Student projects for *Murach's Java Programming (5th edition)*

The projects in this document let your students apply the programming skills they'll learn as they progress through *Murach's Java Programming (5th edition)*. 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 project. For example, the student should complete chapter 2 before starting project 2-1 or 2-2, and the student should complete chapter 3 before starting project 3-1, 3-2, or 3-3.

The last page of this document presents some ideas for additional projects that can be assigned for chapters 15-21. So, if you need additional projects for these chapters, this page might be helpful.

General guidelines

Naming

- When creating project names for your applications, please use the convention specified by your instructor. Otherwise, name the project *first_last_app* where *first_last* specifies your first and last name and *app* specifies the name of the application.
- When creating names for variables and functions, please use the guidelines and recommendations specified by your instructor. Otherwise, use the guidelines and recommendations specified in *Murach's Java Programming*.

User interfaces

- You should think of the user interfaces that are shown for the projects as starting points. If you can improve on them, especially to make them more user-friendly, by all means do so.

Specifications

- You should think of the specifications that are given for the projects as starting points. If you have the time to enhance the applications by improving on the starting specifications, by all means do so.

Development tip

- Always start by developing a working version of the application for a project. That way, you'll have something to show for your efforts if you run out of time. Then, you can build out that starting version of the application until it satisfies all of the specifications.

Files supplied by your instructor

- Some of the projects require starting files. These files are identified in the specifications for the projects, and your instructor should make these starting files available to you.

Project 2-1: Student Registration

Create an application that allows a student to enter registration information. The application should display a completion message that includes the user's full name and a temporary password.

Console

```
Student Registration Form

Enter first name: Harold
Enter last name: Moore
Enter year of birth: 1998

Welcome Harold Moore!
Your registration is complete.
Your temporary password is: Harold*1998
```

Specifications

- The user's full name consists of the user's first name, a space, and the user's last name.
- The temporary password consists of the user's first name, an asterisk (*), and the user's birth year.
- Assume that the user will enter a valid 4-digit integer for the year.

Project 2-2: Grade Converter

Create an application that converts 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
```

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 between 1 and 100 for the grades.
- The application should continue only if the user enters “y” or “Y” to continue.

Project 2-3: Rectangle Calculator

Create an application that calculates the area and perimeter of a rectangle.

Console

```
Welcome to the Area and Perimeter Calculator

Enter length: 100
Enter width:  200
Area:         20000.0
Perimeter:    600.0

Continue? (y/n): y

Enter length: 8
Enter width:  4
Area:         32.0
Perimeter:    24.0

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

Specifications

- The formulas for calculating area and perimeter are:

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

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

Project 3-1: Temperature Converter

Create an application that converts temperature values from Fahrenheit to Celsius.

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
```

Specifications

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

  `c = (f-32) * 5/9`
- The application should accept decimal entries like 77.5.
- The application should format the Celsius temperature to allow for up to 2 decimal places.
- Assume that the user will enter valid data.
- The application should continue only if the user enters “y” or “Y” to continue.

Project 3-2: Travel Time Calculator

Create an application that calculates travel time based on distance and speed.

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
```

Specifications

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

Hint

- Use integer arithmetic and the division and modulus operators to get hours and minutes.

Project 3-3: Interest Calculator

Create an application that calculates the amount of interest for a loan amount.

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
```

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 application should format currencies to allow for up to 2 decimal places and percentages to 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 3-4: Change Calculator

Create an application that calculates the minimum number of quarters, dimes, nickels, and pennies needed for the specified number of cents.

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): 68

Quarters: 2
Dimes:    1
Nickels:  1
Pennies:  3

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

Specifications

- Assume that the user will enter a valid integer value between 0 and 99 for the number of cents.
- The application should continue only if the user enters “y” or “Y” to continue.

Project 4-1: Table of Powers

Create an application that displays a table of squares and cubes from 1 to the value entered by the user.

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
```

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 4-2: Factorial Calculator

Create an application that calculates the factorial of the number entered by the user.

Console

```
Welcome to the Factorial Calculator

Enter an integer that's greater than 0 and less than 10: 3
The factorial of 3 is 6.

Continue? (y/n): y

Enter an integer that's greater than 0 and less than 10: 4
The factorial of 4 is 24.

Continue? (y/n): y

Enter an integer that's greater than 0 and less than 10: 9
The factorial of 9 is 362880.

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

Specifications

- The exclamation point is used to identify a factorial. For example, the factorial of the number n is denoted by n!. Here's how you calculate the factorial of the numbers 1 through 5:

```
1! = 1                   which equals 1
2! = 1 * 2               which equals 2
3! = 1 * 2 * 3           which equals 6
4! = 1 * 2 * 3 * 4       which equals 24
5! = 1 * 2 * 3 * 4 * 5   which equals 120
```

- Use a for loop to calculate the factorial.
- Assume that the user will enter valid numeric data for the length and width.
- Use the long type to store the factorial.
- The application should continue only if the user enters "y" or "Y" to continue.

Enhancement

- Test the application and find the integer for the highest factorial that can be accurately calculated by this application. (A factorial that can't be calculated accurately will be displayed as either a negative number or 0.) Then, modify the prompt so it prompts the user for a number from 1 to the highest integer that returns an accurate factorial calculation.

Project 4-3: Tip Calculator

Create an application that calculates three options for an appropriate tip to leave after a meal at a restaurant.

Console

```
Tip Calculator

Cost of meal: 52.31

15%
Tip amount:    $7.85
Total amount:  $60.16

20%
Tip amount:    $10.46
Total amount:  $62.77

25%
Tip amount:    $13.08
Total amount:  $65.39

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

Specifications

- The application should calculate and display the cost of tipping at 15%, 20%, and 25%.
- Use the BigDecimal class to make sure that all calculations are accurate.
- Assume that the user will enter a valid cost for the meal.
- Format the tip percent, the tip amount, and the total.
- The application should continue only if the user enters “y” or “Y” to continue.

Project 4-4: Common Divisor Calculator

Create an application that finds the greatest common divisor of two positive integers entered by the user.

Console

```
Greatest Common Divisor Finder

Enter first number: 12
Enter second number: 8
Greatest common divisor: 4

Continue? (y/n): y

Enter first number: 77
Enter second number: 33
Greatest common divisor: 11

Continue? (y/n): y

Enter first number: 441
Enter second number: 252
Greatest common divisor: 63

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

Specifications

- The formula for finding the greatest common divisor of two positive integers x and y follows the Euclidean algorithm as follows:

    1. Subtract x from y repeatedly until y < x.
    2. Swap the values of x and y.
    3. Repeat steps 1 and 2 until x = 0.
    4. y is the greatest common divisor of the two numbers.

- You can use one loop for step 1 of the algorithm nested within a second loop for step 3.
- Assume that the user will enter valid integers for both numbers.
- The application should continue only if the user enters “y” or “Y” to continue.

Project 5-1: Dice Roller

Create an application that rolls a pair of six-sided dice.

Console

```
Dice Roller

Roll the dice? (y/n): y

Die 1: 3
Die 2: 1
Total: 4

Roll again? (y/n): y

Die 1: 1
Die 2: 1
Total: 2
Snake eyes!

Roll again? (y/n): y

Die 1: 6
Die 2: 6
Total: 12
Boxcars!

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

Specifications

- You can use the random method of the Math class to generate a random number for a die like this:

  ```
  (int) (Math.random() * 6) + 1;
  ```

- The application should display special messages for two ones (snake eyes) and two sixes (box cars).
- The application should use static methods to organize its code.
- The application should continue only if the user enters “y” or “Y” at the “Roll again?” prompt.

Project 5-2: Data Validation

Add data validation to any of the projects from chapters 2 through 4.

Console

```
Welcome to the Area and Perimeter Calculator

Enter length: ten
Error! Invalid decimal value. Try again.
Enter length: -10
Error! Number must be greater than 0.0
Enter length: 100000000000000000000000
Error! Number must be less than 1000000.0
Enter length: 100
Enter width:  ten
Error! Invalid decimal value. Try again.
Enter width:  -10
Error! Number must be greater than 0.0
Enter width:  10000000000000000000000
Error! Number must be less than 1000000.0
Enter width:  100
Area:         10000.0
Perimeter:    400.0

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
```

Specifications

- If the application requires a numeric value, the application should continue prompting the user until the user enters a valid number.
- If the application requires a string value, it should continue prompting the user until the user enters a valid string value.
- The code that's used to validate data should be stored in separate methods. For example:

```
public static double getDoubleWithinRange(Scanner sc, String prompt,
    double min, double max)
```

```
public static int getIntWithinRange(Scanner sc, String prompt,
    int min, int max)
```

Project 5-3: Guessing Game

Create an application that lets a user guess a number between 1 and 100.

Console

```
Welcome to the Guess the Number Game
++++++++++++++++++++++++++++++++++++

I'm thinking of a number from 1 to 100.
Try to guess it.

Enter number: 50
You got it in 1 tries.
Great work! You are a mathematical wizard.

Try again? (y/n): y

I'm thinking of a number from 1 to 100.
Try to guess it.

Enter number: 50
Way too high! Guess again.

Enter number: 30
Too high! Guess again.

Enter number: 15
Too low! Guess again.

Enter number: 23
Too high! Guess again.

Enter number: 19
Too low! Guess again.

Enter number: 21
Too high! Guess again.

Enter number: 20
You got it in 7 tries.
Not too bad! You've got some potential.

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

Bye - Come back soon!
```

Project 5-3: Guessing Game (continued)

Specifications

- If the user's guess is higher than the random number, the application should display, "Too high!"
- If the user's guess is lower than the random number, the application should display, "Too low!"
- If the user's guess is more than 10 higher or 10 lower than the random number, the application should display, "Way too high!" or "Way too low!"
- The message that's displayed when the user gets the number should vary depending on the 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.
```

- When the user guesses a number, the application should only accept numbers from 1 to 100.
- When the user responds to the "Try Again?" prompt, the application should only accept a value of "y" or "n".
- 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.
- The code that's used to validate data should be stored in separate methods. For example:

```
public static double getDoubleWithinRange(Scanner sc, String prompt,
    double min, double max)

public static int getIntWithinRange(Scanner sc, String prompt,
    int min, int max)
```

- The code that's used to run the application should also be stored in separate methods.
- Use the random() method of the java.lang.Math class to generate a random number.

Project 7-1: Contact List

Create an application that uses a class to store and display contact information.

Console

```
Welcome to the Contact List application

Enter first name: Mike
Enter last name:  Murach
Enter email:      mike@murach.com
Enter phone:      800-221-5528

--------------------------------------------
---- Current Contact -----------------------
--------------------------------------------
Name:           Mike Murach
Email Address:  mike@murach.com
Phone Number:   800-221-5528
--------------------------------------------

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

Specifications

- Use a class named Contact to store the data for each contact. This class should include these methods:

```
public void setFirstName(String name)
public String getFirstName()
public void setLastName(String name)
public String getLastName()
public void setEmail(String email)
public String getEmail()
public void setPhone(String phone)
public String getPhone()
public String displayContact()
```

- Use the Console class presented in chapter 7 or an enhanced version of it to get and validate the user’s entries.
- The application should continue only if the user enters “y” or “Y” to continue.

Project 7-2: Grade Converter

Create an application that uses a class to convert number grades to letter grades and another class for data validation.

Console

```
Welcome to the Letter Grade Converter

Enter numerical grade: 90
Letter grade: A

Continue? (y/n): y

Enter numerical grade: A
Error! Invalid integer. Try again.
Enter numerical grade: 87.9
Error! Invalid integer. Try again.
Enter numerical grade: 87
Letter grade: B

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

Specifications

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

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

- The Grade class should have two constructors. The first one should accept no parameters and set the initial value of the number instance variable to zero. The second should accept an integer value and use it to set the initial value of the number instance variable.
- The grading criteria are as follows:

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

- Use the Console class presented in chapter 7 or an enhanced version of it to get and validate the user’s entries.
- Overload the getString() method of the Console class to add the ability to require a string value, and to require one of two specified string values.
- When the user responds to the Continue prompt, the application should only accept a value of “y” or “n”.

Project 7-3: Guessing Game

Convert a previous application so it uses classes to organize its code.

Console

```
Welcome to the Guess the Number Game
++++++++++++++++++++++++++++++++++++

I'm thinking of a number from 1 to 100.
Try to guess it.

Enter number: 50
You got it in 1 tries.
Great work! You are a mathematical wizard.

Try again? (y/n): y

I'm thinking of a number from 1 to 100.
Try to guess it.

Enter number: 50
Way too high! Guess again.

Enter number: 30
Too high! Guess again.

Enter number: 15
Too low! Guess again.

Enter number: 23
Too high! Guess again.

Enter number: 19
Too low! Guess again.

Enter number: 21
Too high! Guess again.

Enter number: 20
You got it in 7 tries.
Not too bad! You've got some potential.

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

Bye - Come back soon!
```

Specifications

- Your instructor should provide you with a starting project.
- Create a class named Console, and move all the methods that retrieve and validate user input to that class. These methods can remain static.
- Create a class named Game, and move all the methods that display messages and handle user guesses to that class. Adjust these methods so they aren't static, and use instance variables of the Game class to keep track of numbers, guesses, and so on.
- Update the application to use these classes and their methods. Make sure the application functions the same as it did before.

Project 7-4: Dice Roller

Create an application that uses classes to roll a pair of six-sided dice.

Console

```
Welcome to the Dice Roller!

Roll the dice? (y/n): y

Die 1: 2
Die 2: 5
Total: 7
Craps!

Roll again? (y/n): y

Die 1: 2
Die 2: 1
Total: 3

Roll again? (y/n): y

Die 1: 4
Die 2: 6
Total: 10

Roll again? (y/n): y

Die 1: 6
Die 2: 6
Total: 12
Box cars!

Roll again? (y/n): y

Die 1: 1
Die 2: 1
Total: 2
Snake eyes!

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

Project 7-4: Dice Roller (continued)

Specifications

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

  ```
  public Die()              // set the initial value of the die to zero
  public void roll()
  public int getValue()
  ```

- Create a class named Dice to store two dice. This class should contain two instance variables of the Die type and these constructors and methods:

  ```
  public Dice()               // instantiate the Die instance variables
  public int getDie1Value ()
  public int getDie2Value ()
  public int getSum()         // get the sum of both dice
  public void roll()          // roll both dice
  public void printRoll()     // display result of roll
  ```

- You can use the random method of the Math class to generate a random number for a die like this:

  ```
  value = (int) (Math.random() * 6) + 1;
  ```

- When printing the results of the roll of the dice, display the value of each die and the total. In addition, display special messages for craps (sum of both dice is 7), snake eyes (double 1's), and box cars (double 6's).

- Continue only if the user enters "y" or "Y" at the "Roll again?" prompt.

Project 8-1: Number Checker

Create an application that checks whether an integer is an odd or even number.

Console

```
Welcome to the Odd/Even Checker!

Enter an integer: ten
Error! Invalid integer. Try again.
Enter an integer: 10.3
Error! Invalid integer. Try again.
Enter an integer: 10
The number 10 is even.

Continue? (y/n):
Error! This entry is required. Try again.
Continue? (y/n): y

Enter an integer: 9
The number 9 is odd.

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

Specifications

- Create a version of the Console class presented in chapter 7 that doesn't use static methods.
- Create a MyConsole class that inherits the Console class. Then, override the getString() method so it doesn't allow the user to enter an empty string.
- Use an instance of the MyConsole class to get and validate the user's entries.

Hint

- You can use the modulus operator to determine whether the integer entered by the user is odd or even.

Project 8-2: Person Manager

Create an application that lets you enter a new customer or a new employee.

Console

```
Welcome to the Person Manager

Create customer or employee? (c/e):
Error! This entry is required. Try again.
Create customer or employee? (c/e): p
Error! Entry must be 'c' or 'e'. Try again.
Create customer or employee? (c/e): c

First name: Steve
Last name: Trevor
Customer number: M10293

You entered a new Customer:
Name: Steve Trevor
Customer Number: M10293

Continue? (y/n): y

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

First name: Diana
Last name: Prince
SSN: 111-22-3333

You entered a new Employee:
Name: Diana Prince
SSN: xxx-xx-3333

Continue? (y/n): OK
Error! Entry must be 'y' or 'n'. Try again.
Continue? (y/n): n
```

Project 8-2: Person Manager (continued)

Specifications

- Create a class named Person with these constructors and methods:

  ```
  public Person(String first, String last)
  public String getFirstName()
  public void setFirstName(String first)
  public String getLastName()
  public void setLastName()
  ```

  The Person class should override the toString() method so it returns the first name and last name in this format:

  ```
  Name: Frank Jones
  ```

- Create a class named Customer that inherits the Person class and contains these constructors and methods:

  ```
  public Customer(String first, String last, String number)
  public String getCustomerNumber()
  public void setCustomerNumber(String number)
  ```

  The Customer class should override the toString() method so it returns the value returned by the toString() method of the Person class appended with the customer number, like this:

  ```
  Name: Frank Jones
  Customer Number: J54128
  ```

- Create a class named Employee that inherits the Person class and contains these constructors and methods:

  ```
  public Employee(String first, String last, String ssn)
  public String getSsn()
  public void setSsn(String ssn)
  ```

  The getSsn() method should return a masked version of the social security number that only reveals the last four numbers.

  The Employee class should override the toString() method so it returns the value returned by the toString() method of the Person class appended with the social security number, like this:

  ```
  Name: Frank Jones
  SSN: xxx-xx-1111
  ```

- Use the Console class presented in chapter 7 or an enhanced version of it to get and validate the user's entries.

Project 8-3: Area Calculator

Create an application that calculates the area of various shapes.

Console

```
Welcome to the Area Calculator

Calculate area of a circle, square, or rectangle? (c/s/r): c

Enter radius: 15

The area of the Circle you entered is 706.8583470577034

Continue? (y/n): y

Calculate area of a circle, square, or rectangle? (c/s/r): r

Enter width: 15
Enter height: 20

The area of the Rectangle you entered is 300.0

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

Specifications

- Create an abstract class named Shape. This class should contain an abstract method named getArea() that returns a double type.
- Create a class named Circle that inherits the Shape class and contains these constructors and methods:

  ```
  public Circle(double radius)
  public double getRadius()
  public void setRadius(double radius)
  public double getArea()
  ```

- Create a class named Square that inherits the Shape class and contains these constructors and methods:

  ```
  public Square(double width)
  public double getWidth()
  public void setWidth(double width)
  public double getArea()
  ```

- Create a class named Rectangle that inherits the Square class and contains these constructors and methods:

  ```
  public Rectangle(double width, double height)
  public double getHeight()
  public void setHeight(double height)
  public double getArea()
  ```

- Use the following formulas to calculate area:

  Circle: Area = radius^2 * pi
  Square: Area = width^2
  Rectangle: Area = width * height

- Use the Console class presented in chapter 7 or an enhanced version of it to get and validate the user's entries, but assume that the user will enter a valid shape entry.

Project 9-1: Animal Counter

Create an application that creates and counts various animals.

Console

```
Counting alligators...

1 alligator
2 alligator
3 alligator

Counting sheep...

1 Blackie
2 Blackie

1 Dolly
2 Dolly
3 Dolly

1 Blackie
```

Specifications

- Create an interface named Countable that can be used to count an object. This interface should include these methods:

  ```
  void incrementCount()
  void resetCount()
  int getCount()
  String getCountString()
  ```

- Create an abstract class named Animal that implements the Countable interface. This class should include an instance variable that stores the count.
- Create a class named Alligator that extends the Animal class. This class should override the getCountString() method to return a string specific to alligators.
- Create a class named Sheep that extends the Animal class and implements the Cloneable interface. This class should include an instance variable that stores a name, it should provide methods that can set and get the name, and it should override the getCountString() method to return a string that includes the name.
- Create a class named AnimalCounterApp that includes a static method that lets you count any Countable object a specified number of times. For example:

  ```
  public static void count(Countable c, int maxCount)
  ```

- As shown above, count an alligator 3 times. Then, (a) count a sheep 2 times, (b) clone the sheep, change the clone's name, and count the clone 3 times, and (c) count the first sheep again 1 time.

Project 9-2: Account Balance Calculator

Create an application that calculates and displays the starting and ending monthly balances for a checking account and a savings account.

Console

```
Welcome to the Account application

Starting Balances
Checking: $1,000.00
Savings:  $1,000.00

Enter the transactions for the month

Withdrawal or deposit? (w/d): w
Checking or savings? (c/s): c
Amount?: 500

Continue? (y/n): y

Withdrawal or deposit? (w/d): d
Checking or savings? (c/s): s
Amount?: 200

Continue? (y/n): n

Monthly Payments and Fees
Checking fee:              $1.00
Savings interest payment:  $12.00

Final Balances
Checking: $499.00
Savings:  $1,212.00
```

Project 9-2: Account Balance Calculator (continued)

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. This class should include an instance variable for the balance.
- Create a class named CheckingAccount that inherits the Account class. This class should include an instance variable for the monthly fee that's initialized to the value that's passed to the constructor. This class should also include methods that subtract the monthly fee from the account balance and return the monthly fee.
- Create a class named SavingsAccount that inherits the Account class. This class should include instance variables for the monthly interest rate and the monthly interest payment. The monthly interest rate should be initialized to the value that's passed to the constructor. The monthly interest payment should be calculated by a method that applies the payment to the account balance. This class should also include a method that returns the monthly interest payment.
- Create a class named AccountBalanceApp that uses these objects to process and display deposits and withdrawals.
- Use the Console class presented in chapter 7 or an enhanced version of it to get and validate the user's entries. Don't allow the user to withdraw more than the current balance of an account.

Project 9-3: Console Tester

Create and test interfaces that reduce the Console class's linkage to the presentation layer.

Console

```
Welcome to the Console Tester application

Int Test
Enter an integer between -100 and 100: -150
Error! Number must be greater than -101.
Enter an integer between -100 and 100: 50

Double Test
Enter any number between -100 and 100: 200.75
Error! Number must be less than 101.0.
Enter any number between -100 and 100: 50.75

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

String Choice Test
Select one (x/y): z
Error! Entry must be 'x' or 'y'. Try again.
Select one (x/y): x
```

Specifications

- Create an interface named UserOutput that specifies the following methods that display messages to the user:

  ```
  void print(String s)
  void println()
  void println(String s)
  ```

- Create an interface named UserInput that specifies the following methods that get input from the user:

  ```
  int getInt(String prompt);
  int getInt (String prompt, int min, int max);

  double getDouble(String prompt);
  double getDouble (String prompt, double min, double max);

  String getString(String prompt);
  String getString(String prompt, String s1, String s2);
  ```

- Create an interface named UserIO that extends the UserInput and UserOutput interfaces.
- Create a class named ConsoleIO that implements the UserIO interface. The constructor for this class should create an instance of the Scanner class that gets input from the standard input stream. The methods for this class should let you get valid input from the user and display output to the user.
- Code an IOFactory class that contains a static method named getUserIO that returns an instance of a class that implements the UserIO interface. For this project, return an instance of the ConsoleIO class.
- Create a UserIOTestApp class that prompts the user as shown above using a ConsoleIO object from the IO factory. The code in this class shouldn't use the Scanner class or the System.out object directly.

Project 9-3: Console Tester (continued)

Enhancement

- Code a class named JOptionPaneIO that implements the UserIO interface. This class should use the showInputDialog() method of the JOptionPane class to get data from the user, and the showMessageDialog method of the JOptionPane class to show data to the user. (You’ll have to research this method on your own.)
- Edit the getUserIO() method of the IOFactory class so that it accepts a string value and returns either an instance of the ConsoleIO class or the JOptionPaneIO class, depending on the value of the string it receives.
- Edit the UserIOTestApp class to get a JOptionPaneIO object from the factory, and then test the application. Then change the code to get a ConsoleIO object and test the application again.

Project 10-1: Account Balance Calculator

Update the Account Balance Calculator so its code is stored in packages. The functionality of the application should stay the same.

Console

```
Welcome to the Account application

Starting Balances
Checking: $1,000.00
Savings:  $1,000.00

Enter the transactions for the month

Withdrawal or deposit? (w/d): w
Checking or savings? (c/s): c
Amount?: 500

Continue? (y/n): y

Withdrawal or deposit? (w/d): d
Checking or savings? (c/s): s
Amount?: 200

Continue? (y/n): n

Monthly Payments and Fees
Checking fee:              $1.00
Savings interest payment:  $12.00

Final Balances
Checking: $499.00
Savings:  $1,212.00
```

Specifications

- Your instructor should supply you with a starting project.
- Add the following packages to the application:

  ```
  yourLastName.app
  yourLastName.interfaces
  yourLastName.account
  yourLastName.presentation
  ```

- Move the Balanceable, Depositable, and Withdrawable interfaces to the interfaces package. If the IDE identifies any errors, fix them. Then run the application to make sure it still works as expected.
- Move the Account, CheckingAccount, and SavingsAccount classes to the account package. Move the Console class to the presentation package. And move the AccountBalanceApp class to the app package. If the IDE identifies any errors, fix them. Then run the application to make sure it still works as expected.

Project 10-2: Console Tester

Create an application that uses packages and includes documentation for one of its classes.

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
```

Project 10-2: Console Tester (continued)

Specifications

- Name the class that contains the main() method ConsoleTestApp and store it in a package named:

  ```
  yourLastName.app
  ```

- 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 getRequiredString(String prompt);
  public String getChoiceString(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 getDoubleWithinRange(String prompt, double min,
          double max);
  ```

- Store the Console class in a package named

  ```
  yourLastName.presentation
  ```

  Then, add an import statement for this package to the ConsoleTestApp class.

- Add code to the ConsoleTestApp class that uses the Console class as shown above. Feel free to reuse your best code from any previous exercises or projects. Then, test the application to make sure it's working correctly.

- Add javadoc comments to the Console class. These comments should document the purpose, author, and version of the class. It should also document the function 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 10-3: Roshambo

Create an application that uses an enumeration and an abstract class.

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
```

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 represents the player who will play Bart or Lisa. This class should inherit the Player class and implement the generateRoshambo() method. This method can return any value you choose. (You must implement this method even though it isn't used for this player).
- Create a class named RoshamboApp that allows player 1 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 an enhanced version of the Console class to get and validate the user's entries.

Project 10-3: Roshambo (continued)

Enhancement

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

Project 11-1: Batting Statistics

Create an application that calculates batting statistics for baseball players.

Console

```
Welcome to the Batting Average Calculator

Enter number of times at bat: 5

0 = out, 1 = single, 2 = double, 3 = triple, 4 = home
run
Result for at-bat 1: 0
Result for at-bat 2: 1
Result for at-bat 3: 0
Result for at-bat 4: 2
Result for at-bat 5: 3

Batting average: 0.600
Slugging percent: 1.200

Another player? (y/n): y

Enter number of times at bat: 3

0 = out, 1 = single, 2 = double, 3 = triple, 4 = home
run
Result for at-bat 1: 0
Result for at-bat 2: 4
Result for at-bat 3: 0

Batting average: 0.333
Slugging percent: 1.333

Another player? (y/n): n

Bye!
```

Specifications

- The batting average is the total number of at bats for which the player earned at least one base divided by the number of at bats.
- The slugging percentage is the total number of bases earned divided by the number of at bats.
- Use an array to store the at-bat results for each player.
- Validate the input like this:
  - For number of at bats, the user must enter an integer from 1 to 30.
  - For each at bat, the user must enter 0, 1, 2, 3, or 4.
- Format the batting average and slugging percent to show three decimal digits.
- Calculate the statistics for another player only if the user enters “y” or “Y” at the “Another player?” prompt. Otherwise, end the application.

Project 11-2: Student Scores

Create an application that stores student scores and sorts students by name.

Console

```
The Student Scores application

Number of students: 3

STUDENT 1
Last name: Murach
First name: Mike
Score: 99

STUDENT 2
Last name: Murach
First name: Joel
Score: 87

STUDENT 3
Last name: Boehm
First name: Anne
Score: 93

SUMMARY
Boehm, Anne: 93
Murach, Joel: 87
Murach, Mike: 99
```

Specifications

- Create a class named Student that stores the last name, first name, and score for each student. This class should implement the Comparable interface so the students can be sorted by name. If two students have the same last name, the first name should be used to determine the final sort order.
- Use an array to store the Student objects, and sort the array prior to printing the student list.
- Validate the input like this:
  - The number of students must be from 1 to 500.
  - The last and first name can't be an empty string.
  - The score for each student must an integer from 0 to 100.

Project 11-3: Sales Report

Create an application that creates a report from quarterly sales.

Console

```
The Sales Report application

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 sales: $30,534.00
```

Specifications

- The quarterly sales for each region should be hard coded into the application in a rectangular array like this:

```
double[][] sales = {
    {1540.0, 2010.0, 2450.0, 1845.0}, // Region 1
    {1130.0, 1168.0, 1847.0, 1491.0}, // Region 2
    {1580.0, 2305.0, 2710.0, 1284.0}, // Region 3
    {1105.0, 4102.0, 2391.0, 1576.0}  // Region 4
};
```

- 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 getting the sum of the quarterly sales for each region.
- The third section of the report should use nested for loops to calculate the sales by quarter by getting the sum of the individual region sales for each quarter.
- The fourth section of the report should use nested for loops to calculate the total annual sales for the entire company.
- All sections should use the NumberFormat class to format the sales numbers using the currency format.

Project 11-4: Tic Tac Toe

Create a two-player game of Tic Tac Toe.

Console

```
Welcome to Tic Tac Toe

+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+

X's turn
Pick a row (1, 2, 3): 1
Pick a column (1, 2, 3): 1

+---+---+---+
| X |   |   |
+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+

O's turn
Pick a row (1, 2, 3): 1
Pick a column (1, 2, 3): 2

...
...

X's turn
Pick a row (1, 2, 3): 3
Pick a column (1, 2, 3): 3

+---+---+---+
| X | O | O |
+---+---+---+
|   | X |   |
+---+---+---+
|   |   | X |
+---+---+---+

X wins!

Game over!
```

Specifications

- Use an array of arrays to store the values in the Tic Tac Toe grid.
- If the user picks an invalid row or column or a cell that's already taken, display an error message.
- If there is a winner, the game should display an appropriate message and end. Otherwise, it should continue until the grid is full and ends in a tie.

Project 12-1: Prime Number Checker

Create an application that checks whether a number is a prime number and displays its factors if it is not a prime number.

Console

```
Prime Number Checker

Please enter an integer between 1 and 5000: 5
5 is a prime number.

Try again? (y/n): y

Please enter an integer between 1 and 5000: 6
6 is NOT a prime number.
It has 4 factors: 1 2 3 6

Try again? (y/n): y

Please enter an integer between 1 and 5000: 200
200 is NOT a prime number.
It has 12 factors: 1 2 4 5 8 10 20 25 40 50 100 200

Try again? (y/n): n

Bye!
```

Specifications

- A prime number is divisible by two factors (1 and itself). For example, 7 is a prime number because it is only divisible by 1 and 7.
- If the user enters an integer that's not between 1 and 5000, the application should display an error message.
- If the number is a prime number, the application should display an appropriate message.
- If the number is not a prime number, the application should display an appropriate message. Then, it should display the number of factors for the number and a list of those factors.
- Store the factors for each number in an array list.

Project 12-2: Wizard Inventory

Create an application that keeps track of the items that a wizard can carry.

Console

```
The Wizard Inventory game

COMMAND MENU
show - Show all items
grab - Grab an item
edit - Edit an item
drop - Drop an item
exit - Exit program

Command: show
1. wooden staff
2. wizard hat
3. cloth shoes

Command: grab
Name: potion of invisibility
potion of invisibility was added.

Command: grab
You can't carry any more items. Drop something first.

Command: show
1. wooden staff
2. wizard hat
3. cloth shoes
4. potion of invisibility

Command: edit
Number: 1
Updated name: magic wooden staff
Item number 1 was updated.

Command: drop
Number: 3
cloth shoes was dropped.

Command: exit
Bye!
```

Specifications

- Use an array list to store the items. Provide three starting items.
- The wizard can only carry four items at a time.
- For the edit and drop commands, display an error message if the user enters an invalid number for the item.
- When you exit the program, all changes that you made to the inventory are lost.

Project 12-3: Movie List

Create an application that displays all movies for the specified category.

Console

```
The Movie List application

Choose from 100 movies
Categories: drama | musical | scifi | horror | comedy | animated

Enter a category: 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

Enter a category: comedy
Annie Hall
M*A*S*H
Tootsie
Duck Soup

Continue? (y/n): n

Bye!
```

Specifications

- Your instructor should provide a Movie class that stores the title and category for each movie and a MovieIO class that you can use to get an ArrayList of Movie objects.
- When the application starts, it should display a message that indicates the total number of movies available followed by a list of available categories. The category names should be stored in an ArrayList of String objects.
- When the user enters a category, the application should display all movies that match the category.

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 12-4: Stack Calculator

Create an application that simulates an old-school stack calculator that uses Reverse Polish Notation (RPN).

Console

```
Welcome to the Stack Calculator.

Commands: push n, add, sub, mult, div, clear, or quit.

stack> push 4
4.0

stack> push 3
3.0
4.0

stack> push 2
2.0
3.0
4.0

stack> mult
6.0
4.0

stack> add
10.0

stack> clear
empty

stack> quit

Thanks for using the Stack Calculator.
```

Specifications

- The calculator should be implemented as a separate class named StackCalculator. This class should have the following methods:

Method	Explanation
`public void push(double x)`	Pushes x onto the top of the stack.
`public double pop()`	Removes the value from the top of the stack.
`public double add()`	Removes two values from the stack, adds them, and pushes the result back onto the stack.
`public double subtract()`	Same as add() but subtracts the values.
`public double multiply()`	Same as add() but multiplies the values.
`public double divide()`	Same as add() but divides the values.
`public void clear()`	Removes all entries from the stack.
`public double[] getValues()`	Returns all of the values from the stack in an array without removing them from the stack.
`public int size()`	Gets the number of values in the stack.

- The StackCalculator class should use a linked list to maintain the stack data.

Project 13-1: HTML Converter

Create an application that reads an HTML file and converts it to plain text.

Console

```
HTML Converter

INPUT
<h1>Grocery List</h1>
<ul>
    <li>Eggs</li>
    <li>Milk</li>
    <li>Butter</li>
</ul>

OUTPUT
Grocery List
* Eggs
* Milk
* Butter
```

Specifications

- Store the following data in a String variable named html:

```
String html = "<h1>Grocery List</h1>\n" +
              "<ul>\n" +
              "    <li>Eggs</li>\n" +
              "    <li>Milk</li>\n" +
              "    <li>Butter</li>\n" +
              "</ul>";
```

- When the application starts, it should print the HTML input. Then, it should remove the HTML tags, remove any spaces to the left of the tags, add asterisks (*) before the list items, and print the plain text output to the console.

Project 13-2: Email Creator

Create an application that reads a file and creates a series of emails.

Console

```
Email Creator

================================================================
To:      jbutler@gmail.com
From:    noreply@deals.com
Subject: Deals!

Hi James,

We've got some great deals for you. Check our website!
================================================================
To:      josephine_darakjy@darakjy.org
From:    noreply@deals.com
Subject: Deals!

Hi Josephine,

We've got some great deals for you. Check our website!
================================================================
To:      art@venere.org
From:    noreply@deals.com
Subject: Deals!

Hi Art,

We've got some great deals for you. Check our website!
```

Specifications

- Store a list of email addresses in an array like this:

```
String[] csv = {"   james   ,butler,jbutler@gmail.com",
                "Josephine,Darakjy,Josephine_Darakjy@darakjy.org",
                "ART,VENERE,ART@VENERE.ORG"};
```

- Store a template for a mass email in a file like this:

```
String template =
    "To:      {email}\n" +
    "From:    noreply@deals.com\n" +
    "Subject: Deals!\n\n" +
    "Hi {first_name},\n\n" +
    "We've got some great deals for you. Check our website!";
```

- When the application starts, it should read the email addresses and first names from the file, merge them into the mass email template, and display the results on the console.
- All email addresses should be converted to lowercase.
- All first names should be converted to title case.
- If you add names to the list of email addresses, the application should create more emails.
- If you modify the template, the application should change the content of the email that's created.

Project 13-3: Pig Latin Translator

Create an application that translates English to Pig Latin.

Console

```
Pig Latin Translator

Enter a line: This program translates from English to Pig Latin.
isthay ogrampray anslatestray omfray englishway otay igpay atinlay

Another line? (y/n): y

Enter a line: Writing code is hard!
itingwray odecay isway ardhay

Another line? (y/n): y

Enter a line: Your email can't be delivered to joel@murach.com
ouryay emailway an'tcay ebay eliveredday otay joel@murach.com

Another line? (y/n): n

Bye!
```

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.
- 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.
- Make sure the user has entered text before performing the translation.
- Remove punctuation such as periods, commas, and exclamation points if they occur at the end of a word.
- Don't translate words that contain numbers or common symbols such as @, #, and $. For example, 123 should be left as 123, and sergey@gmail.com should be left as sergey@gmail.com.
- Translate words with contractions. For example, *can't* should be *an'tcay*.

Note

- 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 14-1: Reservation Calculator

Create an application that gets arrival and departure dates for a reservation and calculates the total amount for the stay.

Console

```
Reservation Calculator

Enter the arrival month (1-12): 5
Enter the arrival day (1-31): 16
Enter the arrival year: 2018

Enter the departure month (1-12): 5
Enter the departure day (1-31): 18
Enter the departure year: 2018

Arrival Date: May 16, 2018
Departure Date: May 18, 2018
Price: $145.00 per night
Total price: $290.00 for 2 nights

Continue? (y/n): n

Bye!
```

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 $145.00.
- The Reservation class should include the following methods:

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

- Assume that the dates are valid and that the departure date is after the arrival date.

Possible enhancement

- Allow the user to enter the date in the MM/DD/YYYY format.

Project 14-2: Arrival Time Estimator

Create an application that calculates the estimated duration of a trip in hours and minutes. This should include a date/time of departure and an estimated date/time of arrival.

Console

```
Arrival Time Estimator

Departure date (YYYY-MM-DD): 2018-11-09
Departure time (HH:MM): 10:30
Number of miles: 200
Miles per hour: 65

Estimated travel time
Hours: 3
Minutes: 5
Estimated date of arrival: Nov 9, 2018
Estimated time of arrival: 1:35 PM

Continue? (y/n): y

Departure date (YYYY-MM-DD): 2018-12-02
Departure time (HH:MM): 14:00
Number of miles: 180
Miles per hour: 70

Estimated travel time
Hours: 2
Minutes: 40
Estimated date of arrival: Dec 2, 2018
Estimated time of arrival: 4:40 PM

Continue? (y/n): n

Bye!
```

Specifications

- For the date/time of departure and arrival, the application should use the YYYY-MM-DD format for dates and the HH:MM format for times (24-hour).
- For the miles and miles per hour, the application should only accept integer entries like 200 and 65.
- Assume that the dates and times are valid.

Project 15-1: Country List Manager

Console

```
Country List Manager

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

Enter menu number: 1

Canada
United States
Mexico

Enter menu number: 2

Enter country: Thailand
This country has been saved.

Enter menu number: 1

Canada
United States
Mexico
Thailand

Enter menu number: 3

Goodbye.
```

Specifications

- Create a class named CountryIO that contains these methods:

  ```
  public ArrayList<String> getCountries()
  public boolean saveCountries(ArrayList<String> countries)
  ```

- Store the list of countries in a text file named countries.txt in the same directory as the CountriesTextFile class. If the countries.txt file doesn't exist, the CountriesIO 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.
- Use the Console class presented in chapter 7 or an enhanced version of it to get and validate the user's entries.

Possible enhancements

- Modify the CountriesApp class so it includes a menu choice that allows the user to delete a country from the file.

Project 15-2: Length Converter

Create an application that allows the user to convert a length from a list of conversions that are stored in a file. This application should also allow the user to add or delete conversions from the list of conversions.

Console

```
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 15-2: Length Converter (continued)

Specifications

- Create a class named Conversion that can store information about a conversion, including fromUnit, fromValue, toUnit, toValue, and conversionRatio. This class should also contain the methods that perform the conversion calculations and return the results as a formatted string.
- Create a class that contains the methods that store an array list of Conversion objects in a file. For example:

  ```
  public ArrayList<Conversion> getConversions()
  public boolean saveConversions(ArrayList<Conversion> typesList)
  ```

- Store the list of conversions in a text file named conversion_types.txt that's in the same directory as the class that reads and writes the file. If the conversion_types.txt file doesn't exist, the 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.
- Use the Console class or a variation of it to validate the user's entries. A valid integer is required for a menu choice, non-empty strings are required for the "From" and "To" fields, and a valid double is required for the conversion ratio.

Project 15-3: File Cleaner

Create an application that reads a file that contains an email list, reformats the data, and writes the cleaned list to another file.

Console

```
File Cleaner

Source file:  prospects.csv
Cleaned file: prospects_clean.csv

Congratulations! Your file has been cleaned!
```

The prospect.csv file

```
FIRST,LAST,EMAIL
james,butler,jbutler@gmail.com
Josephine   ,Darakjy,josephine_darakjy@darakjy.org
ART,VENERE,ART@VENERE.ORG
...
```

The prospect_clean.csv file

```
First,Last,email
James,Butler,jbutler@gmail.com
Josephine,Darakjy,josephine_darakjy@darakjy.org
Art,Venere,art@venere.org
...
```

Specifications

- Your instructor should provide a CSV file named prospects.csv that contains a list of prospects.
- Your application should fix the formatting problems and write a file named prospects_clean.csv.
- All names should use title case (an initial capital letter with the rest lowercase).
- All email addresses should be lowercase.
- All extra spaces at the start or end of a string should be removed.

Project 15-4: Path Checker

Create an application that checks whether a path exists on the current computer and whether that path points to a directory or a file.

Console

```
Path Checker

Enter a path: /murach

That path points to a directory.

Continue? (y/n): y

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

That path points to a file.

Continue? (y/n): y

Enter a path: /bad

That path does not exist.

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

Specifications

- Use the console to get a path from the user.
- If the path exists, display a message that indicates whether the path points to a directory or a file.
- If the path doesn't exist, displays an appropriate message.

Project 16-1: Customer Viewer

Use a custom exception in an application that displays customer information.

Console

```
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

No customer found for number 2439.

Display another customer? (y/n): n

Bye!
```

Specifications

- Your instructor should provide you with the Customer and CustomerDB classes that are used to get and display the information for a customer.
- Create a NoSuchCustomerException class that can store a message.
- If the getCustomer() method of the CustomerDB class can't find a customer with the specified number, it should throw a CustomerNotFoundException with a message that says "No customer found for number *XXXX*."
- Modify the code for the application so it catches the CustomerNotFoundException.

Project 16-2: Customer Manager

Improve the exception handling for an application that manages customer data.

Console for a FileNotFoundException

```
Welcome to the Customer Manager

Error reading data file! Exiting application.
java.io.FileNotFoundException: customers.txt (The system
cannot find the file specified)
```

Console for an IOException

```
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: add

Enter first name: John
Enter last name: Doe
Enter customer email: johndoe@x.com
Error adding customer. Try again.

Enter a command:
```

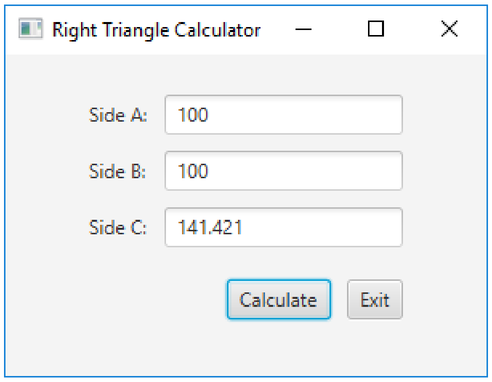
Specifications

- Your instructor should provide you with a starting project for a Customer Manager application that allows you to add and delete customer records. This source code should be the same as the solution for exercise 15-1 in the book.
- Modify the CustomerTextFile class so it throws all exceptions to the calling class. To get this to work, you'll need to modify the DAO interface so each of its methods throws the IOException.
- Modify the CustomerManagerApp class so it handles all exceptions appropriately.
  - If the application can't find the file that stores the data, display an error message and exit the application. To cause a FileNotFoundException, you can move or rename the data file.
  - If the application can't add or delete a record, display an error message and allow the user to try again. To simulate an IOException, you can add a statement to the CustomerTextFile class that throws an IOException.

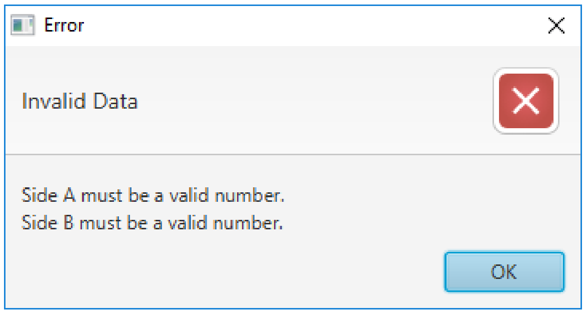
Project 17-1: Hypotenuse Calculator

Create a GUI for an application that lets the user calculate the hypotenuse of a right triangle.

The GUI with valid data



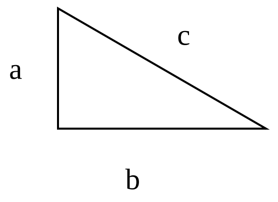
The dialog that's displayed if the user enters invalid data



Side A must be a valid number.
Side B must be a valid number.

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:



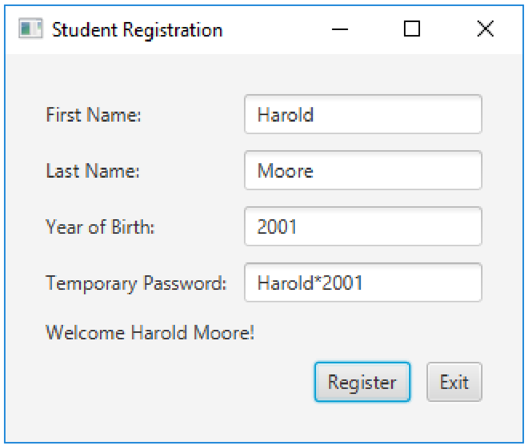
$$c^2 = a^2 + b^2$$

- Validate the user input so that the user must enter a double value for side A and B of the triangle. To do that, you can use the Validation class presented in chapter 18 to validate the user's input.

Project 17-2: Student Registration

Create a GUI application that accepts student registration data.

The GUI with valid data


Student Registration

First Name: Harold

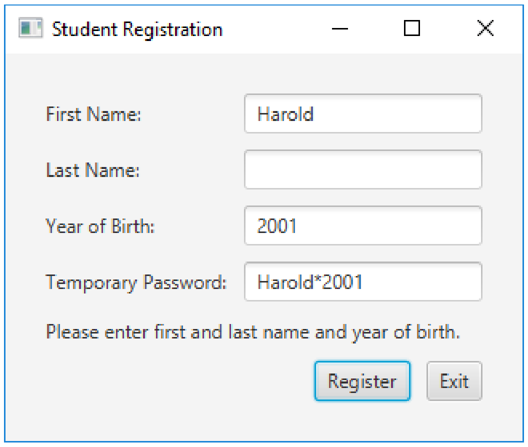
Last Name: Moore

Year of Birth: 2001

Temporary Password: Harold*2001

Welcome Harold Moore!

The GUI with invalid data


Student Registration

First Name: Harold

Last Name:

Year of Birth: 2001

Temporary Password: Harold*2001

Please enter first and last name and year of birth.

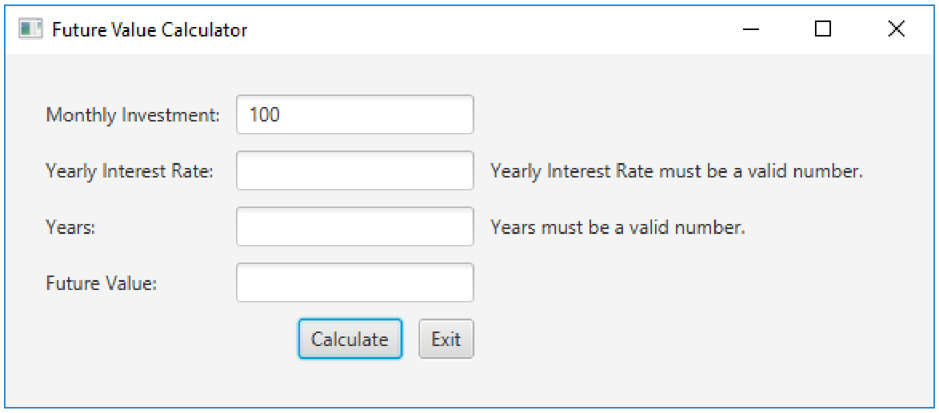
Specifications

- Use FXML to create the GUI.
- The text box that displays the temporary password should be read-only.
- The temporary password consists of the user's first name, an asterisk (*), and the user's birth year.
- If the user enters data in the first three fields, display a temporary password in the appropriate text field and a welcome message in the label below the text fields.
- If the user does not enter data, clear the temporary password from the text field, and display an error message in the label below the text fields.

Project 17-3: Future Value Calculator

Create a Future Value Calculator that displays error messages in labels.

GUI



Specifications

- Start with the JavaFX version of the Future Value application presented in chapter 17.
- Create error message labels for each text field that accepts user input.
- Use the Validation class from chapter 17 to validate user input.
- Format the application so that the controls don't change position when error messages are displayed.

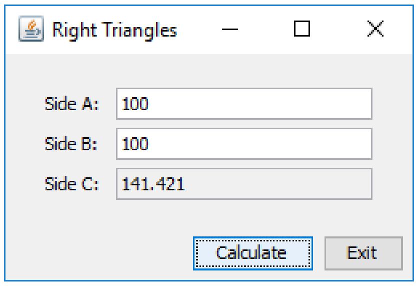
Enhancement

- Do the same thing with the FXML version of the Future Value application presented in chapter 17. Note: To get this to work correctly, you'll need to do some research on how to use the ColumnConstraints class in FXML.

Project 18-1: Hypotenuse Calculator

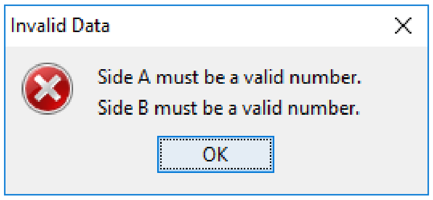
Create a GUI for an application that lets the user calculate the hypotenuse of a right triangle.

The GUI with valid data



Side A: 100
Side B: 100
Side C: 141.421

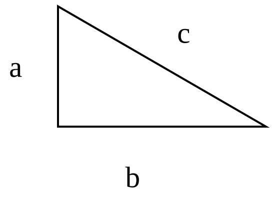
The dialog that's displayed if the user enters invalid data



Side A must be a valid number.
Side B must be a valid number.

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:



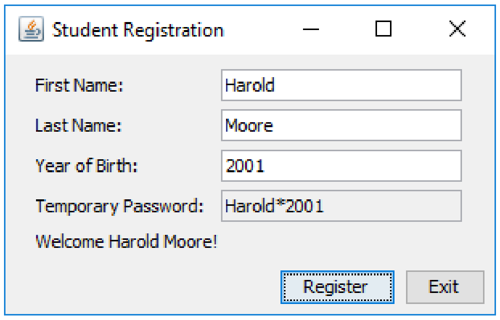
$$c^2 = a^2 + b^2$$

- Validate the user input so that the user must enter a double value for side A and B of the triangle. To do that, you can use the Validation class presented in chapter 18 to validate the user's input.

Project 18-2: Student Registration

Create a GUI application that accepts student registration data.

The GUI with valid data



First Name: Harold

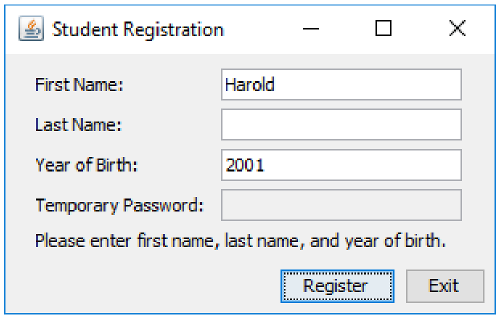
Last Name: Moore

Year of Birth: 2001

Temporary Password: Harold*2001

Welcome Harold Moore!

The GUI with invalid data



First Name: Harold

Last Name:

Year of Birth: 2001

Temporary Password:

Please enter first name, last name, and year of birth.

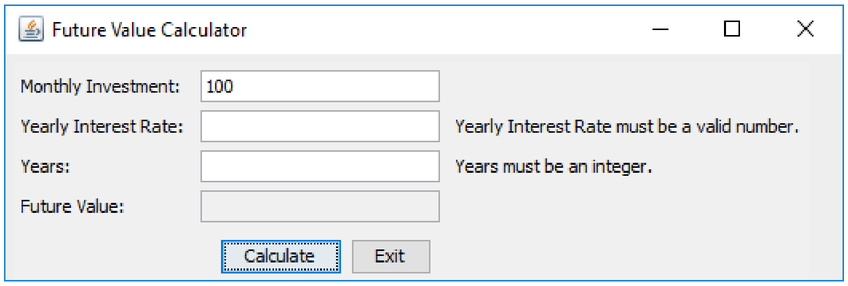
Specifications

- The text box that displays the temporary password should be read-only.
- The temporary password consists of the user's first name, an asterisk (*), and the user's birth year.
- If the user enters data in the first three fields, display a temporary password in the appropriate text field and a welcome message in the label below the text fields.
- If the user does not enter data, clear the temporary password from the text field, and display an error message in the label below the text fields.

Project 18-3: Future Value Calculator

Create a Future Value Calculator that displays error messages in labels.

The GUI



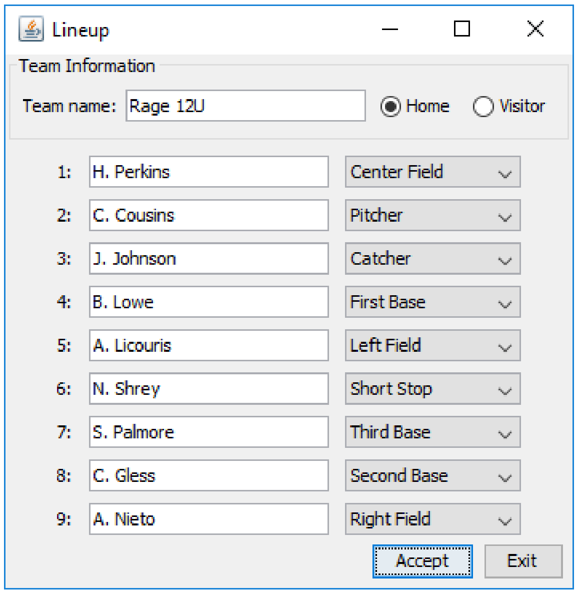
Specifications

- Start with the Swing version of the Future Value application presented in chapter 18.
- Create error message labels for each text field that accepts user input.
- Use the Validation class from chapter 18 to validate user input.
- Format the application so that the controls don't change position when error messages are displayed.

Project 19-1: Team Lineup

Create an application that lets the user enter the lineup for a baseball team.

The GUI

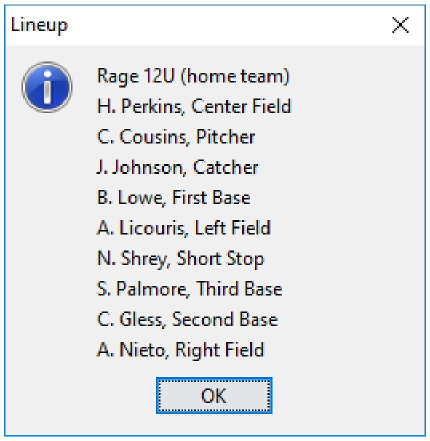


Team Information

Team name: Rage 12U [✓ Home] Visitor

1:	H. Perkins	Center Field
2:	C. Cousins	Pitcher
3:	J. Johnson	Catcher
4:	B. Lowe	First Base
5:	A. Licouris	Left Field
6:	N. Shrey	Short Stop
7:	S. Palmore	Third Base
8:	C. Gless	Second Base
9:	A. Nieto	Right Field

The dialog box that's displayed for the Accept button



```
Rage 12U (home team)
H. Perkins, Center Field
C. Cousins, Pitcher
J. Johnson, Catcher
B. Lowe, First Base
A. Licouris, Left Field
N. Shrey, Short Stop
S. Palmore, Third Base
C. Gless, Second Base
A. Nieto, Right Field
```

Specifications

- The team positions in the combo boxes should offer the following choices:

  Choose a selection | Pitcher | Catcher |
  First base | Second base | Third base | Short stop
  Left field | Center field | Right field

- Add data validation that requires the user to enter all text fields.
- Add data validation that prevents the user from assigning the same position to multiple players.

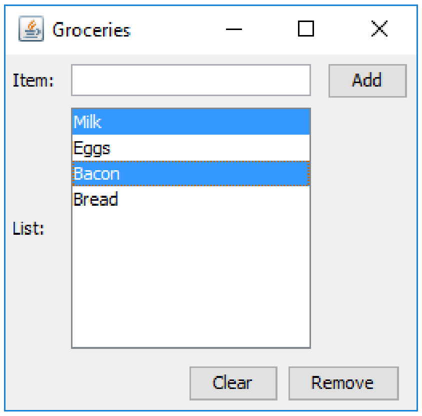
Enhancements

- Store the data for the lineup in a file. Then, modify the application so it reads the lineup from the file when it starts and writes the lineup to the file when the user clicks OK.

Project 19-2: Grocery List

Create an application that lets the user enter the lineup for a baseball team.

The GUI



Specifications

- Allow the user to add an item by entering an item in the Item text field and clicking the Add button.
- If the user clicks the Add button without entering an item in the Item text field, display a dialog box that describes the problem and don't add an item to the list.
- Allow the user to remove one or more items by selecting one or more items and clicking the Remove button.
- If the user clicks the Remove button without selecting one or more items, display a dialog box that describes the problem.
- Allow the user to remove all items by clicking the Clear button.

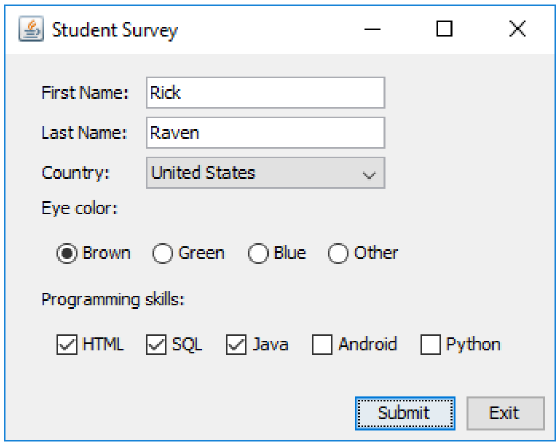
Enhancements

- Store the data for the grocery list in a file. Then, modify the application so it reads the grocery list from the file when it starts and writes the list to the file whenever the user modifies the list by adding or removing an item.
- Add buttons that allow the user to move a selected item up or down in the list.

Project 19-3: Survey

Create an application that lets the user enter information for a survey.

The GUI



First Name: Rick

Last Name: Raven

Country: United States

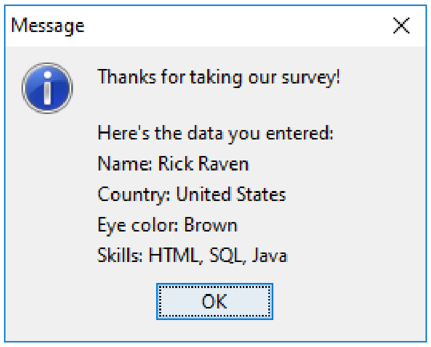
Eye color:

(● Brown) ○ Green ○ Blue ○ Other

Programming skills:

☑ HTML ☑ SQL ☑ Java ☐ Android ☐ Python

The dialog box that's displayed when the user submits the survey



Thanks for taking our survey!

Here's the data you entered:
Name: Rick Raven
Country: United States
Eye color: Brown
Skills: HTML, SQL, Java

Specifications

- The Country combo box should let the user select from at least 10 countries, the first item should be "Select a country…", and the last item should be "Other". So, the items for the combo box should be something like this:

  Select a country…
  At least 10 countries
  Other

- Add data validation that requires the user to enter a first and last name, select a country, and select an eye color. If the validation fails, display a dialog box that asks the user to finish entering data before submitting the survey.

Enhancements

- Store the list of countries in a file. Then, modify the application so it reads the list of countries from the file when it starts and uses them to populate the Country combo box.
- Store the survey data in a file. Then, each time the user clicks the Submit button, the application should append the data in the form to a tab-delimited text file.

Project 21-1: Artists and Albums

Create an application that lists the artists, albums, and albums by artist from the tables in a database.

Console

```
Artist and Album Listing

Artists
-------
Elvis Presley
John Prine
The Beatles

Albums
------
Elvis 30 #1 Hits
Elvis at Sun
John Prine
Revolver
Rubber Soul
Sgt. Pepper's Lonely Hearts Club Band
Sweet Revenge
The White Album

Albums by Artist
----------------
Elvis Presley
        Elvis 30 #1 Hits
        Elvis at Sun
John Prine
        John Prine
        Sweet Revenge
The Beatles
        Revolver
        Rubber Soul
        Sgt. Pepper's Lonely Hearts Club Band
        The White Album
```

Specifications

- Your instructor should provide you with a database file (music_artists.sqlite) that contains tables named Artists and Albums. These tables store the data for the artists and the albums for each artist.
- Create classes named Artist and Album that you can use to create objects that store the data for artists and albums. The Artist object should store a list of all Album objects that are related to the artist.
- Create a class named MusicDB with methods that get a connection to the database and return a list of Artist objects that contain the data for all artists and their albums.
- Sort the artists and album names in ascending sequence (A to Z).
- Create a class with a main() method that uses the Artist, Album, and MusicDB classes to display a list of the artists, a list of the albums, and a list of the albums by artist.
- If the application encounters any exceptions, it should display them on the console.

Project 21-2: Products by Category

Create an application that accepts a category id and displays the products in that category from the tables in a database.

Console

```
Products by Category

CATEGORIES
1 - Guitars
2 - Basses
3 - Drums

Enter a category id (999 to exit): 2

BASSES
Code        Name                                    Price
--------------------------------------------------------------
precision   Fender Precision                        $799.99
hofner      Hofner Icon                             $399.50

CATEGORIES
1 - Guitars
2 - Basses
3 - Drums

Enter a category id (999 to exit): 3

DRUMS
Code        Name                                    Price
--------------------------------------------------------------
ludwig      Ludwig 5-piece Drum Set with Cymbals    $699.99
tama        Tama 5-Piece Drum Set with Cymbals      $799.99

CATEGORIES
1 - Guitars
2 - Basses
3 - Drums

Enter a category id (999 to exit): 999

Bye!
```

Specifications

- Your instructor should provide you with a database file (guitar_shop.sqlite) that contains tables that store the data for the categories and the products.
- Create Category and Product classes that you can use to store with the data from the Category and Product tables of the database.
- Create a class named ProductDB with methods that provides a method for getting all categories and another method for getting all products for the specified category.
- Use prepared statements to retrieve the data.
- Make sure to close database connections, prepared statements, and result sets when you’re done with them.
- Use spaces to align the data in columns on the console. To do that, you can use a class like the StringUtil class presented in chapter 13.
- Use packages to organize the classes for this application.

Project 21-3: Task List

Create an application that allows you to manage a task list that's stored in a database.

Console

```
Task List

COMMAND MENU
view     - View pending tasks
history  - View completed tasks
add      - Add a task
complete - Complete a task
delete   - Delete a task
exit     - Exit program

Command: view
1. Buy toothbrush
2. Do homework

Command: complete
Number: 2

Command: add
Description: Pay bills

Command: view
1. Buy toothbrush
2. Pay bills

Command: history
1. Get bike fixed (DONE!)
2. Call your mom (DONE!)
3. Do homework (DONE!)

Command: exit
Bye!
```

Specifications

- Your instructor should provide you with a database file (task_list.sqlite) that contains a table that stores the tasks.
- Create a class named Task that you can use to store the data for a task.
- Create a class named TaskDB that contains the data access methods necessary for this application.
  - The method for viewing tasks should only display pending tasks.
  - The method for completing a task should only mark a task as completed, not delete it from the database.
  - The method for viewing the history should only display completed tasks.
  - The method for deleting a task should only allow a completed task to be deleted.
- Use prepared statements. Make sure to close the database connection, prepared statement, and result set after each database operation.
- For a new task, make sure the user enters a non-empty string.
- If the application encounters any exceptions, it should display them on the console.
- Use packages to organize the classes for this application.

Project 21-3: Task List (continued)

Hints

- Before you can mark a task as completed, you need to retrieve all pending tasks so you can get the task id for the selected task. Similarly, before you can delete a task, you need to retrieve all completed tasks so you can get the task id for the selected task.
- Because the TaskID column in the Task table is generated automatically, you don't include it in the column list when you add a row to the table. Also, the value of the Completed column for a new row should be 0.

Enhancement

- Add an "update" command that lets the user update a pending task. This command should prompt the user to enter a task number. Then, it should let the user update the task description.

Project 22-1: Movie Ratings

Create an application that collects and displays information about movies.

Console

```
Welcome to the Movie Ratings application!

-----------------------
What do you want to do?
-----------------------
1 - Enter a movie
2 - View top rated movies
3 - View most recent movies
4 - View all movies
5 - View ratings
6 - Quit application

Choose a menu option: 1

How many movies do you want to enter? 3

Movie #1
----------
Enter title: Wonder Woman
Enter year: 2017
Enter rating between 1 and 5 (decimals OK): 4.1

Movie #2
----------
Enter title: Clash of the Titans
Enter year: 2010
Enter rating between 1 and 5 (decimals OK): 2.6

Movie #3
----------
Enter title: Citizen Kane
Enter year: 1941
Enter rating between 1 and 5 (decimals OK): 4.99

~~~~ menu repeats here ~~~~

Choose a menu option: 2

Movies rated 4.0 or higher
--------------------------
Wonder Woman (2017) Rating: 4.1
Citizen Kane (1941) Rating: 4.99

~~~~ menu repeats here ~~~~

Choose a menu option: 5

Movie rating data
-----------------
Number of movies: 3
Lowest rating:    2.6
Highest rating:   5
Average rating:   3.9

~~~~ menu repeats here ~~~~

Choose a menu option: 6

Goodbye!
```

Specifications

- Create a class named Movie to store the data for a movie.
- Create a class named MovieCollection. This class should contain an instance variable that’s a list of Movie objects. In addition, it should contain these methods:

```
public void add()           // add Movie object to internal list
public List<Movie> filterMovies(Predicate<Movie> condition)()
public double getLowestRating()
public double getHighestRating()
public double getAverageRating()
public void getSize()       //return number of items in internal list
```

- The filterMovies() method should use the Predicate function parameter to filter the movies list of movies, and it should return a list of Movie objects that meet the condition in the filter.
- The getLowestRating(), getHighestRating(), and getAverageRating() methods should use a map and reduce operation to get the movie ratings.
- Create a class named MovieRatingsApp. In this class, pass different Predicate functions to the filterMovies() method of the MovieCollection object to display all movies, top rated movies, and most recent movies.

Project 22-2: Animal List

Create an application that collects and displays various animals.

Console

```
Welcome to the Animal List

Type of animal:
1 - Dog
2 - Cat
3 - Turtle

Choose type: 2
Enter animal's name: Percy

Continue? (y/n): y

Type of animal:
1 - Dog
2 - Cat
3 - Turtle

Choose type: 3
Enter animal's name: Yertle

Continue? (y/n): n

And now let's hear the animals speak
------------------------------------
Percy the Cat says 'Meow'
Yertle waves! (turtles don't have vocal cords)
```

Specifications

- Create an abstract class named Animal. This class should have a private name variable of type String, and contain these constructors and methods:

```
public Animal(String name)
public String getName()
public void setName(String name)
public String getNameAndType()
public abstract void speak()
protected void speak(Consumer<Animal> consumer)
```

- The getNameAndType() method should return the animal's name concatenated with "the" and the type of the class (example: "Percy the Cat").
- Create subclasses named Dog, Cat, and Turtle that extend the Animal class. These classes should have constructors that call the constructor of the parent class.
- When the subclasses override the abstract speak() method, they should call the protected speak() method of the parent class and pass it a function. This function contains the functionality for the speak() method for that subclass.
- You should be able to easily modify this class so an animal "speaks" by printing data to the console or by displaying a GUI dialog box.
- After the user has entered all the animals they want, each animal that they've entered should "speak".
- Use the Console class from chapter 7 or a variation of it to validate the user's input.

Project 23-1: Timer

Create a number guessing game that has the user race against a thread that sleeps for a specified number of seconds.

Console when the user does not guess the number in time

```
Guess the number!
I'm thinking of a number from 1 to 10.
Try to guess it in 10 seconds.

Timer started!
Your guess: 5
Too high.
Your guess: 3
Too high.
Your guess: 1
Too low.
Your guess: 2
You guessed it in 4 tries.

Not fast enough!
Bye!
```

Console when the user guesses the number in time

```
Guess the number!
I'm thinking of a number from 1 to 10.
Try to guess it in 10 seconds.

Timer started!
Your guess: 5
You guessed it in 1 tries.

You did it!
The application will end when the timer finishes.
```

Specifications

- Start with the Guess the Number application presented in chapter 4.
- Create a TimerThread class that sleeps for the specified number of seconds.
- Add code to the Guess the Number application so that it starts the timer thread when the game starts. Then, when the user guesses the number, the application can use the isAlive() method of the timer thread to check whether the thread is still running. If so, the application can display one message. If not, it can display another.

Project 23-2: Tortoise and Hare Race

Create an application that uses threads to simulate a race between two runners that differ in their speed and how often they need to rest.

Console

```
Get set...Go!
Tortoise: 10
Hare: 100
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
Hare: 200
Tortoise: 160
Tortoise: 170
Tortoise: 180
Tortoise: 190
Tortoise: 200
Hare: 300
Tortoise: 210
Tortoise: 220
Hare: I finished!
Tortoise: 230
Tortoise: 240
Tortoise: 250
Tortoise: 260
Tortoise: 270
Tortoise: 280
Tortoise: 290
Tortoise: 300
Tortoise: I finished!
```

Project 23-2: Tortoise and Hare Race (continued)

Specifications

- Each runner should be implemented as a separate thread using a class named RunnerThread. This class should include these instance 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 1000 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 method should add the speed value to the position value and display the new position. The run() method should sleep for 100 milliseconds on each repetition of the loop. When the loop ends, this method should display a message indicating that the runner has finished.
- 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.

Hint

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

Enhancements

- Modify the main class so it runs the race 100 times and reports how many times each runner wins. (To make the application run faster, you may want to reduce the sleep time in the runner threads.)
- Modify the application so it can support up to 9 runners.
- Add an additional random element to the runner’s performance. For example, have a “clumsiness percentage” that indicates how often the runner trips and hurts himself. When the runner trips, he sprains his or her ankle and can run only at half speed for the next five moves.
- Add the ability for runners to interfere with each other. For example, have an “orneriness percentage” that indicates how likely the runner is to trip another runner who is passing him. This will require additional communication among the threads.

More ideas for projects

- If you need more projects for later chapters, you can create them by enhancing any of the projects presented in earlier chapters.
- After chapter 15, you can convert any of the applications that need to store data so they store their data in a text file or a binary file.
- After the chapters on GUI programming, you can have your students convert any of the console applications presented in chapters 1-16 to run as GUI applications.
- After chapter 21, you can convert any of the applications that need to store data so they store their data in a database.