Student projects for *Murach's Beginning Java with Eclipse*

The projects in this document let your students apply the programming skills they'll learn as they progress through *Murach's Beginning Java with Eclipse*. If you review these projects, you'll see that they represent different levels of difficulty, so you can assign projects that are appropriate for the skill levels of your students. In addition, you can easily modify the projects to make them more or less challenging.

In the project name, the first number specifies the chapter that the student should complete before starting the exercise. For example, the student should complete chapter 3 before starting project 3-1 or 3-2, and the student should complete chapter 7 before starting project 7-1, 7-2, or 7-3.

Project 3-1: Convert number grades to letter grades 2
Project 3-2: Convert temperature 3
Project 4-1: Display customer information 4
Project 4-2: Use a class to store grade data 5
Project 7-1: Calculate travel time 6
Project 7-2: Calculate interest 7
Project 7-3: Calculate coins for change 8
Project 8-1: Display a table of powers 9
Project 8-2: Calculate the monthly payment on a loan 10
Project 8-3: Roll the dice 12
Project 10-1: Display sales report 14
Project 10-2: Translate English to Pig Latin 15
Project 11-1: Work with customer and employee data 16
Project 12-1: Calculate a monthly balance 18
Project 13-1: Test and document the Console class 20
Project 13-2: Create a Roshambo game 22
Project 14-1: Store email addresses and phone numbers 24
Project 14-2: List movies by category 26
Project 15-1: Calculate reservation totals 27
Project 16-1: View customer data 28
Project 17-1: Check if a path exists 29
Project 17-2: Convert lengths 30
Project 18-1: Tortoise and the hare race 32
Project 20-1: Manage a list of countries 34
Project 20-2: Manage customer data 36
Project 22-1: Calculate the hypotenuse of a right triangle 38
Project 22-2: Validate user entries 39
Project 22-3: Manage customer data (GUI) 40
More ideas for projects 41

Project 3-1: Convert number grades to letter grades

Console

```
Welcome to the Letter Grade Converter

Enter numerical grade: 90
Letter grade: A

Continue? (y/n): y

Enter numerical grade: 88
Letter grade: A

Continue? (y/n): y

Enter numerical grade: 80
Letter grade: B

Continue? (y/n): y

Enter numerical grade: 67
Letter grade: C

Continue? (y/n): y

Enter numerical grade: 59
Letter grade: F

Continue? (y/n): n
```

Operation

- The user enters a numerical grade from 0 to 100.
- The application displays the corresponding letter grade.
- The application prompts the user to continue.

Specifications

- The grading criteria is as follows:

```
A    88-100
B    80-87
C    67-79
D    60-67
F    <60
```

- Assume that the user will enter valid integers for the grades.
- The application should continue only if the user enters “y” or “Y” to continue.

Project 3-2: Convert temperature

Console

```
Welcome to the Temperature Converter

Enter degrees in Fahrenheit: 212
Degrees in Celsius: 100

Continue? (y/n): y

Enter degrees in Fahrenheit: 32
Degrees in Celsius: 0

Continue? (y/n): y

Enter degrees in Fahrenheit: 77.5
Degrees in Celsius: 25.28

Continue? (y/n): n
```

Operation

- The application prompts the user to enter a temperature in Fahrenheit degrees.
- The application displays the temperature in Celsius degrees.
- The application prompts the user to continue.

Specifications

- The formula for converting temperatures from Fahrenheit to Celsius is:

  ```
  c = (f-32) * 5/9
  ```

  Note: The parentheses are necessary to control the order of precedence!
- The application should accept decimal entries like 77.5.
- Assume that the user will enter valid data.
- The application should continue only if the user enters “y” or “Y” to continue.

Project 4-1: Display customer information

Console

```
Welcome to the Customer Viewer

Enter a customer number: 1003

Ronda Chavan
518 Commanche Dr.
Greensboro, NC 27410

Display another customer? (y/n): y

Enter a customer number: 2439

There is no customer number 2439 in our records.

Display another customer? (y/n): n
```

Operation

- The application prompts the user to enter a customer number. If a customer with that number exists, the application displays the customer's name and address. If not, the application displays an appropriate message that includes the customer number.

Specifications

- Create a class named Customer that has instance variables that store the name, address, city, state, and postalCode for the customer. This class should have get and set methods that provide access to all instance variables. In addition, the class should have a method named getNameAndAddress that returns the name and address formatted as shown above.
- Create a class named CustomerDB class that contains a static method named getCustomer that accepts a customer number (an int value) and returns a Customer object. Within this class, code an if statement that returns this data:

```
1001
Barbara White
3400 Richmond Parkway #3423
Bristol, CT 06010

1002
Karl Vang
327 Franklin Street
Edina, MN 55435

1003
Ronda Chavan
518 Commanche Dr.
Greensboro, NC 27410
```

Project 4-2: Use a class to store grade data

Console

```
Welcome to the Letter Grade Converter

Enter numerical grade: 90
Letter grade: A

Continue? (y/n): y

Enter numerical grade: 88
Letter grade: A

Continue? (y/n): y

Enter numerical grade: 80
Letter grade: B

Continue? (y/n): y

Enter numerical grade: 67
Letter grade: C

Continue? (y/n): y

Enter numerical grade: 59
Letter grade: F

Continue? (y/n): n
```

Operation

- The user enters a number grade from 0 to 100.
- The application displays the corresponding letter grade.
- The application prompts the user to continue.

Specifications

- This application should use a class named Grade to store the data for each grade. This class should include these three methods:

```
void setNumber(int number)
int getNumber()
String getLetter()
```

- The grading criteria is as follows:

```
A    88-100
B    80-87
C    67-79
D    60-67
F    <60
```

- Assume that the user will enter valid integers for the grades.
- The application should continue only if the user enters “y” or “Y” to continue.

Project 7-1: Calculate travel time

Console

```
Welcome to the Travel Time Calculator

Enter miles:           200
Enter miles per hour:  65

Estimated travel time
Hours:   3
Minutes: 4

Continue? (y/n): y

Enter miles:           100
Enter miles per hour:  65

Estimated travel time
Hours:   1
Minutes: 32

Continue? (y/n): n
```

Operation

- The application prompts the user to enter values for miles and miles per hour.
- The application displays the approximate travel time in hours and minutes.
- The application prompts the user to continue.

Specifications

- The application should accept decimal entries like 10.5 and 20.65.
- Assume that the user will enter valid data.
- The application should continue only if the user enters “y” or “Y” to continue.

Hint

- Use integers with the division and modulus operators to get hours and minutes.

Project 7-2: Calculate interest

Console

```
Welcome to the Interest Calculator

Enter loan amount:   520000
Enter interest rate: .05375

Loan amount:         $520,000.00
Interest rate:       5.375%
Interest:            $27,950.00

Continue? (y/n): y

Enter loan amount:   4944.5
Enter interest rate: .01

Loan amount:         $4,944.50
Interest rate:       1%
Interest:            $49.45

Continue? (y/n): n
```

Operation

- The application prompts the user to enter a loan amount and an interest rate.
- The application calculates the interest amount and formats the loan amount, interest rate, and interest amount. Then, it displays the formatted results to the user.
- The application prompts the user to continue.

Specifications

- This application should use the BigDecimal class to make sure that all calculations are accurate. It should round the interest that's calculated to two decimal places, rounding up if the third decimal place is five or greater.
- The value for the formatted interest rate should allow for up to 3 decimal places.
- Assume that the user will enter valid double values for the loan amount and interest rate.
- The application should continue only if the user enters "y" or "Y" to continue.

Project 7-3: Calculate coins for change

Console

```
Welcome to the Change Calculator

Enter number of cents (0-99): 99

Quarters: 3
Dimes:    2
Nickels:  0
Pennies:  4

Continue? (y/n): y

Enter number of cents (0-99): 55

Quarters: 2
Dimes:    0
Nickels:  1
Pennies:  0

Continue? (y/n): n
```

Operation

- The application prompts the user to enter a number of cents from 0 to 99.
- The application displays the minimum number of quarters, dimes, nickels, and pennies that represent the coins that make up the specified number of cents.
- The application prompts the user to continue.

Specifications

- Assume that the user will enter a valid integer value for the number of cents.
- The application should continue only if the user enters “y” or “Y” to continue.

Project 8-1: Display a table of powers

Console

```
Welcome to the Squares and Cubes table

Enter an integer: 9

Number  Squared Cubed
======  ======= =====
1       1       1
2       4       8
3       9       27
4       16      64
5       25      125
6       36      216
7       49      343
8       64      512
9       81      729

Continue? (y/n): y

Enter an integer: 3

Number  Squared Cubed
======  ======= =====
1       1       1
2       4       8
3       9       27

Continue? (y/n): n
```

Operation

- The application prompts the user to enter an integer.
- The application displays a table of squares and cubes from 1 to the value entered by the user.
- The application prompts the user to continue.

Specifications

- The formulas for calculating squares and cubes are:

```
square = x * x
cube = x * x * x
```

- Assume that the user will enter a valid integer.
- The application should continue only if the user enters “y” or “Y” to continue.

Project 8-2: Calculate a monthly payment

Console

```
Welcome to the Loan Calculator

DATA ENTRY
Enter loan amount:          ten
Error! Invalid decimal. Try again.
Enter loan amount:          -1
Error! Number must be greater than 0.0
Enter loan amount:          100000000000
Error! Number must be less than 1000000.0
Enter loan amount:          500000
Enter yearly interest rate: 5.6
Enter number of years:      thirty
Error! Invalid integer value. Try again.
Enter number of years:      -1
Error! Number must be greater than 0
Enter number of years:      100
Error! Number must be less than 100
Enter number of years:      30

FORMATTED RESULTS
Loan amount:          $500,000.00
Yearly interest rate: 5.6%
Number of years:      30
Monthly payment:      $2,870.39

Continue? (y/n):
Error! This entry is required. Try again.
Continue? (y/n): x
Error! Entry must be 'y' or 'n'. Try again.
Continue? (y/n): n
```

Operation

- The Data Entry section prompts the user to enter values for the loan amount, yearly interest rate, and number of years. If the user doesn't enter data that's valid, this section displays an appropriate error message and prompts the user again.
- The Formatted Results section displays a formatted version of the user's entries as well as the formatted result of the calculation.
- The application prompts the user to continue.

Project 8-2: Calculate a monthly payment (cont.)

Specifications

- The formula for calculating monthly payment is:

```
double monthlyPayment =
    loanAmount * monthlyInterestRate/
    (1 - 1/Math.pow(1 + monthlyInterestRate, months));
```

- The application should accept decimal entries for the loan amount and interest rate entries.
- The application should only accept integer values for the years field.
- The application should only accept integer and decimal values within the following ranges:

	Greater Than	Less Than
Loan amount:	0	1,000,000
Yearly interest rate:	0	20
Years:	0	100

- The application should only accept a value of "y" or "n" at the Continue prompt.
- If the user enters invalid data, the application should display an appropriate error message and prompt the user again until the user enters valid data.
- Use a Console class like the one described in chapter 8 to make sure the user enters valid double and int values. To do that, you need to add more methods. For example, you can to add the following methods to specify the min and max values for an entry:

```
public static double getDouble(String prompt, double min, double max)

public static int getInt(String prompt, int min, int max)
```

Project 8-3: Roll the dice

Console

```
Welcome to the Paradise Roller

Roll the dice? (y/n): y

Roll 1:
2
5
Craps!

Roll again? (y/n): y

Roll 2:
2
1

Roll again? (y/n): y

Roll 3:
4
6

Roll again? (y/n): y

Roll 4:
6
6
Box cars!

Roll again? (y/n): y

Roll 5:
1
1
Snake eyes!

Roll again? (y/n): n
```

Operation

- If the user chooses to roll the dice, the application rolls two six-sided dice, displays the results of each, and asks if the user wants to roll again.

Project 8-3: Roll the dice (cont.)

Specifications

- Create a class named Die to store the data about each die. This class should contain these constructors and methods:

```
public Die()              // default to a six-sided die
public Die(int sides)     // allow a variable number of sides
public void roll()
public int getValue()
```

- Create a class named PairOfDice to store two dice. This class should contain two instance variables of the Die type, an instance variable that holds the sum of the two dice, and these constructors and methods:

```
public PairOfDice()           // default to six-sided dice
public PairOfDice(int sides)  // allow a variable number of sides
public void roll()
public int getValue1()        // get value of die1
public int getValue2()        // get value of die2
public int getSum()           // get the sum of both dice
```

- You can use the random method of the Math class to generate a random number from 1 to the number of sides on a die like this:

```
int value = (int) (Math.random() * sides);
```

- Create a class named DiceRollerApp that uses the PairOfDice class to roll the dice. This class should display special messages for craps (sum of both dice is 7), snake eyes (double 1's), and box cars (double 6's). For this application, assume that two six-sided dice are used.

Project 10-1: Display sales report

Console

```
Welcome to the Sales Report

Region  Q1              Q2              Q3              Q4
1       $1,540.00       $2,010.00       $2,450.00       $1,845.00
2       $1,130.00       $1,168.00       $1,847.00       $1,491.00
3       $1,580.00       $2,305.00       $2,710.00       $1,284.00
4       $1,105.00       $4,102.00       $2,391.00       $1,576.00

Sales by region:
Region 1: $7,845.00
Region 2: $5,636.00
Region 3: $7,879.00
Region 4: $9,174.00

Sales by quarter:
Q1: $5,355.00
Q2: $9,585.00
Q3: $9,398.00
Q4: $6,196.00

Total annual sales, all regions: $30,534.00
```

Operation

- This application displays a four-section report of sales by quarter for a company with four sales regions (Region 1, Region 2, Region 3, and Region 4).
- The first section of the report lists the sales by quarter for each region.
- The second section summarizes the total annual sales by region.
- The third section summarizes the total annual sales by quarter for all regions.
- The fourth section prints the total annual sales for all sales regions.

Specifications

- The quarterly sales for each region should be hard-coded into the program using the numbers shown in the console output above. The sales numbers should be stored in a rectangular array.
- The first section of the report should use nested for loops to display the sales by quarter for each region. Use tabs to line up the columns for this section of the report.
- The second section of the report should use nested for loops to calculate the sales by region by adding up the quarterly sales for each region.
- The third section of the report should use nested for loops to calculate the sales by quarter by adding up the individual region sales for each quarter.
- The fourth section of the report should use an enhanced for loop to calculate the total annual sales for all regions.
- Use the NumberFormat class to format the sales numbers using the currency format.

Project 10-2: Translate English to Pig Latin

Console

```
Welcome to the Pig Latin Translator.

Enter a line to be translated to Pig Latin:
this program translates from english to pig latin

isthay ogrampray anslatestray omfray englishway otay igpay atinlay

Translate another line? (y/n): n
```

Operation

- The application prompts the user to enter a line of text.
- The application translates the text to Pig Latin and displays it on the console.
- The program asks the user if he or she wants to translate another line.

Specifications

- Parse the string into separate words before translating. You can assume that the words will be separated by a single space and there won't be any punctuation. To do that, you can use the split function of the String object like this:

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

- Convert each word to lowercase before translating.
- If the word starts with a vowel, just add *way* to the end of the word.
- If the word starts with a consonant, move all of the consonants that appear before the first vowel to the end of the word, then add *ay* to the end of the word.
- If a word starts with the letter *y*, the *y* should be treated as a consonant. If the *y* appears anywhere else in the word, it should be treated as a vowel.
- Check that the user has entered text before performing the translation.

Notes

- This application requires the use of string handling to parse the input string into separate words, to analyze letters at the beginning of each word, to identify consonants and vowels, and to add Pig Latin word endings.
- There are no official rules for Pig Latin. Most people agree on how words that begin with consonants are translated, but there are many different ways to handle words that begin with vowels.

Project 11-1: Work with customer and employee data

Console

```
Welcome to the Person Tester application

Create customer or employee? (c/e): c

Enter first name: Frank
Enter last name: Jones
Enter email address: frank44@hotmail.com
Customer number: M10293

You entered:
Name: Frank Jones
Email: frank44@hotmail.com
Customer number: M10293

Continue? (y/n): y

Create customer or employee? (c/e): e

Enter first name: Anne
Enter last name: Prince
Enter email address: anne@murach.com
Social security number: 111-11-1111

You entered:
Name: Anne Prince
Email: anne@murach.com
Social security number: 111-11-1111

Continue? (y/n): n
```

Operation

- The application prompts the user to enter a customer or an employee.
- If the user selects customer, the application asks for name, email, and customer number.
- If the user selects employee, the application asks for name, email, and social security number.
- When the user finishes entering data for a customer or employee, the application displays the data that the user entered.

Project 11-1: Work with customer and employee data (cont.)

Specifications

- Create an abstract Person class that stores first name, last name, and email address. This class should provide a no-argument constructor, get and set methods for each instance variable, and it should override the toString method so it returns the first name, last name, and email fields in this format:

```
Name: Frank Jones
Email: frank44@hotmail.com
```

  In addition, it should contain an abstract method named getDisplayText that returns a string.

- Create a class named Customer that inherits the Person class. This class should store a customer number, it should provide get and set methods for the customer number, it should provide a no-argument constructor, and it should provide an implementation of the getDisplayText method. The getDisplayText method should return a string that consists of the string returned by the toString method of the Person class appended with the Customer number like this:

```
Name: Frank Jones
Email: frank44@hotmail.com
Customer number: M10293
```

- Create a class named Employee that inherits the Person class. This class should store a social security number, it should provide get and set methods for the social security number, it should provide a no-argument constructor, and it should provide an implementation of the getDisplayText method. The getDisplayText method should return a string that consists of the string returned by the toString method of the Person class appended with the Employees social security number like this:

```
Name: Anne Prince
Email: anne@murach.com
Social security number: 111-11-1111
```

- Create a class named PersonApp that prompts the user as shown in the console output. This class should create the necessary Customer and Employee objects from the data entered by the user, and it should use these objects to display the data to the user. To print the data for an object to the console, this application should use a static method named print that accepts a Person object.
- Use the Console class from chapter 8 or a variation of it to get entries from the user.

Project 12-1: Calculate a monthly balance

Console

```
Welcome to the Account Calculator

Starting Balance
Checking: $1,000.00

Enter the transactions for the month

Withdrawal or deposit? (w/d): w
Amount: 500

Continue? (y/n): y

Withdrawal or deposit? (w/d): d
Amount: 200

Continue? (y/n): n

Monthly Fees
Checking fee:              $1.00

Final Balance
Checking: $699.00
```

Operation

- The application begins by displaying the starting balance for a checking account.
- The application prompts the user to enter the amount for a withdrawal or deposit.
- When the user finishes entering deposits and withdrawals, the application displays the fees for the month followed by the final balances for the month.

Project 12-1: Calculate a monthly balance (cont.)

Specifications

- Create interfaces named Depositable, Withdrawable, and Balanceable that specify the methods that can be used to work with accounts. The Depositable interface should include this method:

```
void deposit(double amount)
```

  The Withdrawable interface should include this method:

```
void withdraw(double amount)
```

  And the Balanceable interface should include these methods:

```
double getBalance()
void setBalance(double amount)
```

- Create a class named Account that implements all three of these interfaces. In addition, it should supply a method like the following method that returns a balance that has been formatted as currency:

```
String getBalanceFormatted()
```

- Create a class named CheckingAccount that inherits the Account class. This class should include an instance variable for the monthly fee and these methods:

```
void subtractMonthlyFee()
void setMonthlyFee(double monthlyFee)
double getMonthlyFee()
String getMonthlyFeeFormatted()
```

  By default, the monthly fee for a checking account should be $1.

- Create a class named Transactions that contains the following static methods for depositing and withdrawing funds from either type of account:

```
public static void deposit(Depositable account, double amount) {
    account.deposit(amount);
}

public static void withdraw(Withdrawable account, double amount) {
    account.withdraw(amount);
}
```

- Create a class named AccountApp that prompts the user for a transaction, posts the transaction, and displays the information shown in the console output. Create the necessary objects for each transaction, and post the transaction using the appropriate method of the Transactions class.
- Use the Console class presented in chapter 8 or a variation of it to get entries from the user.
- This application should not allow the user to withdraw more than the current account balance.
- This application should not allow the user to deposit more than $10,000 per transaction.

Project 13-1: Test and document the Console class

Console

```
Welcome to the Console Tester application

Int Test
Enter an integer between -100 and 100:
Error! This entry is required. Try again.
Enter an integer between -100 and 100: x
Error! Invalid integer value. Try again.
Enter an integer between -100 and 100: -101
Error! Number must be greater than -101
Enter an integer between -100 and 100: 101
Error! Number must be less than 101
Enter an integer between -100 and 100: 50

Double Test
Enter any number between -100 and 100:
Error! This entry is required. Try again.
Enter any number between -100 and 100: x
Error! Invalid decimal value. Try again.
Enter any number between -100 and 100: -101
Error! Number must be greater than -101.0
Enter any number between -100 and 100: 101
Error! Number must be less than 101.0
Enter any number between -100 and 100: 50

Required String Test
Enter your email address:
Error! This entry is required. Try again.
Enter your email address: joelmurach@yahoo.com

String Choice Test
Select one (x/y):
Error! This entry is required. Try again.
Select one (x/y): q
Error! Entry must be 'x' or 'y'. Try again.
Select one (x/y): x
```

Operation

- This application prompts the user to enter a valid integer within a specified range, a valid double within a specified range, a required string, and one of two strings. If a user entry isn't valid, the application displays an appropriate error message.

Project 13-1: Test and document the Console class (cont.)

Specifications

- Create a class named Console that can be used to display output to the user and get input from the user. Feel free to reuse your best code from any previous exercises or projects. At a minimum, this class should include these methods:

```
// for output
public void print(String s);
public void println(String s);
public void println();

// for input
public String getString(String prompt);
public String getRequiredString(String prompt);
public String getChoice(String prompt, String s1, String s2);
public int getInt(String prompt);
public int getIntWithinRange(String prompt, int min, int max);
public double getDouble(String prompt);
public double getDouble(String prompt, double min, double max);
```

- Create a class named ConsoleTestApp that tests the Console application to make sure it's working correctly as shown in the console output.
- Store the Console class in a package named

  `yourLastName.util`

- Add javadoc comments to the Console class. These comments should document the purpose, author, and version of the class. It should also document the purpose of each method, including any parameters accepted by the method and any value it returns.
- Generate the documentation for this project and store it in the javadoc subdirectory of the dist directory.

Project 13-2: Create a Roshambo game

Console

```
Welcome to the game of Roshambo

Enter your name: Joel

Would you like to play Bart or Lisa? (b/l): b

Rock, paper, or scissors? (r/p/s): r

Joel: rock
Bart: rock
Draw!

Play again? (y/n): y

Rock, paper, or scissors? (r/p/s): p

Joel: paper
Bart: rock
Joel wins!

Play again? (y/n): y

Rock, paper, or scissors? (r/p/s): s

Joel: scissors
Bart: rock
Bart wins!

Play again? (y/n): n
```

Operation

- The application prompts the player to enter a name and select an opponent.
- The application prompts the player to select rock, paper, or scissors. Then, the application displays the player's choice, the opponent's choice, and the result of the match.
- The application continues until the user doesn't want to play anymore.
- If the user makes an invalid selection, the application should display an appropriate error message and prompt the user again until the user makes a valid selection.

Project 13-2: Create a Roshambo game (cont.)

Specifications

- Create an enumeration named Roshambo that stores three values: rock, paper, and scissors. This enumeration should include a toString method that can convert the selected value to a string.
- Create an abstract class named Player that stores a name and a Roshambo value. This class should include an abstract method named generateRoshambo that allows an inheriting class to generate and return a Roshambo value. It should also include get and set methods for the name and Roshambo value.
- Create classes named Bart and Lisa that inherit the Player class and implement the generateRoshambo method. The Bart class should always select rock. The Lisa class should randomly select rock, paper, or scissors (a 1 in 3 chance of each).
- Create a class named Player1 that inherits the Player class and implements the generateRoshambo method (even though it isn't necessary for this player). This method can return any value you choose.
- Create a class named RoshamboApp that allows the player to play Bart or Lisa as shown in the console output. Rock should beat scissors, paper should beat rock, and scissors should beat paper.
- Use the Console class described in chapter 8 or a variation of it to get the user's entries.

Enhancement

- Keep track of wins and losses and display them at the end of each session.

Project 14-1: Store email addresses and phone numbers

Console

```
Welcome to the Address Book application

1 - List entries
2 - Add entry
3 - Exit

Enter menu number: 1

Name                Email                Phone
------------------  -------------------  ------------------
Larry Elison        larry@oracle.com     (444) 555-6666
Sergey Brin         sergey@gmail.com     (415) 222-3333

1 - List entries
2 - Add entry
3 - Exit

Enter menu number: 2

Enter name: Joel Murach
Enter email address: joel@murach.com
Enter phone number: (415) 123-4567

This entry has been saved.

1 - List entries
2 - Add entry
3 - Exit

Enter menu number: 1

Name                Email                Phone
------------------  -------------------  ------------------
Larry Elison        larry@oracle.com     (444) 555-6666
Sergey Brin         sergey@gmail.com     (415) 222-3333
Joel Murach         joel@murach.com      (415) 123-4567

1 - List entries
2 - Add entry
3 - Exit

Enter menu number: 3

Goodbye!
```

Operation

- If the user selects the first menu option, the application displays the email addresses and phone numbers that have been saved. Then, it displays the menu again.
- If the user selects the second menu option, the application prompts the user to enter a name, email address, and phone number. Then, it displays the menu again.
- If the user selects the third menu option, the application exits.

Project 14-1: Store email addresses and phone numbers (cont.)

Specifications

- Use the AddressBookEntry class that's provided to store the data for each entry.
- Use the AddressBookIO class that's provided to get and save entries. This class contains two static methods that you can use to read and write data to the address_book.txt file. They are:

```
// get a String that displays all entries in columns
public static ArrayList<AddressBookEntry> getEntries ()

// save an AddressBookEntry object to the file
public static boolean saveEntry(AddressBookEntry entry)
```

  If necessary, you can open the address_book.txt file in a text editor to debug this application.
- Create a class named AddressBookApp. This class should display the menu and respond to the user's menu choices using the AddressBookEntry and AddressBookIO classes as necessary.
- Use the Console class described in chapter 8 or a variation of it to get the user's entries.
- Use the String Util class described in chapter 9 to align data that's displayed by the application.

Project 14-2: List movies by category

Console

```
Welcome to the Movie Lister

There are 100 movies in the list.

What category are you interested in? scifi

Star Wars
2001: A Space Odyssey
E.T. The extra-terrestrial
A Clockwork Orange
Close Encounters Of The Third Kind

Continue? (y/n): y

What category are you interested in? comedy

Annie Hall
M*A*S*H
Tootsie
Duck Soup

Continue? (y/n): n
```

Operation

- This application stores a list of 100 movies and displays them by category.
- The user can enter any of the following categories to display the films in the list that match the category:

  animated
  drama
  horror
  musical
  scifi

- After each list is displayed, the user is asked whether to continue. If the user enters Y or y, the program asks for another category. Otherwise, the program ends.

Specifications

- Use the Movie class that's provided to store the title and category for each movie.
- Use the MovieDB class that's provided to you to get the ArrayList objects. To do that, you'll need to finish the code for the getMovies methods.

Possible enhancement

- Display a menu of category choices and ask the user to select the category like this:

```
1. Animated
2. Drama
3. Horror
4. Musical
5. Scifi

Enter category number:
```

Project 15-1: Calculate reservation totals

Console

```
Welcome to the Reservation Calculator

Enter the arrival month (1-12): 5
Enter the arrival day (1-31): 16
Enter the arrival year: 2015

Enter the departure month (1-12): 5
Enter the departure day (1-31): 18
Enter the departure year: 2015

Arrival Date: May 16, 2015
Departure Date: May 18, 2015
Price: $115.00 per night
Total price: $230.00 for 2 nights

Continue? (y/n): n
```

Operation

- This application calculates the charges for a stay at a hotel based on the arrival and departure dates.
- The application prompts the user for the month, day, and year of the arrival and the departure. Then, the application displays the arrival date, the departure date, the room rate, the total price, and the number of nights.

Specifications

- Create a class named Reservation that defines a reservation. This class should contain instance variables for the arrival date and departure date. It should also contain a constant initialized to the nightly rate of $115.00.
- The Reservation class should include the following methods:

```
LocalDate getArrivalDate()
String getArrivalDateFormatted()
setArrivalDate(LocalDate arrivalDate)
LocalDate getDepartureDate()
String getDepartureDateFormatted()
setDepartureDate(LocalDate departureDate)
int getNumberOfNights()
String getPricePerNightFormatted()
double getTotalPrice()
String getTotalPriceFormatted()
```

- To calculate the total number of nights, you can use the toEpochDay method to get the number of days since Jan 1, 1970 for the arrival and departure dates. Then, you can use normal arithmetic operators.

Possible enhancement

- Allow the user to enter the date in the form *mm*/*dd*/*yyyy*.

Project 16-1: View customer data

Console

```
Welcome to the Customer Viewer

Enter a customer number: 1003

Ronda Chavan
518 Comanche Dr.
Greensboro, NC 27410

Display another customer? (y/n): y

Enter a customer number: 2439

There is no customer with a number of 2439

Display another customer? (y/n): n
```

Operation

- The application prompts the user to enter a customer number.
- If a customer exists for that number, the application displays the customer's name and address.
- If no customer exists for that number, the application displays an appropriate message that includes the customer number.

Specifications

- Use the Customer class that's provided to work with customer data.
- Use the CustomerDB class that's provided to get a Customer object that corresponds with the specified customer number.
- Create a NoSuchCustomerException class that can store a message.
- Modify the getCustomer method of the CustomerDB class so that it throws a NoSuchCustomerException if no customer exists for the specified number. The message for this exception should include the specified customer number.

Project 17-1: Check if a path exists

Console

```
Welcome to the Path Checker

Enter a path: /murach

That path points to a directory.

Continue? (y/n): y

Enter a path: /murach/java_eclipse/files/products.txt

That path points to a regular file.

Continue? (y/n): y

Enter a path: /bad

That path does not exist.

Continue? (y/n): n
```

Operation

- The application prompts the user to enter a path. Then, the application checks whether the path exists on the current computer. If so, the application checks whether the path is a directory or a file and displays an appropriate message. Otherwise, it displays a message that indicates that the path doesn't exist.

Project 17-2: Convert lengths

Console

```
Welcome to the Length Converter

1 - Convert a length
2 - Add a type of conversion
3 - Delete a type of conversion
4 - Exit

Enter menu number: 1

1 -  Miles to Kilometers: 1.6093
2 -  Kilometers to Miles: 0.6214
3 -  Inches to Centimeters: 2.54

Enter conversion number: 2

Enter Kilometers: 10
10.0 Kilometers = 6.214 Miles

1 - Convert a length
2 - Add a type of conversion
3 - Delete a type of conversion
4 - Exit

Enter menu number: 2

Enter 'From' unit: Centimeters
Enter 'To' unit: Inches
Enter the conversion ratio: .3937

This entry has been saved.

1 - Convert a length
2 - Add a type of conversion
3 - Delete a type of conversion
4 - Exit

Enter menu number: 1

1 -  Miles to Kilometers: 1.6093
2 -  Kilometers to Miles: 0.6214
3 -  Inches to Centimeters: 2.54
4 -  Centimeters to Inches: 0.3937

Enter conversion number: 4

Enter Centimeters: 2.54
2.54 Centimeters = 1 Inches

1 - Convert a length
2 - Add a type of conversion
3 - Delete a type of conversion
4 - Exit

Enter menu number: 4

Goodbye.
```

Project 17-2: Convert lengths (cont.)

Operation

- This application begins by displaying a main menu with four items.
- If the user chooses the first main menu item, the application displays a menu of possible conversions. After the user selects a conversion, the application prompts the user to enter a unit of measurement, calculates the conversion, displays the result, and displays the main menu again.
- If the user chooses the second main menu item, the application prompts the user to enter the values for a new conversion, saves this new conversion to a file, and displays a message to the user.
- If the user chooses the third main menu item, the application displays a menu of possible conversions. After the user selects the conversion, the application deletes that conversion from the file, displays a message to the user, and displays the main menu again.
- If the user chooses the fourth main menu item, the application displays a goodbye message and exits.

Specifications

- Create a class named Conversion that can store information about a conversion, including the from unit, from value, to unit, to value, and conversion ratio. This class should also contain the methods that perform the conversion calculations and return the results as a formatted string.
- Create a class named ConversionIO that contains two methods: one that reads an array list of Conversion objects from a file and another that writes an array list of Conversion objects to a file. For example:

```
public static ArrayList<Conversion> getConversions()

public static void saveConversions(ArrayList<Conversion> conversions)
```

- Store the list of conversions in a text file named conversion_types.txt in the same directory as the ConversionIO class. If the conversion_types.txt file doesn't exist, the ConversionIO class should create it. This class should use buffered I/O streams, and it should close all I/O streams when they're no longer needed.
- Create a class named ConversionsApp that displays the menus and responds to the user's choices.
- Use the Console class shown in chapter 8 or a variation of it to get the user's entries.

Project 18-1: Tortoise and the hare race

Console

```
Get set...Go!
Tortoise: 10
Tortoise: 20
Tortoise: 30
Tortoise: 40
Tortoise: 50
Tortoise: 60
Tortoise: 70
Tortoise: 80
Tortoise: 90
Tortoise: 100
Tortoise: 110
Tortoise: 120
Tortoise: 130
Tortoise: 140
Tortoise: 150
Tortoise: 160
Hare: 100
Tortoise: 170
Tortoise: 180
Tortoise: 190
Tortoise: 200
Tortoise: 210
Tortoise: 220
Tortoise: 230
Tortoise: 240
Tortoise: 250
Tortoise: 260
Tortoise: 270
Hare: 200
Tortoise: 280
Tortoise: 290
Tortoise: 300
Tortoise: I finished!
Hare: 300
Hare: I finished!
```

Operation

- This application simulates a race between two runners. The runners differ in their speed and how often they need to rest. One of the runners, named "Tortoise," is slow but never rests. The other runner, named "Hare," is ten times as fast but rests 90% of the time.
- There is a random element to the runners' performance, so the outcome of the race is different each time the application is run.
- The race is run over a course of 300 meters. Each time one of the runners moves, the application displays the runner's new position on the course. The first runner to reach 300 meters wins the race.
- When each runner finishes the race, the application displays a message that indicates that the runner has finished.

Project 18-1: Tortoise and the hare race (cont.)

Specifications

- The main method of the application's main class should create two runner threads and start them. One of the threads should be named "Tortoise." It runs only 10 meters each move, but plods along without ever resting. The other thread should be named "Hare." It should run 100 meters each move, but should rest 90% of the time.
- Each runner should be a separate thread created from the class named RunnerThread. This class should include four variables:
    - a string representing the name of the runner
    - an int value from 1 to 100 indicating the likelihood that on any given move the runner will rest instead of run
    - an int value that indicates the runners speed—that is, how many meters the runner travels in each move
    - an int value indicating the runner's progress on the course
- The run method of the RunnerThread class should consist of a loop that repeats until the runner has reached 300 meters. Each time through the loop, the thread should decide whether it should run or rest based on a random number and the percentage passed to the constructor. If this random number indicates that the runner should run, the class should add the speed value for the runner. The run method should sleep for 300 milliseconds on each repetition of the loop.

Hint

- To determine whether a runner should run or rest, calculate a random number between 1 and 100. Then, have the runner rest if the number is less than or equal to the percentage of time that the runner rests. Otherwise, the runner should run.

Project 20-1: Manage a list of countries

Console

```
Welcome to the Country Manager

1 - List countries
2 - Add a country
3 - Exit

Enter menu number: 1

India
Japan
Mexico
Spain
United States

1 - List countries
2 - Add a country
3 - Exit

Enter menu number: 2

Enter country: France

France has been added.

1 - List countries
2 - Add a country
3 - Exit

Enter menu number: 1

France
India
Japan
Mexico
Spain
United States

1 - List countries
2 - Add a country
3 - Exit

Enter menu number: 3

Goodbye!
```

Operation

- The application begins by displaying a menu with three menu items.
- If the user chooses the first item, the application displays a list of countries that are stored in a database.
- If the user chooses the second item, the application prompts the user to enter a country and then it adds that country to the database.
- If the user chooses the third item, the application displays a goodbye message and exits.

Project 20-1: Manage a list of countries (cont.)

Specifications

- Create a table in the mma database described in chapter 19 to store the necessary data. To do that, you can use the SQL script stored in the create_country_table.sql file that's supplied. If this script isn't supplied, you can create your own SQL script.
- Create a class named CountryDB that contains two methods: one that allows you to read a list of countries and another method that allows you to add a country to the list. For example:

```
public ArrayList<String> getCountries()

public boolean addCountry(String country)
```

- Create a class named CountryApp that displays the menu and responds to the user's choices.
- Use the Console class described in chapter 8 or a variation of it to get the user's entries.

Possible enhancement

- Modify the application so it allows the user to delete a country from the database.

Project 20-2: Manage customer data

Console

```
Welcome to the Customer Manager

COMMAND MENU
list    - List all customers
add     - Add a customer
del     - Delete a customer
help    - Show this menu
exit    - Exit this application

Enter a command: list

CUSTOMER LIST
frankjones@yahoo.com                Frank               Jones
johnsmith@hotmail.com               John                Smith
seagreen@levi.com                   Cynthia             Green
wendyk@warners.com                  Wendy               Kowolski

Enter a command: add

Enter customer email address: test@test.com
Enter first name: Mick
Enter last name: Stipe

Mick Stipe was added to the database.

Enter a command: list

CUSTOMER LIST
frankjones@yahoo.com                Frank               Jones
johnsmith@hotmail.com               John                Smith
seagreen@levi.com                   Cynthia             Green
test@test.com                       Mick                Stipe
wendyk@warners.com                  Wendy               Kowolski

Enter a command: del

Enter email for customer to delete: test@test.com

Mick Stipe was deleted from the database.

Enter a command: list

CUSTOMER LIST
frankjones@yahoo.com                Frank               Jones
johnsmith@hotmail.com               John                Smith
seagreen@levi.com                   Cynthia             Green
wendyk@warners.com                  Wendy               Kowolski

Enter a command: exit

Bye!
```

Project 20-2: Manage customer data (cont.)

Operation

- This application begins by displaying a menu with five choices: list, add, del, help, and exit.
- If the user enters "list", the application displays the customer data.
- If the user enters "add", the application prompts the user to enter data for a customer and saves that data.
- If the user enters "del", the application prompts the user for an email address and deletes the corresponding customer.
- If the user enters "help", the application displays the menu again.
- If the user enters "exit", the application displays a goodbye message and exits.

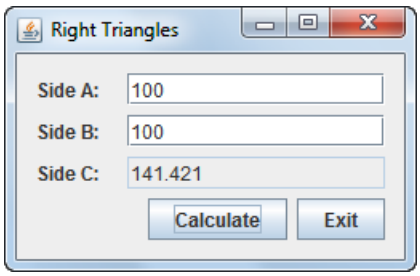
Specifications

- Create a table in the mma database described in chapter 19 to store the necessary data. To do that, you can use the SQL script stored in the create_customer_table.sql file that's supplied. If this script isn't supplied, you can create your own SQL script.
- Create a class named Customer that stores data for the user's email address, first name, and last name.
- Create a class named CustomerDB that contains the methods necessary to get an array list of Customer objects, to get a Customer object for the customer with a specified email address, to add a row to store the data in a Customer object, and to delete a row for the specified email address.
- Create a CustomerApp class that works as shown in the console output. This class should use the Customer and CustomerDB classes to work with the customer data.
- Use the Console class described in chapter 8 or a variation of it to get the user's entries.
- Use the StringUtil class described in chapter 9 to use spaces to align the columns of data.

Possible enhancements

- Add an "update" command that lets the user update an existing customer. This command should prompt the user to enter the email address for a customer. Then, it should let the user update the first name and last name for the customer.
- Add a method to the Console class that uses string parsing techniques to validate the email address. At the least, you can check to make sure that this string contains some text, followed by an @ sign, followed by some more text, followed by a period, followed by some more text. For example, "x@x.x" would be valid while "xxx" or "x@x" would not.

Project 22-1: Calculate the hypotenuse of a right triangle

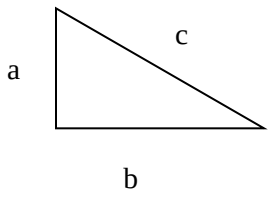


Operation

- This application lets the user enter the lengths of the two shortest sides of a right triangle. When the user clicks the Calculate button, the application calculates and displays the length of the third side.

Specifications

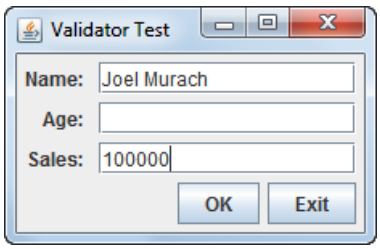
- Use the Pythagorean Theorem to calculate the length of the third side. The Pythagorean Theorem states that the square of the hypotenuse of a right-triangle is equal to the sum of the squares of the opposite sides:



$$c^2 = a^2 + b^2$$

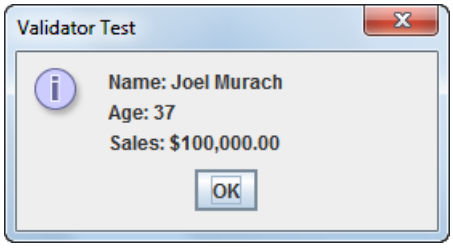
- Validate the user input so that the user must enter a double value for side A and B of the triangle.

Project 22-2: Validate user entries



A validation dialog box

A validation dialog box



Operation

- This application accepts user entries and validates them according to the specifications below.
- If the data is valid, this app displays the data in a dialog box. Then, when the user clicks OK in the dialog box, the application clears the text fields, so the user can make another entry.
- If the data is not valid, this app displays a dialog box with an appropriate error message and attempts to move the focus to the text field with the invalid data.

Specifications

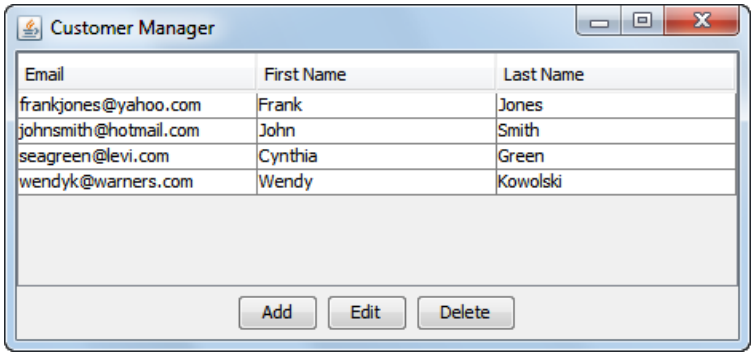
- The Name field is required.
- The Age field is required and must be a valid integer value.
- The Sales field is required and must be a valid double value.
- Create a class named SwingValidator to perform the validation. This class should contain these methods:

```
boolean isNotEmpty(JTextField field, String fieldName)

boolean isInteger(JTextField field, String fieldName)

boolean isDouble(JTextField field, String fieldName)
```

Project 22-3: Manage customer data (GUI)



Operation

- This application begins by displaying a table of customer data.
- If the user clicks the Add button, the application allows the user to add customer data to the table (and the underlying database).
- If the user selects a customer row and clicks the Edit button, the application allows the user to update the data for the selected customer row in the table (and the database).
- If the user selects a customer row and clicks the Delete button, the application deletes the selected customer row from the table (and the database).

Specifications

- Create a table in the mma database described in chapter 19 to store the necessary data. To do that, you can use the SQL script stored in the create_customer_table.sql file that's supplied. If this script isn't supplied, you can create your own SQL script.
- Create a class named Customer that stores data for the user's id, email address, first name, and last name.
- Create a class named CustomerDB that contains the methods necessary to get an array list of Customer objects, to get a Customer object for the customer with the specified id, and to add, update, or delete the specified customer.
- Create a CustomerManagerFrame class like the one shown above. This frame should display a table of customer data as well as the Add, Edit, and Delete buttons. This class should use the Customer and CustomerDB classes to work with the customer data.
- Create a CustomerForm class that allows the user to add or edit customer data.

More ideas for projects

- After chapter 17, you can convert any of the applications that need to store data so that they store their data in a text file.
- After chapter 20, you can convert any of the applications that need to store data so that they store their data in a database.
- After chapter 22, you can have your students convert any of the console applications presented earlier in the book to run as GUI applications.