Learning Outcomes

- Design and implement recursive algorithms
- Design and implement iterative algorithms
- Compare iterative and recursive approaches to different methods

Pre-Lab

- Create a new Java project called Lab9
- Download the file: FibonacciDemo.java
- Save these downloaded files into the Lab9 src folder

Exercise 1 – Iterative vs. recursive methods

For some problems it is possible to design natural algorithmic solutions that are recursive and solutions that are iterative. One of these problems is that of computing the n-th Fibonacci number. The Fibonacci numbers are: 1, 1, 2, 3, 5, 8, 13, 21…

1. Open FibonacciDemo.java and examine the code. Carefully read through the `ifib()` and `rfib()` methods to see the iterative and recursive approaches to computing the Fibonacci numbers respectively. Why does the iterative approach require so many variables? What are the base cases of the recursive Fibonacci method?
2. Run the program and when asked enter the number 20, 30, 40, 45, 50. Write down the running times reported by the algorithm.
3. Modify the program to include a new variable called methodCalls2 that keeps track of the number of calls made to method `rfib()` when the value of the parameter n is 2 and prints it out in the `main()` method. Run the program and enter the value 40. How many calls are made to `rfib()` when the value of the parameter n=2? Write down this result.
4. Why is the recursive algorithm for computing Fibonacci numbers so slow compared to the iterative algorithm? Does the value of methodCalls2 make sense with this rationale?

Exercise 2 – More iterative vs. recursive methods

1. Create a new class called Abo.java.
2. Consider a mathematical series, Abo, defined as:
   - Abo(n) = 0 for n <= 0
   - Abo(1) = 1
   - Abo(n) = 1 + Abo(n/2), if n > 1 is even
   - Abo(n) = 2 + Abo((n+1)/2), if n > 1 is odd
3. In the Abo class, create a method called `rabo(int n)` that uses recursion to calculate the value of Abo(n) for a given integer n.
4. Add a `main()` method and print out the first 20 Abo numbers in the series, i.e. Abo(0) through Abo(19).
   Note: the results should be: 0, 1, 2, 4, 3, 6, 5, 5, 4, 8, 7, 7, 6, 7, 6, 6, 5, 10, 9, 9
5. Create a method called `iabo(int n)` and **try** to calculate the series iteratively.
6. It is not impossible to do, but it is much more complex than the recursive method. Why is this algorithm difficult to design using the iterative approach?

Exercise 3 – Permutations and Recursion

Given a sequence of characters, we want to find and print all possible permutations of these characters. For example, for the sequence of characters "car", the possible permutations are "car", "cra", "acr", "arc", "rca", and "rac". (Note that they don't have to be actual words.)

1. Create a new class called Permutations.java.
2. Within this class, write a recursive method called `permutations(String prefix, String suffix)` that receives as parameters a String prefix and a String and it prints all permutations of the characters in suffix. The prefix parameter will be empty initially and then change upon recursive calls to the method (more info below).
3. Parameter prefix will store a permutation of several of the characters of the original string, while suffix will store the remaining (and not yet permuted) characters of the original string. Use the following hints and explanations with this method:
   - The base case is when suffix is empty. In this case prefix contains a permutation of all characters of the original string, so the algorithm must just print prefix.
   - If suffix is not empty, then the algorithm will consider each one of the characters c of suffix and for each one of them it will do the following:
     - remove character c from suffix to get a new string suff
     - append c to prefix to get a new string pre
     - invoke `permutations(pre, suff)`
4. Method `length()` returns the length of a string, and method `charAt(i)` returns the character of a string at index i. To concatenate a character c to a string prefix use c+prefix. The following algorithm can be used to remove the character at index i from a

string s:

```
private static String removeChar(String s, int i) {
    return s.substring(0, i) + s.substring(i+1, s.length());
}
```

5. You also need to write a method that asks the user to enter a string, and it prints all permutations of the string's characters by calling `permutations()` with that string. The initial prefix parameter will be an empty string.
6. This is another example of a problem that would be very difficult to solve using an iterative algorithm. However, the following short recursive algorithm can be used to solve this problem.

Submission

When you have completed the lab, navigate to the weekly module page on OWL and click the Lab link (where you found this document). Make sure you are in the page for the correct lab. Upload the files listed below and remember to hit Save and Submit. Check that your submission went through and look for an automatic OWL email to verify that it was submitted successfully.

Rules

- Please only submit the files specified below. Do not attach other files even if they were part of the lab.
- Do not ZIP or use any other form of compressed file for your files. Attach them individually.
- Submit the lab on time. Late submissions will receive a penalty.
- Forgetting to hit "Submit" is not a valid excuse for submitting late.
- Submitting the files in an incorrect submission page will receive a penalty.
- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular lab deadline will receive a penalty.

Files to submit

- FibonacciDemo.java
- Abo.java
- Permutations.java