Sit-in Lab #4

CS1020 AY2010/11 Semester 2
Date: 7 April 2011, Thursday
School of Computing, National University of Singapore

Instructions

There are **two** exercises in this lab. Time limit is **1 hour 40 minutes**.

Exercise 1: Top 2 Values [60 marks]

You are asked to extend the Java Stack class to a `Top2Stack` class. `Top2Stack` class has all of the basic stack operations (`push`, `pop`, `peek` and `empty`) and in addition, contains new methods `getMax`and `get2ndMax`that can efficiently return the maximum element, 2nd maximum items in the stack respectively. In addition, `deleteMax` method will delete the maximum item from the stack.

Now let's see technically how we can achieve this.

First of all, `Top2Stack` class inherits Java Stack class, therefore *push*, *pop* or *peek* operations will be inherited (however later on you will see that some of these inherited methods need to be overridden).

Secondly, `Top2Stack` class internally maintains a second Java stack called `sortedStack` as its data field. As the name suggests, values in this `sortedStack` is always sorted in descending order from stack top to bottom. If a user program wishes to learn the maximum value in a `Top2Stack` object, we simply return the topmost value in its associated `sortedStack` – very fast.

```
class Top2Stack<E extends Comparable<? super E>> extends Stack<E> {

    private Stack<E> sortedStack = new Stack<E>(); // an internal sorted stack

    public E getMax() { return sortedStack.peek(); }
    public E get2ndMax() { /* Part A: code to be filled in by you. 10% */ }
    public E deleteMax() { /* Part B: code to be filled in by you. 10% */ };

    public E push(E item) {
        // push item onto the top of this Top2Stack object, code provided already
    }
    private void push(E item, Stack<E> temp) {
        // Part C: update sortedStack accordingly, code to be filled in by you. 20%
    };
    public E pop(E item) {
        // pop and return topmost item from this Top2Stack object, code provided
    };
    private void pop(E item, Stack<E> temp) {
        // Part D: update sortedStack accordingly, code to be filled in by you. 20%
    };
}
```

However, to enjoy such a fast *retrieval* of maximum, 2nd maximum value, we need to bear some overhead: each time a driver program *updates* the contents of a `Top2Stack` object, internally we need to update its `sortedStack` so that they are always synchronized. See the example on next page for a demonstration.

Example:

Suppose a driver program enter values: 4 8 5 6 2 into a `Top2Stack` stack in that order. The following diagram illustrates the contents of a `Top2Stack` object and its internal data structure: `sortedStack`.

Top2Stack object

sortedStack that maintained by a Top2Stack object **internally**

Next, the driver program calls `pop` method which will remove value 2. The following diagram illustrates the updated scenario. Note that `sortedStack` is still sorted and the bottom value 2 is removed.

Top2Stack object

sortedStack that maintained by a Top2Stack object **internally**

Next, the driver program calls `deleteMax` method. The following diagram illustrates the updated scenario. Note that `sortedStack` just pop the topmost item while `Top2stack` object will need to process a bit and remove 8.

Top2Stack object

sortedStack that maintained by a Top2Stack object **internally**

Finally, the driver program calls `getMax` and `get2ndMax` and we simply return the two top values in the `sortedStack`.

Note that `sortedStack` stack is an internal data structure kept by a `Top2Stack` object, for the purpose of fast retrieval of max and 2nd max value in the stack. Such a private data structure is hidden from the view of a driver program. From a driver program's point of view, a `Top2Stack` object is just a Java stack with additional functionalities such as `getMax`, `get2ndMax` and `deleteMax` – but not aware how these methods are implemented internally – this is the idea of Abstract Data Type.

Also note that there could be several ways to efficiently retrieve max and 2ndmax values from a stack. The "internal sorted stack" approach as we use in this lab is just one of them.

Assumptions:

- There are no duplicated values in input data.
- No error checking is needed. For example, you can safely assume that the driver program won't call `get2ndMax` method if a stack contains less than 2 values; won't call `deleteMax`, `getMax`, `pop`, if a stack is empty.

Coding Restrictions:

- You are not allowed to define and use any additional data structure (e.g., array, list, vector, stack, queue…) anywhere by yourself.
- You must use recursion in writing two private methods: `push` and `pop`. Strictly no appearance of any loops inside these two methods.
- Do NOT modify any place of `Top2Stack.java` that you are not asked to.
- Two test cases are hardcoded in the driver program `TestStack.java`. This file will not be collected and uploaded to CourseMarker. Therefore you can feel free to modify it for your own testing.

Sample Run:

If you implement `Top2Stack.java` correctly, by issuing the command `Java TestStack` after compilation, you should observe the following output:

```
//////// TESTING DATA GROUP 1 ///////
Max    = 7
2ndMax = 6
Top2Stack object: 5 6
Internal sortedStack: 6 5
//////// TESTING DATA GROUP 2 ///////
Max    = d
2ndMax = a
Max    = e
2ndMax = d
Top2Stack object: b c d
Internal sortedStack: d c b
```

Exercise 2: N Choose K [40 marks]

Find out all possible combinations of choosing *K* letters out of an input string of *N* distinct letters.

The input consists of an integer, *K* and a string of *N* distinct lowercase letters listed in alphabetical order. Assume that $1 \le N \le 26$ and $1 \le K \le N$. Print out all distinct letter combinations in alphabetical order: every combination can be represented as a string consisting of K letters listed in alphabetical order.

You are to complete one method `kOutOfN` only. No helper method is needed; you are also not allowed to modify any other parts of the program.

Sample Runs

Three sample runs of the program are given below. Outputs are shown in bold.

```
2 abcd
ab
ac
ad
bc
bd
cd
```

```
4 abcd
abcd
```

```
1 abcd
a
b
c
d
```

Marking Scheme

The allocated marks are as indicated in each part of Ex1 and Ex2. Please note that 20% of the marks in each part will be allocated to the style and design of your solution.

Please make sure you follow the instructions closely and complete those methods as indicated. Heavy penalty will be imposed on unauthorized modification no matter whether difficulty level would be increased or decreased thereof.

=== End of paper ===