Practical 2

Conditionals, repetition statements, methods, and random numbers in Java

Exercises

Download the file "Week-2-Examples.zip" from the COMP122 website. This contains all of the programs listed under "Week 2 Example Programs" in one zipped file. By unzipping this file in a suitable location in your (CS or university) filestore (or laptop/desktop computer where you have Java installed), you can experiment with these files.

In a similar fashion "Lab-2-Examples.zip" contains some code (or partial code) for some of the exercises in this second practical.

Note: While I have provided "Model Solutions" to these Lab questions online, you should attempt these exercises before looking at my solutions. And, of course, my solutions need not be the only way that they can be done.

1. Modify the program "DrunkardWalk.java" (which can be found in the Week 2 example files) so that the random walker does not go beyond the pub. In other words, the random walker starts at the integer 5, and cannot go beyond the integer 5 (unlike in the original program where he could, in principle, go to an arbitrarily large number). What this means is that if the random walker tries to go beyond 5, he instead stays where he is for that step.

   Hint: Use the `Math.min` function in an appropriate location. Doing so means the change is pretty minimal to the code. There are, of course, other ways to achieve this same result.

2. Look at the program "SumAndAverage.java" from this week's example files (in "Week-2-Examples.zip").

   (a) There is a line of code near the end of the program that is:

   ```
   System.out.println("Average␣is:␣" + (1.0*total/count));}
   ```

   Why is the number 1.0 in that line? What purpose is it serving (being multiplied by the value `total`)?
   Would this line of code have the same effect in this program?

   ```
   System.out.println("Average␣is:␣" + 1.0*(total/count));}
   ```

   (b) Modify "SumAndAverage.java" to take input from the user that includes both positive and negative integers, i.e. it will end only when the user inputs 0. Again, like in question (1), the needed change to the code is actually rather small.

3. Before you start working on this question, read all parts of it. You're going to write some methods and a program described in part (c) that will use those methods, but you might first start by writing a program that (kind of) works for part (c) of this question, and then add the methods described in parts (a) and (b).

   (a) Write a method called `getIntegerInRange` that takes three parameters, `lower` (an **`int`**), `upper` (another **`int`**), and `text` (a **`String`**), and returns an **`int`** from the method call. This method should display the `text` and ask for input from the user until the input is bigger than or equal to `lower` and less than or equal to `upper`. Display some appropriate "error message" when the input is out of range.

   So the method declaration should look like this:

```
static int getIntInRange(int lower, int upper, String text)
{

        //  Your code in here...
}
```

   As an example, you can make a method call like

```
int month = getIntInRange(1, 12, "Enter the month of your birth: ");
```

   to get an integer between 1 and 12 (inclusive).

   The user might see output like the following (as part of the execution of a program that uses this method):

```
Enter the month of your birth:  0
Sorry, that input is invalid!
Enter the month of your birth: -10
Sorry, that input is invalid!
Enter the month of your birth: 10
```

   (b) Write another method that uses a **`switch`** statement that returns a string that corresponds to the birth month. Call this method `getMonthString` that takes one parameter, `month` (an **`int`**), as input and returns a String. (Make certain to use the modifier **`static`** in your method definition, otherwise you will get an error message from the method call in your program.)

   (c) Write a program that uses the methods from part (a) and (b) to get the birth month, day, and year of someone, and displays that date as a string like "12 October 1975" if the corresponding inputs are 10, 12, and 1975 for `month`, `day`, and `year`.

   (Don't (yet) worry about checking if the user is trying to enter a date like September 31, or if the year is a leap year to determine if February 29 is valid or not. But you can see that dealing with dates can already get complicated...)

   So a user might see output of the form if he/she runs your program:

```
Enter the month of your birth (as an integer): 0
Sorry, that input is invalid!
```

```
Enter the month of your birth (as an integer): 10
Enter the day of your birth: 12
Enter the year of your birth: 1975

Your date of birth is 12 October 1975.
```

4. In the Gregorian calendar, a leap year is a year that is divisible by 4, except if it is also divisible by 100, in which case it's a leap year only if it's divisible by 400. So 1980, 2000, and 2016 are (or were) all leap years, while 1900, 2001 and 2018 are not.

   Write a program that will take input from the user (an integer) and will print out whether or not that year is a leap year. In your program, you should write a separate method called `isLeapYear` that takes one argument (an **`int`**) and returns **`true`** or **`false`** as appropriate.

   (Hint: Think carefully how to write some tests to cover all cases. And make sure you get all the parentheses in the correct places when you write your test in Java.)

5. Have a look at the program "RandomNumbers.java" in the example files for this week's practical ("Lab-2-Examples.zip"). Note how we can use the `Math.random()` function, with some scaling, to generate random integers. (That method was also used in the examples from class, and in the "DrunkardOne.java" and the other related examples about the "Drukard's Walk".) This particular example program will generate ten random numbers from the set of integers $\{1, 2, \ldots, 5\}$ and sum them up.

   Modify "RandomNumbers.java" so that you will take input from the user, both for the quantity of numbers to generate, and the range of numbers to generate them from.

   Let us specify that the quantity of numbers to *generate* is at least 0 and no more than 100, and that the *range* of integers is $\{1, \ldots, n\}$ for some positive integer $1 \leq n \leq 50$.

   (Hint: Use your `getIntInRange` function from part (3a) to help modify your program to satisfy these requirements. And don't forget to **`import`** `java.io.*;` in order to read input from the user with the `Scanner` class.)

6. The Java `Math.random()` method returns **`double`** values that are (approximately) uniformly distributed on the interval $[0.0, 1.0)$. By scaling, for example, we can create (approxiamately) unifrom integers on a range, as was done in "DrunkardWalk.java" and in the "RandNumbers.java" program.

   The `Random` class also implements a (pseudo)random number generator, and provides many more methods for generating random **`int`** values, **`long`** values, etc. You can import this class by including the line

   ```
   import java.util.Random;
   ```

   in your code (or **`import`** `java.util.*;` will also work, and give you a lot more stuff too, including the `Scanner` class).

See the program "RandomNumbers2.java" in the example files for this week. This program basically duplicates the functionality of "RandomNumbers.java", but uses an instance of the `Random` class to do the work of generating random **`double`** values instead of the `Math.random()` method. Aside from that one change, the rest of the code is the same.

In particular, we replace the line

```
number = (int)(Math.random()*SCALE + 1);
```

with

```
number = r.nextInt(SCALE) + 1;
```

to generate a (pseudo)random integer in the range $\{1, \ldots, \text{SCALE}\}$. We have to first create an instance of the `Random` class to do so, which is the variable `r` in the line of code above.

See the code in "RandomNumbers2.java" for this syntax. Basically only two lines of code are different between "'RandomNumbers.java" and "RandomNumbers.java" (aside from the additional **`import`** statement at the top of the "RandomNumbers2.java" to import this new class).

(a) Write a program that will generate a random integer between 1 and 100 (or some other bounds you choose). Have a user input guesses until they get it correct. The program should take input and repond "Too high!" or "Too low!" until the user guesses corrctly. The program should report on the number of guesses the user needed to guess correctly. Use the `Random` class to create the integer that the user will guess. (Again, see the code of "RandomNumbers2.java" to see how this is done.)

(b) Ok, now put your "game playing" routine into a separate method in your program. Call it, say, `game`, and make it a **`void`** method (i.e. a method that returns no value when it is called).

Alter your program so that when the game is done, the program will ask if the user wants to play again. Suppose the user supplies a character "y" or "n" in reply, and the program will play again, or not, as appropriate.

(Hint: The `Scanner` class **does not** have a `nextChar()` method. To read a single **`char`** value, you can use this code:

```
c = input.next().charAt(0);
```

to get the first letter of a string that a user has entered, where `input` is an instance of the `Scanner` class.)

(c) **BONUS QUESTION:** How many guesses (at most) will it take to find the computer's number if the range is $\{1, \ldots, 100\}$? How about if the range is $\{1, \ldots, 200\}$? Or $\{1, \ldots, 1000000\}$? How about if the range is $\{1, \ldots, n\}$ for some positive integer n?

7. Write a program that will take input from a user that consists of an **`int`** between 0 and 10000, which we will call n, and a real number (a **`double`**), which we will call p. We will assume that $0 \leq p \leq 1$.

(Note, the `Scanner` class has a method `nextDouble()` that will read a **`double`** from user input. If you want, you can modify your `getIntInRange` method to create a similar method called `getDoubleInRange` that you can use to get a double in a specified range of numbers to guarantee that $0 \leq p \leq 1$.)

Your program should simulate flipping a coin n times that will come up "heads" with probaility p for each flip (independently of any other flip). Assume you start at the integer 0 and add 1 each time the coin comes up "heads", and subtract 1 each time the coin comes up "tails".

Print out the final number you get after the n coin flips are done.

Use the `nextDouble()` method in the `Random` library to simulate the coin flips. So, like in "RandomNnumber2.java", or in Problem (6), you need to import that library, and create an instance of the `Random` class, for example:

```
Random r = new Random();
```

Then `r.nextDouble()` will get the next (pseudo)random **`double`** value.