Project 3: Yarn Ball Converter

Due: Monday, Feb. 10 at 11:59 PM

Description: Blankets, pillows, clothing, and many other items can be knit by hand using patterns. A pattern explains how to knit an item and, importantly, how much yarn is needed. Yarn comes in balls, and the total number of yards per ball varies from one type of yarn to another.

Sometimes it's impossible to find the type of yarn that a pattern calls for. When this happens, a different type of yarn must be used. Since yarn types vary in total yards per ball, the number of balls of the substitute yarn will be different than the number given in the pattern.

In this project, you will write a yarn ball conversion program! When the program runs, the user will input (1) the number of yarn balls required by a pattern, (2) the number of yards per ball of this yarn, and (3) the number of yards per ball of a second type of yarn. The program will then output the number of balls of the second type that are needed to substitute it for the first.

Objectives: Your program will be graded according to the rubric below. Please review each objective before submitting your work so you don't lose points.

1. Create a new project and class in Eclipse and add the main method. (10 points)
2. Construct a Scanner to read input from the keyboard. (10 points)
3. Use the Scanner to read the name, number of balls, and yards per ball of a yarn type specified by a pattern. Prompt the user to input this data as shown in the example below, and store the data in variables. (15 points)
4. Use the Scanner to read the name and yards per ball of a substitute yarn. Prompt the user to input this data, and store the data in variables. (10 points)
5. Use conditional statements to check that each numerical input is a positive integer. Give the user one chance to correct each invalid input. Prompt the user to input a positive value as shown in the example. (15 points)
6. Calculate the number of substitute yarn balls. Use a method of the Math class to round up to the nearest full ball, and store the result in a variable. (20 points)
7. Print the number of substitute yarn balls to the console, matching the format of the example. (10 points)
8. Use meaningful variable names, consistent indentation, and whitespace (blank lines and spaces) to make your code readable. Add comments where appropriate to explain the overall steps of your program. (10 points)

Note for CS 1324 Students: If you do not complete the project during lab with your partner, finish writing the program individually during the week and submit before the deadline. Make sure you and your partner both have a copy of your code before leaving. Whether you finish in lab or not, you both must submit the project to Zybooks.
Sample Output: Below is an example run of the program. The user input is in bold. The output of your program should match this exactly when given the same input.

```
Enter the name of the yarn specified by your pattern.
Chunky Alpaca
Enter the number of balls of Chunky Alpaca that are required.
-2
The number of balls must be positive. Please re-enter.
2
Enter the number of yards per ball of Chunky Alpaca.
50
Enter the name of the substitute yarn.
Big and Fluffy
Enter the number of yards per ball of Big and Fluffy.
0
The number of yards must be positive. Please re-enter.
40
You should purchase 3 balls of Big and Fluffy instead of 2 balls of Chunky Alpaca.
```

Notice the following things about this example:

1. The number of balls and yards per ball (of both yarn types) are input as integers. You can assume that the user will always input these values as integers.
2. Error checking is performed on the numerical input. If the user inputs a nonpositive integer, the program gives them one chance to correct their input.
3. The pattern requires 100 yards of yarn. 3 balls of Big and Fluffy yarn have a total of 120 yards, which is more than needed. 2 balls of Big and Fluffy, however, have only 80 yards, which is too little. The exact number of balls needed is 2.5, but it's not possible to buy half a yarn ball. Thus, the calculation must be rounded up to the nearest full ball.

Create a New Project, Class, and Main Method: Refer to the Project 1 handout for instructions on creating a new project and class in Eclipse. Refer to the Project 2 handout for instructions on naming the class and adding the main method.

Construct a Scanner: To get data from the user, you need to construct a Scanner object that reads input from the keyboard. This requires two steps:

1. Unlike the String class, the Scanner class is not contained in the java.lang package, which the compiler imports automatically when you build a program. Instead, the Scanner class is in the java.util package, which you must tell the compiler to import. This can be done by adding the following line to the top of your source file (before any other code):

   ```
   import java.util.Scanner;
   ```

2. Once the Scanner class is imported, a Scanner object can be constructed and assigned to a variable with a line of code like the following:

   ```
   Scanner keyboard = new Scanner(System.in);
   ```

   The code to the left of the assignment operator declares a Scanner variable named "keyboard." You can use a different name, but choose something descriptive. Common alternatives include "input" and "stdin".

The code to the right of the assignment operator constructs the Scanner. The code "System.in" tells the Scanner to read input from the keyboard. In future projects, we will use Scanner objects to read data from text files. In those projects, the code that appears in parentheses will specify the name of the file, rather than the keyboard.

Read User Input: The Scanner class has four read methods that we will use throughout the semester: next, nextInt, nextDouble, and nextLine. The first three of these methods read a single token (i.e., word) of input. The method next reads the token as a String, nextInt reads it as an integer, and nextDouble reads it as a double. Each of these methods stops after it reads a token and then encounters a whitespace character like a space or newline.

The method nextLine reads an entire line of input as a String. It stops after it encounters a newline character, even if it hasn't read a single token.

The methods next, nextInt, and nextDouble throw away any whitespace characters that come before a token. They stop at the first whitespace character after the token and leave this character in the input buffer. The method nextLine, in contrast, simply reads input until it encounters a newline character. Unlike the other three methods, however, it throws away the newline character where it stops.

Handle Newline Characters: Let's consider an example. Suppose the following text is entered by the user:

```
CS 1324<newline>
is my favorite class!<newline>
```

The text <newline> represents a single newline character. These characters are input whenever the user presses the enter key. They are invisible in a text editor, but they are still present in the data. They tell the editor to move down to the next line.

Suppose we want to read the token 1324 and store it in an integer variable named "courseNumber". We then want to read the entire line "is my favorite class!" and store it in a String variable named "evaluation". How do we do this?

Scanner objects read input sequentially, so we first need to get rid of the token "CS". To do this, we call the method next, which reads the token and stops at the first space. The remaining text in the input buffer is the following:

```
 1324<newline>
is my favorite class!<newline>
```

Note that the method stops *before* the first whitespace character after the token. The methods next, nextInt, and nextDouble all behave this way.

Now we can read the token "1324" by calling the method nextInt. This method reads the space before the token, the token itself, and then stops before the newline character (the first whitespace character after the token). The method throws away whitespace characters before the token, so only "1324" is returned. The remaining text in the input buffer is the following:

```
<newline>
is my favorite class!<newline>
```

Now comes the part that trips people up. We want to read the entire final line of text. To do this, we can use the method nextLine; however, we need to call the method twice. The reason for this is that, unlike next, nextInt, and nextDouble, the method nextLine stops as soon as it reads a newline character. Since the first character left in the input buffer is a newline, calling nextLine reads just this character and stops! This leaves the final line in the buffer:

```
is my favorite class!<newline>
```

Now we can read this line by calling nextLine a second time.

In summary, the code needed to read the example input is the following:

```
keyboard.next();  // skip CS
int courseNumber = keyboard.nextInt();  // read 1324
keyboard.nextLine();  // skip <newline>
String evaluation = keyboard.nextLine();  // read the last line
```

The moral of this story is that if you read input in a program using both nextLine and one of the other three Scanner methods, you need to think carefully about newline characters. In particular, if you read a token at the end of a line with next, nextInt, or nextDouble and then want to read the entire proceeding line, you will need to call nextLine twice.

Check for Input Errors: For most of the programs we write in this class, we will assume the user only enters valid data. In this project, however, the user can enter a non-positive integer for the number of yarn balls or the number of yards per ball (of both yarn types). You should check for this with conditional statements. Assume that if an integer input is non-positive, the user will enter a positive value next time. (In order to handle the general case, where the user repeatedly enters invalid data, we need loops, which we won't discuss until after the midterm.)

Perform the Yarn Ball Conversion: To calculate the number of balls of the substitute yarn, you will need to perform multiplication, division, casting, and rounding. Pick some test values and figure out the calculation on paper before writing code. The following suggestions may help:

1. Use double (not integer) division to calculate the exact number of substitute yarn balls. Then round up to the nearest full ball. Note that the user will enter all numbers as integers. How will you perform the calculation so the result is a double?
2. There is a method in the Math class you can use to round up. (It's not the round method.) When you find this method, note that its return type is double, not int. We want the number of substitute balls to be stored in an int variable. How will you do this?

Upload Code to Zybooks: After you complete each project, you need to upload your source code to Zybooks so it can be graded. If you created the folder structure described earlier, your code is in the folder "Intro to Programming\Projects\Project 3\src". The code is contained in the file Project_3.java. Note that .java files are simply text files that contain Java code. They can be opened with any text editor (e.g., Notepad or TextEdit).

Upload your .java file to Zybooks on the Project 3 assignment page. This requires a few steps. Click the "Submit Assignment" button in the top-right corner. This will reveal a button near the bottom of the page with the text "Choose File." Click this button and browse to the location of your .java file. Finally, click the “Submit for grading” button below the “Choose on hard drive” button.

If Zybooks won’t accept the file, make sure you’re submitting a .java file, not a .class file. Make sure to see if the output you obtained and the output which I have mentioned match to obtain complete grade.

Submission Link:
https://learn.zybooks.com/zybook/OUCS1324Winter2020/chapter/21/section/2