main class, modify the code so it prints the value of Pi after the Welcome message.
6. Run the application and make sure it works correctly. After a successful run, the console should look something like this:

```
Welcome to the Circle Calculator

The value of Pi is: 3.141592653589793

Enter radius:  100
Area:          31415.926535897932
Circumference: 628.3185307179587
Diameter:      200.0

Continue? (y/n):
```

Exercise 8-3 Improve the Number Guessing Game

In this exercise, you'll improve the Number Guessing Game.

Modify the loop statement

1. Import the project named ch08_ex3_GuessingGame in the extra_ex_starts directory.
2. Open the Main class. Modify the loop that prompts the user for a number so that it's an infinite while loop. When you do that, you only need to code the two statements that get input from the user at the top of the loop.
3. Modify the if/else statement so it uses a break statement to jump out of the loop if the user guesses the correct number.
4. Run the application to make sure it works correctly.

Add some if/else statements

5. Within the loop, modify the if/else statement so the application says, "Way too high!" if the user's guess is more than 10 higher than the random number. Otherwise, the application should just say, "Your guess is too high."
6. After the loop, add an if/else statement that displays a message that depends on the user's number of guesses. For example:

```
Number of guesses      Message
=================      =======
<=3                    Great work! You are a mathematical wizard.
>3 and <=7             Not too bad! You've got some potential.
>7                     What took you so long? Maybe you should take
                       some lessons
```

7. Run the application to make sure it works correctly.

Add a try/catch statement to get a valid integer

8. In the Main class, add a try/catch statement so it catches the NumberFormatException that's thrown by the parseInt method. If this exception is thrown, display a message to the console that says, "Invalid number" and jump to the top of the loop. This should prompt the user to enter the number again.
9. Run the application to make sure it works correctly.

Add an if/else statement to make sure the integer is within a range

10. Add an if/else statement to make sure the user enters a value between the minimum and maximum values. If the user enters a value that's less than or equal to 0, display a user-friendly error message and jump to the top of the loop. Conversely, if the user enters a value that's greater than or equal to the game's upper limit, display a user-friendly error message and jump to the top of the loop.
11. Run the application to make sure it works correctly.

Exercise 8-4 Improve the Circle Calculator

In this exercise, you'll improve the way the Circle Calculator handles user input.

Review the application

1. Import the project named ch08_ex4_Circle in the extra_ex_starts directory.
2. Open the classes for this application and review the code. Note that the Main class does not use the Console class to get input from the user.
3. Run the application to make sure it works correctly.

Use the Console class to get user input

4. In the Main class, modify the code so it uses the Console class to get input from the user. When you're done, remove all unnecessary code. This should include all code that works with the Scanner class since that code is all in the Console class.
5. Run the application to make sure it works correctly.

Add a new method to the Console class

6. In the Console class, add a getDouble method that accepts three parameters: (1) a String object for the prompt, (2) a double type for a minimum value, and (3) a double type for a maximum value. This method should continue to prompt the user for a double value until the user enters a valid double value within the minimum and maximum parameters. Then, this method should return the valid double value.
7. Open the Main class and edit it so it uses the getDouble method of the Console class to specify minimum and maximum values for the radius. For example, you might want to specify that the radius should be greater than 0 but less than 100,000.
8. Run the application to make sure it works correctly.

Add another new method to the Console class

9. In the Console class, add a getRequiredString method that accepts one parameter: (1) a String object for the prompt. This method should continue to prompt the user for a string until the user enters a non-empty string. If the user enters an empty string, display a user-friendly error message and prompt the user again.
10. In the Main class, use the getRequiredString method of the Console class to get the choice string that determines whether the application continues or exits. This should prevent the user from pressing Enter at the prompt without entering a string.
11. Run the application to make sure it works correctly.

Exercise 9-3 Parse a string of product data

In this exercise, you'll parse a string that contains data for a product and store that data in a Product object.

Review the application

1. Import the project named ch08_ex3_ProductParser in the extra_ex_starts directory.
2. Open the Main and Product classes and review the code. Note that the Main class defines a Product object and a String object that stores the data for a product with each field delimited by a colon (:). However, this code doesn't parse this string and store the data in the Product object.
3. Run the application. At this point, it shouldn't print any product data to the console.

Add the code that parses the string

4. In the Main class, add code that parses the string and gets the three fields that are stored in the string. Then, store this data in the Product object.
5. Run the application and make sure it works correctly. After a successful run, the console should look something like this:

```
Code:        java
Description: Murach's Java Programming
Price:       $57.50
```

Exercise 9-4 Modify the Area and Perimeter Calculator

In this exercise, you'll modify the Area and Perimeter Calculator so it allows the user to enter the length and width on the same line.

Review the application

1. Import the project named ch09_ex4_AreaAndPerimeter in the extra_ex_starts directory.
2. Open the Main class and review its code.
3. Run the application to make sure it works correctly.

Modify the application so it allows two entries on the same line

4. In the Main class, modify the code that gets the length and the width so it prompts the user to enter the length and width on the same line. Then, parse the string that's entered by the user to get one String object for the length and another for the width.
5. Modify the code that displays the data for the Rectangle object so that it includes the length and width.
6. Run the application to make sure it works correctly. If the user enters 100 and 20, the data that's printed to the console should look something like this:

```
Enter length and width: 100 20

Length:       100.0
Width:        20.0
Area:         2,000.000
Perimeter:    240.000
```

Exercise 10-3 Work with an array of product data

In this exercise, you'll work with a two-dimensional array that stores the data for three products.

Review the application

1. Import the project named ch10_ex3_ProductArray in the extra_ex_starts directory.
2. Open the classes and review the code. Note that the ProductDB class defines a two-dimensional array that stores the data for three products. Also, note that this class contains three methods that aren't implemented.
3. Run the application. At this point, it should print some messages to the console, but these messages won't contain any product data.

Add the code that works with the array of product data

4. In the ProductDB class, add code that implements the getProductByIndex method. This should return a Product object for the product data at the specified index.
5. In the Main class, add code that uses the getProductByIndex method and displays the data that's returned by it.
6. Run the application and make sure it works correctly. At this point, it should print data for the PRODUCT BY INDEX heading.
7. In the ProductDB class, add code that implements the getProductByCode method. This should return a Product object for the product data with the specified code.
8. In the Main class, add code that uses the getProductByCode method and displays the data that's returned by it.
9. Run the application and make sure it works correctly. At this point, it should print data for the PRODUCT BY CODE heading.
10. In the ProductDB class, add code that implements the getAllProducts method. This should return an array of Product objects for all of the product data.
11. In the Main class, add code that uses the getAllProducts method and displays the data that's returned by it. To do that, you'll need to loop through the array of Product objects that's returned.
12. Run the application and make sure it works correctly. At this point, it should print data for the LIST OF ALL PRODUCTS heading.

Exercise 10-4 Work with an array of Circle objects

In this exercise, you'll modify the Circle Calculator so it works with two or more circles.

Review the application

1. Import the project named ch10_ex4_Circle in the extra_ex_starts directory.
2. Open the Main class and review its code.
3. Run the application to make sure it works correctly.

Modify the application so it works with two circles

4. Modify the code that gets the radius from the user so it prompts the user to enter one or more radiuses on the same line.
5. Modify the code so it parses the line of text that the user enters to get an array of String objects with one String object for each entry. To do that, you can use the split method of the String class like this:

```
String entries[] = line.split(" ");
```

6. Create a loop that creates an array of Circle objects where each circle object corresponds with an entry in the array of String objects.
7. Create another loop that loops through the array of Circle objects and displays the data for each Circle object on the console. This data should include the radius. To do this, use a StringBuilder object to store the string that contains the data for the Circle objects.
8. Run the application to make sure it works correctly. If the user enters 100 and 200, the data that's printed to the console should look something like this:

```
Enter one or more radiuses:  100 200

Radius:        100.0
Area:          31415.899999999998
Circumference: 628.318
Diameter:      200.0

Radius:        200.0
Area:          125663.59999999999
Circumference: 1256.636
Diameter:      400.0
```

However, this should also work if the user enters one radius or if the user enters more than two radiuses.

Exercise 11-4 Inherit the Product class

In this exercise, you'll modify a version of the Product application so it adds a class named MyProduct that extends the Product class and enhances its functionality.

Review the application

1. Import the project named ch11_ex4_Product in the extra_ex_starts directory.
2. Open the classes and review their code.
3. Run the application to make sure it works correctly.

Create a new subclass and use it

4. In the murach.business package, create a new class named MyProduct that inherits the Product class. This new class should add the following method to the Product class:

   ```
   public String getPrice(NumberFormat nf)
   ```

   This method should format the price for the product using the format that's provided by the NumberFormat object that's passed as a parameter.
5. In the ProductDB class, modify the getProduct method so it returns a MyProduct object, not a Product object.
6. In the ProductApp class, modify the code so it uses a MyProduct object, not a Product object. This should include using the getPrice method that's only available from the MyProduct class to apply currency formatting to the price.
7. Run the application to make sure that it still works correctly.

Exercise 11-5 Work with an abstract class

In this exercise, you'll create an abstract class that contains an abstract method, and you'll modify other classes so they implement this abstract class.

Review the application

1. Import the project named ch11_ex5_ProductLister that's in the extra_ex_starts directory.
2. Open the classes and review their code.
3. Run the application to make sure it works correctly.

Create an abstract class and inherit it

4. In the murach.db package, add an abstract class named AbstractProductDB that contains the following abstract method:

```
public abstract Product get(String productCode);
```

5. In the ProductDB class, modify the code so that this class inherits the AbstractProductDB class.
6. Attempt to compile the application. This should display an error message that indicates that the ProductDB class must override the abstract get method.
7. In the ProductDB class, modify the code so it overrides the abstract get method.
8. In the Main class, modify the code so it calls the get method, not the getProduct method.
9. In the Main class, modify the code so it creates an instance of the AbstractProductDB class like this:

```
AbstractProductDB db = new ProductDB();
```

   This shows that the ProductDB class implements the abstract get method specified by the AbstractProductDB class.

10. Run the application to make sure it works correctly.

Modify another class so it inherits the abstract class

11. In the ProductDB2 class, modify the code so it inherits the AbstractProductDB class and implements its abstract get method.
12. In the Main class, modify the code so it uses ProductDB2, not ProductDB. This should be easy since both classes implement the abstract get method.
13. Run the application to make sure it works correctly.

Exercise 12-2 Work with an interface

In this exercise, you'll create two interfaces. Then, you'll implement them and use them.

Review the code

1. Import the project named ch12_ex2_ProductLister in the extra_ex_starts folder.
2. Review the code. Note that the Main class contains three constants for the width of the three columns of data.
3. Run the application to make sure that it works correctly.

Create an interface and implement it

4. In the murach.db package, create an interface named IProductDB. This interface should specify this abstract method:

```
public abstract Product get(String productCode);
```

5. Modify the ProductDB class so it implements the IProductDB interface.
6. In the Main class, modify the code so it works with the new ProductDB class. This code should create an instance of the IProductDB interface like this:

```
IProductDB db = new ProductDB();
```

   This shows that the ProductDB class implements the IProductDB interface.

7. Run the application to make sure it works correctly.

Modify another class so it implements the interface

8. In the ProductDB2 class, modify the code so it implements the IProductDB interface.
9. In the Main class, modify the code so it uses ProductDB2, not ProductDB. This should be easy since both classes implement the IProductDB interface.
10. Run the application to make sure it works correctly.

Create an interface that contains constants and use it

11. In the murach.db package, add an interface named IProductConstants that contains the three constants that are currently in the Main class. To do that, you can move the code that declares and initializes the constants from the Main class to the interface. In the Main class, make sure to remove the code that declares and initializes the three constants.
12. Attempt to compile the application. This should display an error message that indicates that constants aren't available to the Main class.
13. In the Main class, implement the IProductConstants interface. This should make the constants available to the Main class.
14. Run the application to make sure that it works correctly.

Exercise 13-4 Work with an enumeration and documentation

In this exercise, you'll create an enumeration. Then, you'll retrofit some existing code to use the enumeration.

Review the code

1. Import the project named ch13_ex4_CardSuitTester in the extra_ex_starts folder.
2. Review the code. Note that the Main class creates a Card object and sets its suit to 0 and its number to 1.
3. Run the application. It should display a message that says, "Ace of spades!".
4. In the Main class, modify the code so it sets the suit to 5.
5. Run the application again. Since there is no suit that corresponds to 5, the application should display a message that says, "Ace of ????!".

Create an enumeration and use it

6. In the murach.card package, create an enumeration named Suit. This enumeration should specify four suits: spades, hearts, diamonds, and clubs.
7. In the Card class, modify the code so it uses the Suit enumeration as the type for the suit instance variable.
8. In the Main class, modify the displayCard method so it uses the Suit enumeration to check the suit of the card.
9. In the Main class, experiment by modifying the code that sets the suit and number for the Cart object. Note that you can't set an illegal suit, but you can set an illegal number.
10. Run the application to make sure it works correctly.

Add javadoc comments and generate documentation

11. In the Card class, add javadoc comments. Make sure these comments include @param and @return tags wherever those tags are appropriate.
12. Generate the documentation for the project.
13. View the documentation for the Card class. This documentation should include descriptions for all of the constructors and methods as well as information about all parameters and return values.
14. View the documentation for the Suit enumeration. Note that this documentation includes a method named values returns an array of the constants that are in the Suit enumeration.

Exercise 14-3 Work with an array list of Circle objects

In this exercise, you'll modify the Circle Calculator so it works with two or more circles.

Review the application

1. Import the project named ch14_ex3_Circle in the extra_ex_starts directory.
2. Open the Main class and review its code. Note that it only creates one Circle object at a time.
3. Run the application to make sure it works correctly.

Modify the application so it stores circles in an array list

4. Before the while loop, add a statement that creates an array list of Circle objects.
5. Within the while loop, add code that adds that Circle object that's created for the specified radius to the array list.
6. After the while loop add another loop that loops through the array list of Circle objects and displays the data for each Circle object on the console. This data should include the radius. To do this, you can move the code that displays the data from the while loop to the second loop.
7. Run the application to make sure it works correctly. If the user enters 100 and 200, the console should look something like this:

```
Enter radius:  100
Continue? (y/n): y

Enter radius:  200
Continue? (y/n): n

Radius:        100.0
Area:          31415.899999999998
Circumference: 628.318
Diameter:      200.0

Radius:        200.0
Area:          125663.59999999999
Circumference: 1256.636
Diameter:      400.0
```

However, this should also work if the user enters one radius or if the user enters more than two radiuses.

Exercise 15-3 Add a start time and end time to the Number Guessing Game

In this exercise, you'll modify the Number Guessing Game so it stores a start time and an end time and calculates the number of seconds that it took the user to guess the number.

Review the application

1. Import the project named ch15_ex3_GuessingGame in the extra_ex_starts directory.
2. Open the classes and review their code. Note that they don't record the start and end time of the game.
3. Run the application to make sure it works correctly.

Store the start and end time of the game

4. In the NumberGame class, add instance variables of the LocalDateTime type for the start and end time of the game. Then, add get and set methods for these instance variables.
5. In the NumberGame class, add a method that returns the number of seconds between the start and end time. To do that, you can use code like the following code to get the number of seconds that have elapsed since January 1, 1970:

```
long startSeconds = startTime.toInstant(ZoneOffset.UTC).getEpochSecond();
```

   Then, you can use subtraction to determine the number of seconds between the start and end time.
6. In the Main class, store the start time in the NumberGame object just before the code prompts the user for the first number. Then, store the end time in the NumberGame object just after the code that displays the message that says, "Correct!".
7. In the Main class, add code that displays a message that contains the number of seconds just before the code that displays the message that says, "Bye!".
8. Run the application to make sure it works correctly. If it takes the user 20 seconds to guess the number, the console should display a message something like this:

```
You guessed the correct number in 20 seconds.
```

Exercise 16-4 Throw and catch custom exceptions

In this exercise, you'll modify the Customer Manager application so it handles exceptions differently. This application is similar to the Product Manager application described in chapter 17.

Review the application

1. Import the project named ch16_ex4_CustomerManager in the extra_ex_starts directory.
2. Open the CustomerTextFile class and the Main class and review the code. Note that the CustomerTextFile class handles all I/O exceptions, and the Main class doesn't catch any exceptions.
3. Run the application to make sure it works correctly.

Cause an exception to be thrown

4. In the CustomerTextFile class, add the following code to the beginning of the try block that's in the getCustomers method:

```
if (true) {
    throw new IOException("Test");
}
```

5. Run the application and test the exception handling. At this point, you should be able to test the exception handling for the list, add, and del commands. For these commands, the application should throw a NullPointerException and display a message like this:

```
java.io.IOException: Test
```

Create a custom exception and throw it

6. In the murach.io package, add a class that defines a custom exception named DBException. This exception should allow for exception chaining.
7. In the CustomerTextFile class, modify the getCustomers method so that the catch block creates a DBException object that stores the IOException object and throws the DBException object.
8. Add a throws clause that throws a DBException to all of the methods in the CustomerTextFile that need it.

Catch the custom exception

9. In the Main class, add a try/catch statement that catches any DBExceptions that are thrown by the CustomerTextFile class. In the catch clauses, display a user-friendly message.

10. Run the application and enter the list command. When you do, it should display a message like this:

```
Error! Unable to read customer file.
```

11. Use the add command to attempt to add a customer named John Doe. When you do, the application should display a message like this:

```
Error! Unable to add John Doe.
```

12. Use the del command to try to delete a customer for an email that exists in the file, the application should display a message like this:

```
Error! Unable to delete customer with email of johndoe@gmail.com.
```

Remove the code that causes an exception to be thrown

13. In the CustomerTextFile class, comment out the statement that throws the IOException.
14. Run the application to make sure it works correctly.

Exercise 17-3 Save rectangle data in a file

In this exercise, you'll modify the Area and Perimeter application so it stores the results of its calculations in a text file.

Review the application

1. Import the project named ch17_ex3_AreaAndPerimeter in the extra_ex_starts directory.
2. Review the code. Note that the Rectangle class provides methods to get the length, width, area, and perimeter for the rectangle.
3. Run the application to make sure it works correctly.

Create a file I/O class

4. Add a class named RectangeIO.
5. Add the following methods to the RectangleIO class:

```
public static void save(Rectangle r)
public static void printFile()
```

6. Add code to the save method so that it appends the length, width, area, and perimeter of the specified Rectangle object to a file named "rectangles.txt" that's stored in the root directory of the application.
7. Add code to the printFile method that reads each line of the text file for the application and prints each line to the console.

Use the file I/O class

8. In the Main class, add code to the while loop that uses the ProductIO class to save the Rectangle object.
9. After the code that prints the "Bye!" message, add code that uses the ProductIO class to print the contents of the text file for the application.
10. Run the application to make sure it works correctly. If this application has made four calculations in its lifetime, it should display data like this after the "Bye!" message:

```
RECTANGLES (length|width|area|perimeter):
100.0|200.0|20000.0|600.0
300.0|400.0|120000.0|1400.0
100.0|50.0|5000.0|300.0
100.0|75.0|7500.0|350.0
```

Exercise 18-3 Perform a long running task

In this exercise, you'll allow the main thread to run while a long running task runs on another thread in the background. When you've completed this exercise, your console should look similar to the following console. However, it might look slightly different depending on whether the thread manager selects the main thread or the long task thread to run first.

```
Main thread started.
Main thread still running.
Main thread running: 1
Long task thread started.
Main thread running: 2
Main thread running: 3
Main thread running: 4
Main thread running: 5
Main thread finished.
Long task thread finished.
```

Review the application

1. Import the project named ch18_ex3_LongTask in the extra_ex_starts folder.
2. Review the code for the Main class. Note that the main thread begins by performing a long task that takes at least 10 seconds. Then, it continues by counting from 1 to 5 with at least 1 second between each number.
3. Run the application to see how it works. Note that the long task prevents the counter in the main method from executing until the long task has finished.

Add a thread to the application

4. Add a class named TaskThread that implements the Runnable interface.
5. Move the code that performs the long task from the main method of the Main class into the run method of the TaskThread class.
6. In the Main class, use the Thread class to start the thread defined by the TaskThread class.
7. Run the application again to make sure it works correctly. Note that the counter in the main method starts immediately. That's because the run method of the TaskThread class is now running on its own thread.

Exercise 19-3 Create and run some SQL scripts

In this exercise, you'll get more practice working with the SQL statements you learned about in this chapter.

Start MySQL Workbench and examine the Product table

1. Start MySQL Workbench.
2. In the MySQL Connections section, click on the "Local instance MySQL" item. If prompted, enter the password for the root user. This should connect you to the local MySQL server as the root user.
3. Use the Navigator window to view the Schemas section and note which databases are installed on your system.
4. View the columns of the Product table by expanding the mma node, the Tables node, the Product node, and the Columns node.

Create scripts that modify one row at a time

5. Open the script named insert_product.sql that's in the ch19_ex3_SqlScripts folder of the extra_ex_starts folder. Note that it contains a USE statement that selects the database named mma as well as a SELECT statement that selects all rows from the Product table.
6. Run the script. It should select all rows from the Product table.
7. Before the SELECT statement, add an INSERT statement that inserts a new product into the database.
8. Run the script again. Make sure the new product you added appears in the result set.
9. Create a second script named delete_product.sql. This script should work like the first script, but it should use a DELETE statement to delete the product inserted by the first script.
10. Run the second script. Make sure the product you deleted in the previous step doesn't appear in the result set.

Create scripts that modify multiple rows at a time

11. Create a third script named insert_products.sql. This script should work like the first script, but it should insert at least two products into the database.
12. Run the script again. Make sure all of the new products you added appear in the result set.
13. Create a fourth script named delete_products.sql. This script should work like the second script, but it should delete all products where the ProductID column is greater than 7.

Exercise 20-3 Save rectangle data in a database

In this exercise, you'll modify the Area and Perimeter application so it stores the results of its calculations in a database.

Review the application

1. Import the project named ch20_ex3_AreaAndPerimeter in the extra_ex_starts directory.
2. Review the code. Note that the Rectangle class provides a constructor that can create a Rectangle object from its length and width.
3. Run the application to make sure it works correctly.

Create the Rectangle table in the database

4. Start MySQL Workbench and connect to the local MySQL server as the root user.
5. Open the script named create_rectangle_table.sql that's in the ch20_ex3_AreaAndPerimeter folder of the extra_ex_starts folder.
6. Review the code and note the name of the table and its columns.
7. Run the script. This should create the Rectangle table in the database named mma.

Create a database class

8. Add a MySQL database driver to the project.
9. Add the DBUtil class to the project. To do that, you can copy it from another project such as the ch20_ProductManager project in the book_apps directory.
10. In the DBUtil class, edit the methods so they don't throw the DBException. Instead, they can print the SQLException to the console.
11. Add a class named RectangeDB. Then, add the following methods to the RectangleDB class:

```
public static void insert(Rectangle r)
public static List<Rectangle> getAll()
```

12. Add code to the insert method so that it inserts a row for the specified Rectangle object into the Rectangle table.
13. Add code to the getAll method so that it returns an ArrayList object that contains one Rectangle object for each row in the Rectangle table.

Use the file DB class

14. In the Main class, add code to the while loop that uses the RectangleDB class to insert a Rectangle object.
15. After the code that prints the "Bye!" message, add code that uses the RectangleDB class to get an ArrayList of Rectangle objects. Then, add a loop that prints each Rectangle object to the console.

16. Run the application to make sure it works correctly. If this application has made four calculations in its lifetime, it should display data like this after the "Bye!" message:

```
RECTANGLES (length|width|area|perimeter):
100.0|200.0|20000.0|600.0
300.0|400.0|120000.0|1400.0
100.0|50.0|5000.0|300.0
100.0|75.0|7500.0|350.0
```

Exercise 20-4 Add a notes field to the Product Manager

In this exercise, you'll add a notes field to the Product Manager application. This field can store a note of up to 65,535 characters.

Review the application

1. Import the project named ch20_ex4_ProductManager in the extra_ex_starts directory.
2. Review the code. Then, run the application to make sure it works correctly.

Add a Note column to the Product table

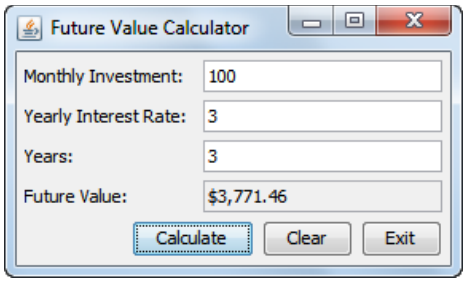
3. Start MySQL Workbench and connect to the local MySQL server as the root user.
4. Open the script named alter_product_table.sql that's in the ch20_ex4_ProductManager folder of the extra_ex_starts folder.
5. Review the code. It should (1) select the mma database, (2) add a Note column to the Product table, and (3) set the Note column to an empty string for the first seven products.
6. Run the script.

Modify the Java classes

7. Open the Product class and add a new instance variable named note. Then, add get and set methods for this instance variable.
8. Open the ProductDB class and add code to the getProductFromRow method that sets the note in the Product object.
9. Update the add method so that it adds the note to the database.
10. Update the update method so that it updates the note in the database. Hint: Don't forget to change the parameter number for the product ID.
11. Open the Main method and add code to the displayAllProducts method that displays the notes field of the Product class.
12. Add code to the addProduct method to add a note to the product.
13. Add code to the updateProduct method to update the note for the product.
14. Run the application and to make sure the new code works correctly. Create a new product that has a note. Display all of the products and make sure you can see the notes. Update the note for an existing product. And so on.

Exercise 22-2 Add a button to a GUI

In this exercise, you'll add a Clear button to the GUI version of the Future Value Calculator. When you are finished with this exercise, your application should look like this:



Review the application

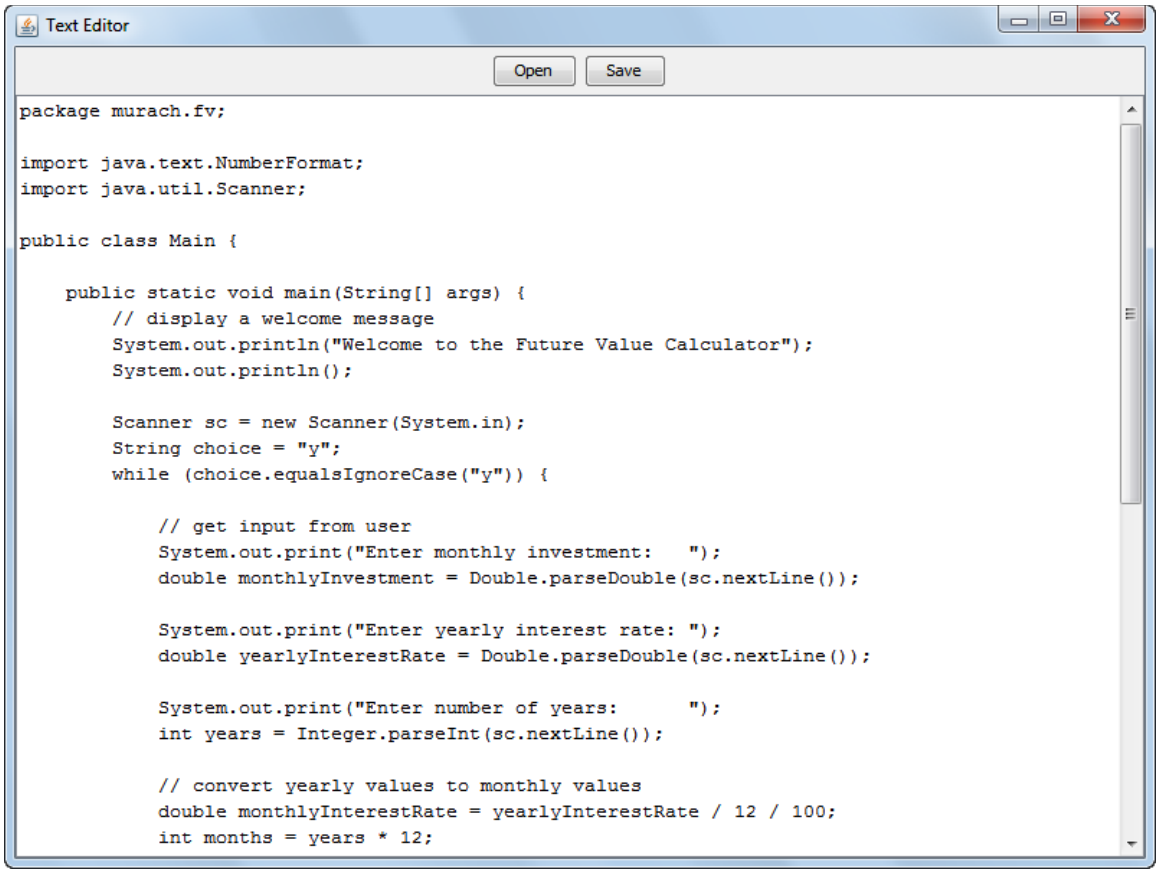
1. Import the project named ch22_ex2_FutureValue in the extra_ex_starts folder.
2. Review the code in the FutureValueFrame class.
3. Run the application to make sure it works correctly.

Add a Clear button

4. Declare a private instance variable for a JButton component for the Clear button.
5. In the initComponents method, add the code that creates the button and adds it to the frame.
6. Add an action listener to the Clear button. This action listener should set the text for each text field on the frame to an empty string.
7. Run the application to make sure it works correctly. After you make a calculation, you should be able to click the Clear button to clear the text from all text fields on the frame.

Exercise 22-3 Modify a text editor

In this exercise, you'll add code to the beginnings of a basic, but functional, text editor. When you are finished with this exercise, your application should look like this:



Review the application

8. Import the project named ch22_ex3_TextEditor in the extra_ex_starts folder.
9. Review the code in the MainWindow class. Note that the doOpenButton method uses a JFileChooser component. This method displays a Swing dialog that allows the user to select a file from the file system. Then, it prints the contents of the selected file to the console.
10. Run the application to see how it works. It should display a frame that doesn't have any components on it.

Add buttons to the frame

11. In the buildButtonPanel method, add an Open button and a Save button to the panel. Then, add tooltips to the buttons.
12. Add action listeners to the Open button and Save button that invoke the doOpenButton and doSaveButton methods.

13. In the constructor for the frame, add the button panel to the north area of the BorderLayout for the frame. To do that, you can call the buildButtonPanel method to get the panel for the buttons.
14. Run the application to make sure it works correctly. When you use the Open button to open a text file on your system, it should print the contents of the file to the console. When you use the Save button to save a file, it should print a message to the console that indicates that the file has been saved.

Add a JTextArea control to the frame

15. In the MainWindow class, add an instance variable for a JTextArea component named textArea. Although the JTextArea component isn't covered in this book, it works similarly to the JTextField component, except that it allows multiple lines of text.
16. In the constructor, create a new instance of the JTextArea and add it to the center area of the BorderLayout for the frame.
17. In the doOpenButton method, find the line of code that prints the contents of the file to the console. Replace this line with code that sets the text of the JTextArea component to the contents of the file.
18. In the doSaveButton method, add a line of code that sets the fileContents variable to the text that's in the JTextArea component.
19. Run the application to make sure it works correctly. You should be able to open a text file, edit it, and save those changes. However, if you open a text file that's too large to fit on the screen, it will disappear off the bottom of the application.

Add a JScrollPane control to the frame

20. To fix this problem, add the JTextArea to a JScrollPane and then add the JScrollPane to the center area of the BorderLayout for the frame.
21. Run the application to make sure it works correctly. Now, if you open a large text file, scrollbar should appear that allow you to scroll through the file.