Homework #1 Instruction
<Data Structures, CSCI-UA 0102-009>

- **Attention**
  1. Homework must be completed individually! (Group work is not allowed.)
  2. For implementation assignments, if you successfully pass the test cases in each ~Demo.java and faithfully implement the instructions specified in the comments for each function, you can receive a full score on the Autograder as well.
  3. Feel free to post any questions or concerns you have during the implementation process in the Ed discussion. Discussions among students are also welcome.
  4. However, please be cautious not to directly share your code with other students. The Autograder automatically measures the similarity of codes between students. If an excessively high similarity is detected, it may lead to strong penalties after discussion with the TA.
  5. Also, a 50% penalty will be applied for late submissions.

Part 1: Time Complexity [30 points]

Read the following pseudo code and calculate the Time Complexity (Big-O) for the input size (n). **You should submit the answer to this on Gradescope under HW1 – Part 1**.

Problem 1 [10 points]

```
public int fn(int n){
    int num = 0;
    int k = 10
    for (int i = 0; i < k; i++)
        num += n;
    return num;
}
```

(1) $O(1)$
(2) $O(n)$
(3) $O(2^n)$
(4) $O(n!)$
(5) $O(n^2)$

Problem 2 [10 points]

```
public int fn(int n){
    for (int i = 0; i < n; i++)
        fn(n-1);
    return n;
}
```

(1) $O(1)$
(2) $O(n)$
(3) $O(2^n)$
(4) $O(n!)$
(5) $O(n^2)$

Problem 3 [10 points]

```
public int fn(int n){
    int i, j, m = 0;
    for (i = 0; i < n; i++)
        for (j = 0; j < i; j++)
            m += 1;
    return m;
}
```

(1) $O(1)$
(2) $O(n)$
(3) $O(2^n)$
(4) $O(n!)$
(5) $O(n^2)$

Part 2: Recursive function [20 points]

Read the following problem and write a solution code using a recursive function. Make sure to solve the problem using recursion, you don't need to be overly concerned about the time-out limit. **You should submit the answer to this on Gradescope under HW1 – Part 2**.

Problem 1: Climb Stairs

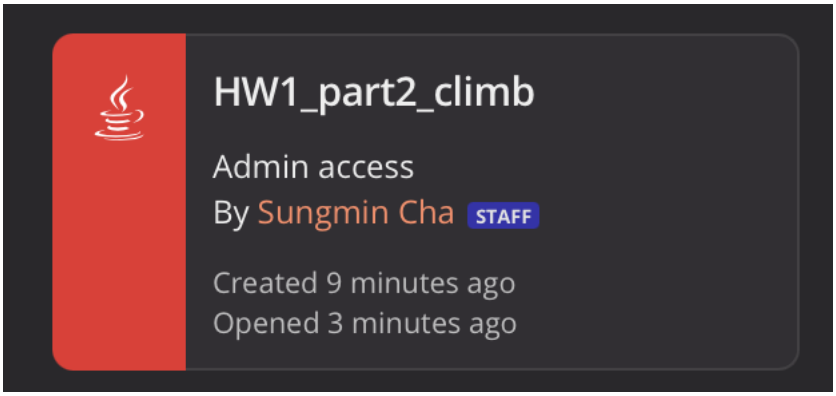
You are climbing a staircase. Let's say you need to climb n stairs to reach the top. If you can climb 1 or 2 stairs at a time, write a function to calculate the number of distinct ways to reach the top. Also, consider that the input values (n) are always limited to integers between 0 and 45.

Example
:
Input: 4
Output: 5

Explanation: If you start from the ground floor and climb up to the 4th floor, there are a total of 5 different paths:

- 1 step + 1 step + 1 step + 1 step
- 1 step + 1 step + 2 steps
- 1 step + 2 steps + 1 step
- 2 steps + 1 step + 1 step
- 2 steps + 2 steps

You can fork the example code from **ED Workspace/HW1_part2_climb**.



Then, you can compile and run the code with the below commands:

```
[user@sahara ~]$ javac hw1/Climb.java
[user@sahara ~]$ java hw1/Climb
Enter an integer between 0 and 45: 20
0
```

*Penalty

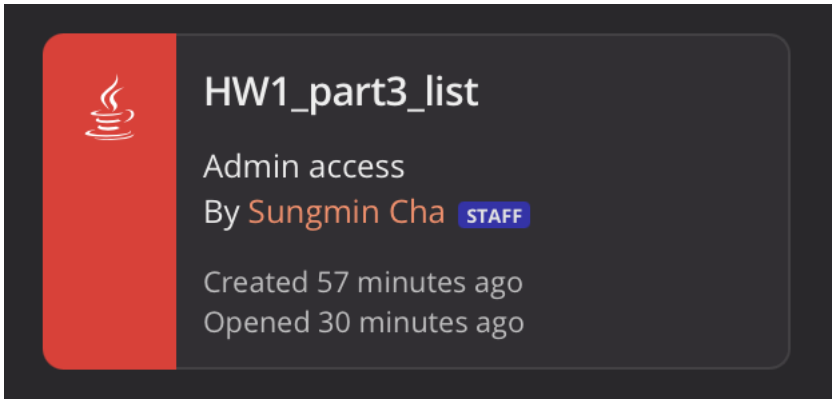
If you don't use a recursive function, there will be a penalty in the final evaluation score.

Part 3: Array and Linked List [50 points]

Please read the instructions below and implement CustomArrayList and CustomLinkedList.

* Fork and Usage

1. You can fork the template code for implementation from **ED workspace/HW1_part3_list**.



2. Compile and execute the above code using the following command.

```
[user@sahara ~]$ javac customlist/CustomArrayListDemo.java
[user@sahara ~]$ java customlist/CustomArrayListDemo
Array List Demo!
Print list (#items=0)
Print list (#items=0)
Print list (#items=0)
Print list (#items=0)
Print list (#items=0)
Print list (#items=0)
get(1) : null
Print list (#items=0)
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "jav
"customlist.CustomArrayList.get(int)" is null
        at customlist.CustomArrayListDemo.main(CustomArrayListDemo.java:47)
```

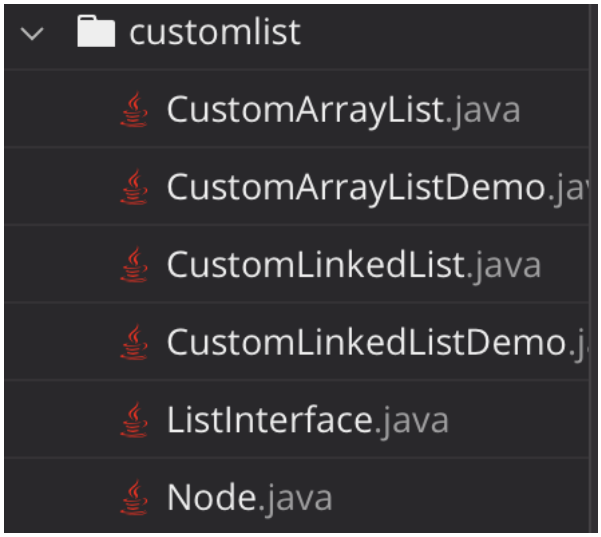
* Generic Usage and Notes

When implementing a List, fixing the element type such as Integer can be a drawback. This is because, if you want to create a list with a different data type, you need to create a new one for each data type,

such as StringArrayList, FloatArrayList, HouseArrayList, etc. To overcome this drawback, you can use Java Generics, which allows you to parameterize the data type and later decide the parameter type when creating objects of the class. Using generics in the implementation of List ensures not only universality for various data types but also provides strong type-checking features. Reference materials for the Generics usage are provided below.

[Example and tutorial]
[Youtube Video: Generics in JAVA - Full Simple Tutorial]

* Class Composition and Details for Each Code



1. ListInterface.java

```java
public interface ListInterface<E> {
    public int add(int i, E x);
    public int append(E x);
    public E remove(int i);
    public boolean removeItem(E x);
    public E get(int i);
    public int set(int i, E x);
    public int indexOf(E x);
    public int len();
    public boolean isEmpty();
    public void clear();
}
```

: It provides details for interfaces required for implementing CustomArrayList and CustomLinkedList.

2. Node.java

```java
public class Node<E> {
    public E item;
    public Node<E> next;

    public Node(E newItem) {
        item = newItem;
        next = null;
    }

    public Node(E newItem, Node<E> nextNode) {
        item = newItem;
        next = nextNode;
    }
}
```

: It implements the Node class. The node consists of constructors with one parameter (assigning only the item value) and two parameters (assigning the item value and next value).

3. CustomArrayList.java

```
public class CustomArrayList<E> implements ListInterface<E> {

    private E item[];
    private int numItems;

    private static final int DEFAULT_CAPACITY = 10;

    // Constructor 1: Creating an ArrayList with DEFAULT_CAPACITY.
    public CustomArrayList() {
        item = (E[]) new Object[DEFAULT_CAPACITY];
        numItems = 0;
    }

    // Constructor 2: Creating an ArrayList of size n.
    public CustomArrayList(int n) {
        item = (E[]) new Object[n];
        numItems = 0;
    }
```

: This is the class for implementing CustomArrayList. Two constructors for different cases and a function to print the list are already implemented. Students need to implement the remaining functions. Detailed instructions for implementation are specified as comments above each function.

4. CustomArrayListDemo.java

```java
public class CustomArrayListDemo {
    public static void main(String[] args) {
        System.out.println("Array List Demo!");
        CustomArrayList<Integer> list = new CustomArrayList<>();

        // add()
        list.add(0, 300);
        list.add(0, 200);
        list.add(0, 100);
        list.printAll(); // expected output: "Print list (#items=3) 100 200 300"

        // append()
        list.append(500);
        list.append(600);
        list.printAll(); // expected output: "Print list (#items=5) 100 200 300 500 600"
```

: This class is for the usage and evaluation of CustomArrayList. The main function creates a CustomArrayList and tests each operation. The expected output after executing these operations should match the comments' specified expected output.

5. CustomLinkedList.java

```java
public class CustomLinkedList<E> implements ListInterface<E> {
    private Node<E> head;
    private int numItems;

    // Constructor 1: Creating an LinkedList with Dummy Head.
    // We consider that the index of Dummy Head as '-1'.
    public CustomLinkedList() {
        numItems = 0;
        head = new Node<>(null, null);  //  Dummy Head (index = -1)
    }
```

: This class is for implementing CustomLinkedList. The constructor and the function to print the list are implemented. Students need to implement the remaining functions. Note that a dummy node is used as a default. Detailed instructions for implementation are specified as comments above each function.

6. CustomLinkedListDemo.java

```java
public class CustomLinkedListDemo {
    public static void main(String[] args) {
        System.out.println("Linked List Demo!");
        CustomLinkedList<Integer> list = new CustomLinkedList<>();

        // add()
        list.add(0, 300);
        list.add(0, 200);
        list.add(0, 100);
        list.printAll(); // expected output: "Print list (#items=3) 100 200 300"

        // append()
        list.append(500);
        list.append(600);
        list.printAll(); // expected output: "Print list (#items=5) 100 200 300 500 600"
```

: This class is for the usage and evaluation of CustomLinkedList. The main function creates a CustomLinkedList and tests each operation. The expected output after executing these operations should match the comments' specified expected output.

* Uploading Code Guideline

To submit your code to Gradescope, please compress the directory containing the code, named "customlist" (do not change the directory name), into a zip file named "customlist.zip". And then, submit this zip file under "HW1 – Part 3".