Assignment 04

CS 120: Software Design I

Program 1: `Lucas`

Lucas numbers[1] are a sequence of numbers in mathematics. These numbers have been used in cryptography and have many interesting properties, including their use in calculating the golden ratio[2], considered to be such an aesthetically pleasing ratio that numerous artists have based the details of their work off of it.

To calculate the Lucas numbers, start with $L_0 = 2$ and $L_1 = 1$. Subsequent numbers are calculated using the following formula:

$$L_n = L_{n-1} + L_{n-2}$$

So, for example, $L_2 = 3$ and $L_3 = 4$. The first ten Lucas numbers can be found in Example 1 below. Your program will take as input the upper bound for what Lucas numbers to print, meaning that no Lucas numbers should be printed that is larger than that number (but it can be equal to the upper bound). Notice how Example 1 prints the number 76, but the next Lucas number, 123, because 123 is larger than 100, the upper bound provided.

Below are three example runs of the program with input `Highlighted`. Make sure that your output matches the examples below (except for the sections determined by user input).

Example 1

```
Upper bound: 100

2, 1, 3, 4, 7, 11, 18, 29, 47, 76
```

Example 2

```
Upper bound: 3

2, 1, 3
```

Example 3

```
Upper bound: 2

2, 1
```

Some prompts and questions to consider *before* starting to write the program:

- How can you use variables to track to values of L_n, L_{n-1}, and L_{n-2}? What should the starting values of L_{n-1} and L_{n-2} be?
- How will you know to stop printing numbers?

The following is a high level checklist of requirements for your program:

- ☐ Your class is named `Lucas` and is placed in the project created for this assignment
- ☐ Your code is commented according to the guidelines in the Java Style Guide found on D2L

[1] Read more: `https://en.wikipedia.org/wiki/Lucas_number`
[2] Read more: `https://en.wikipedia.org/wiki/Golden_ratio`

- ☐ Your code is formatted according to the guidelines in the Java Style Guide found on D2L
- ☐ Your code fulfills the functionality outlined above
- ☐ You have a blank line between the prompts and the output
- ☐ Your strings (both prompts and output) are formatted **exactly** as shown in the examples

Program 2: `ReplaceAll`

Earlier in the semester, we learned about the `replaceAll` method that is used with strings. In this program, you will be writing the code to replace all the text in the string that matches a pattern with some replacement text, the printing the new string, exactly like the `replaceAll` method. Your program will ask for three values: the input text to modify, the pattern to match in the input, and the string to replace the pattern with. Note that the pattern to match is case sensitive. Your program **cannot** use the `replaceAll` method or any variants of it provided by Java.

Below are four example runs of the program with input `Highlighted`. Make sure that your output matches the examples below (except for the sections determined by user input).

Example 1

```
Text to modify: A pattern to match.
Characters to replace: t
Characters to replace "t" with: !

A pa!!ern !o ma!ch.
```

Example 2

```
Text to modify: Nothing to match here...
Characters to replace: z
Characters to replace "z" with: +

Nothing to match here...
```

Example 3

```
Text to modify: Replacing more than you got.
Characters to replace: o
Characters to replace "o" with: ***

Replacing m***re than y***u g***t.
```

Example 4[3]

```
Text to modify: Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo
Characters to replace: Buffalo
Characters to replace "Buffalo" with: $

$ buffalo $ buffalo buffalo buffalo $ buffalo
```

Some prompts and questions to consider *before* starting to write the program:

- Consider a different problem first: given the first two inputs, how can you print "Match found!" if the input contains the pattern to replace at all (without using the `replaceAll` method)? Note that this program would print "Match found!" once for Examples 1, 3, and 4, but not for Example 2. How can you print "Match found!" each time the pattern is found in the input string? Note that this program would print "Match found!" four times for Example 1, no times for Example 2, three times for Example 3, and three times for Example 4. You might find it useful to write these two programs first.

[3]Read more: `https://en.wikipedia.org/wiki/Buffalo_buffalo_Buffalo_buffalo_buffalo_buffalo_Buffalo_buffalo`

- What method can you use to get an individual `char` at a specific position from a `String`?
- Remember that you can check if two `char` values are the same using `==`.

The following is a high level checklist of requirements for your program:

☐ Your class is named `ReplaceAll` and is placed in the project created for this assignment

☐ Your code is commented according to the guidelines in the Java Style Guide found on D2L

☐ Your code is formatted according to the guidelines in the Java Style Guide found on D2L

☐ Your code fulfills the functionality outlined above

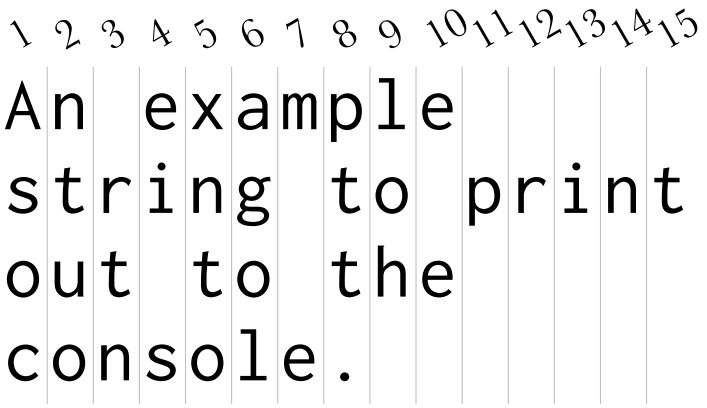
☐ The `replaceAll` method and its variants are **not** used in your program.

☐ You have a blank line between the prompts and the output

☐ Your strings (both prompts and output) are formatted **exactly** as shown in the examples

Program 3: `FlushLeft`

Aligning text is a common action in text processing programs. Alignment can be flush to the left, the right, centered, or justified (i.e., taking up all the space on the line, like the text in this document). You will be writing a program named `FlushLeft` that will take in text and a column width measured in characters, then flush text to the left and leave the right side of the text ragged (i.e., uneven) according to the column width. Note that there will never be a leading space (i.e., a line will never start with a space). Below, you can see an example of the text "An example string to print out to the console." flushed left with a column width of 15, with each character annotated with its column number (note that the space between each character is exaggerated for the sake of readability in this example):



You can assume the input to this program is one or more sentences (concatenated into a single string) formatted as is common in the English language. This means spaces will only occur between words and sentences (not at the beginning or end of the input), and there will **never** be more than one space in a row. You can also assume that none of the words in the input will be longer than the output width (i.e., if the column width is five, you will never have a word longer than five characters).

Below are four example runs of the program with input `Highlighted`. Make sure that your output matches the examples below (except for the sections determined by user input).

Example 1

```
Enter the text to justify: An example string to print out to the console.
Enter the output column width in terms of number of characters: 15

An example
string to print
out to the
console.
```

Example 2

```
Enter the text to justify: four char test
Enter the output column width in terms of number of characters: 4

four
char
test
```

Example 3

```
Enter the text to justify: A short sentence.
Enter the output column width in terms of number of characters: 20

A short sentence.
```

Example 4

```
Enter the text to justify: four char test part deux
Enter the output column width in terms of number of characters: 7

four
char
test
part
deux
```

Some prompts and questions to consider *before* starting to write the program:

- Consider printing only the first line of the output. How do you know where in the input to stop printing? How do you know what will still need to be printed after the first line?
- Once you have mastered the first line, how can you repeat your algorithm for subsequent lines too?
- You might consider writing instructions for a friend on how to flush left some text. Think about how specific you need to be; treat your friend like a computer. Pretend that your friend doesn't know what a word is; how would you explain to a computer what a word is? (hint: think about words in terms of individual characters, including what characters surround a word)

- Be sure to consider different situations for the inputs! For example, consider the input "A new car." What is the output for 3 characters? 5 characters? 6 characters? 9 characters? Examples 2 and 4 also offer one possibility for testing whether your algorithm holds up under different width conditions.
- How will you ensure that spaces do not appear at the beginning of the line?

The following is a high level checklist of requirements for your program:

☐ Your class is named `FlushLeft` and is placed in the project created for this assignment

☐ Your code is commented according to the guidelines in the Java Style Guide found on D2L

☐ Your code is formatted according to the guidelines in the Java Style Guide found on D2L

☐ Your code fulfills the functionality outlined above

☐ You have a blank line between the prompts and the output

☐ Your strings (both prompts and output) are formatted **exactly** as shown in the examples