Seoul National University

Data Structure

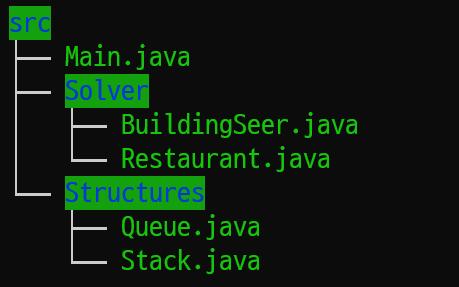
Spring 2023, Kang

Programming Assignment 1: Lists, Stacks, and Queues (Chapter 4)

Due: April 5, 23:59, submit at eTL

**Reminders**

* The points of this homework add up to 100.
* Like all homework, this has to be done individually.
* Lead T.A.: Jongjin Kim (j2kim99@snu.ac.kr)
* Write a program in Java.
* Do not use Java Collection Framework and third-party implementation from the Internet.

1. **How to submit the programming assignment**
2. Fill in the skeleton code with your own answer.
3. Compress your codes as ‘src.zip’ file. The structure of your ‘src.zip’ file should look like follows:
4. Your code will be compiled and executed with following commands in Linux environment. Make sure that your code runs well with following commands.  
     
   **javac $(find src/\* | grep .java)  
   java ./src/Main ${input\_filepath} ${output\_filepath}**
5. Submit the zip file to the eTL (<http://etl.snu.ac.kr/>).
6. **How to grade your programming assignment**
7. We made a grading machine to automatically grade your programming assignment. The machine will run your program and compare the answers and outputs that your program generates for given inputs. If your program cannot generate correct answers for an input file, it will not give you the point corresponding to the input. Our machine will consider the following scenarios:

(***Accept***) When your program generates exact outputs for an input file, the machine will give you the point of the input.

(***Wrong* *Answer***) When your program runs normally but generates incorrect outputs for an input file, including typos, the machine will not give you the point of the input.

(***Run Error***) When your program does not run or is terminated suddenly for some reason, the machine will not give you the point of an input file because it cannot generate any outputs.

(***Time Limit***) When your program runs over a predefined execution time for an input file, our machine will stop your program, and it will not give you the point of the input. The time limit of the execution is ***5 seconds***.

1. We will generate 10 input files, and assign 10 points for each input file. For example, if your program gets 9 accepts, and 1 wrong answer by the machine, the total point will be 90 points. Hence, before submitting your programming assignment, please be sure that your program makes correct answers in a reasonable time for any input case.
2. **Problem**

**Stack** and **Queue** are simple but very powerful data structures when you make applications. You will implement these two data structures and solve problems using the implemented classes.

1. **Stack**

A stack is an abstract data type that serves as a collection of elements, with the following main principal operations.

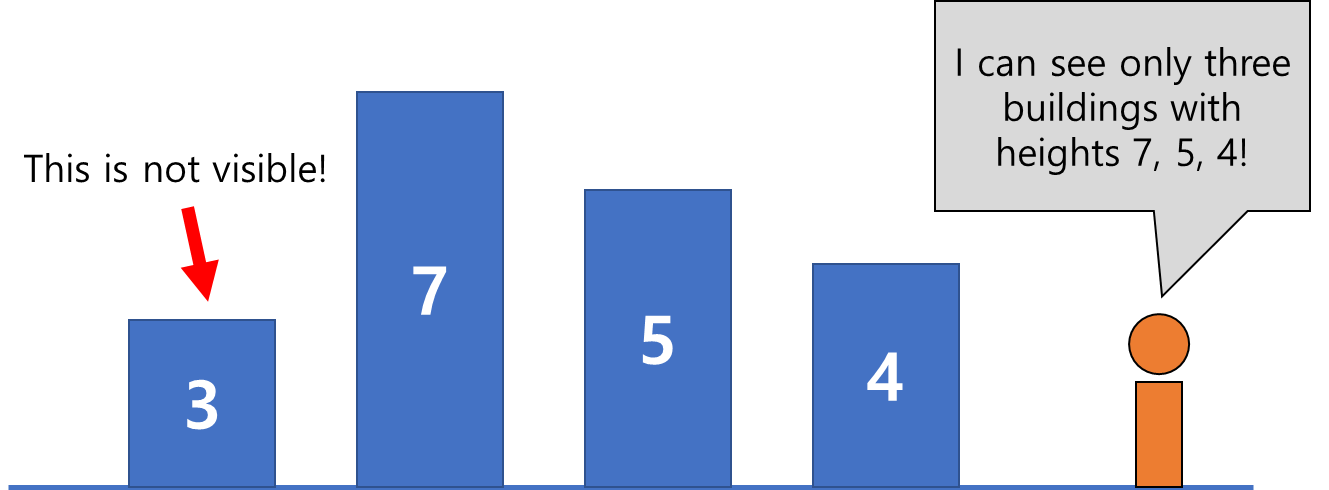
* *Push*: Adds an element to the collection.
* *Pop*: Removes the most recently added element.
* *Clear*: Clear the stack.
* *Length*: Return the length of the queue.
* *isEmpty*: Determine if the stack is empty.

1. **Queue**

A queue is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities from the other end of the sequence,with the following main principal operations.

* *Push*: Adds an element to the collection.
* *Pop*: Removes the oldest element in the collection.
* *Clear*: Clear the queue.
* *Length*: Return the length of the queue.
* *isEmpty*: Determine if the queue is empty.

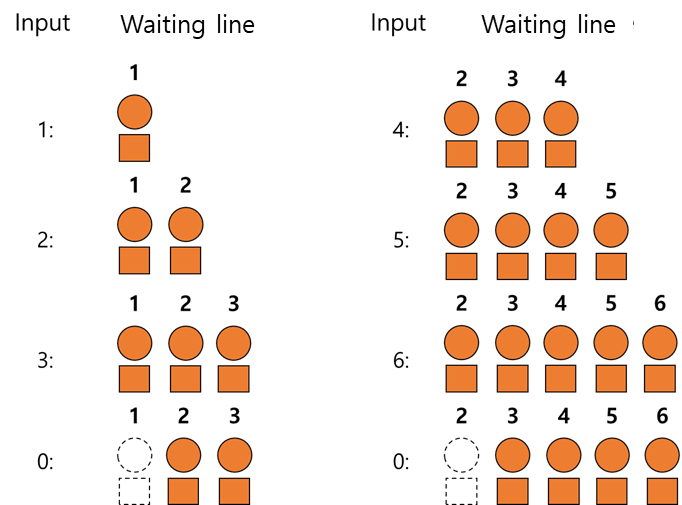
1. **BuildingSeer**

In this problem, for a given sequence of building heights, you need to find how many buildings are visible from the end of the sequence. Note that smaller buildings are blocked by taller buildings. For example, if the heights of buildings are [3, 7, 5, 4], the first building is blocked by taller buildings in front of it so only second, third, fourth buildings are visible at the end of the sequence.

BuildingSeer class is designed to solve the above problem. You have to implement following operations for the class **using stack that you implemented**:

* *newBuilding*: Adds a new building with a given heights at the end of the sequence.
* *visibleBuildingNum*: Return the number of visible buildings from the end of the sequence.

1. **Restaurant**

****In this problem, you will implement a code to manage a waiting line of a restaurant. You will continuously receive an notification that a new student has arrived or a meal is served. You have to report the last student who is served a meal and the maximum length of the waiting line during the service. For example, assume that the given sequence is [1, 2, 3, 0, 4, 5, 6, 0]. Positive numbers are arrived students’ id and ‘0’ means that a meal is served. In this case, the last student who was served the meal is student ‘2’, and the maximum length of the line was 5. Thus, the answer is “2 5”.

Restaurant class is designed to solve the above problem. You have to implement following operations for the class **using queue that you implemented**:

* *newStudent*: Adds a new arrived student behind the waiting line.
* *serve*: Serve a meal to the first student in the waiting line.
* *answer*: Return the string “n m” where n is the last student who was served the meal and m is the maximum length of the line during the service. (Two numbers should be separated with a single space ‘ ‘.)

# Important Notes

1. You **should not modify the Main.java** file in the skeleton. We will overwrite the Main.java file with our file during evaluation.
2. You **should not import any other packages or modules** other than already imported ones in the skeleton code. You will receive a severe penalty if you import other packages such as java.util.LinkedList or java.util.Stack.
3. The number of maximum entries of the stack or the queue will be given as an argument of the constructors of each class. For example, if we initialize a stack with a statement “stack A = new stack(10);”, A should be able to handle at least ten entries.
4. The maximum number of buildings and the maximum number of students will also be given as an argument of the constructors of respective solvers. For example, if we initialize a BuildingSeer class with a statement “BuildingSeer B = new BuiildingSeer(10);”, B should be able to handle at least ten buildings.
5. **You do not have to take account for detailed format of inputs and outputs in this assignment.** The Main class in skeleton would handle the i/o process of this assignment instead of you.
6. **I/O Examples**

Inputs and outputs of the Main.java file will be given and generated as a form of text files. Below examples are examples of sample inputs and outputs. You may test your code with the given sample files in the skeleton or your own inputs with following command after compile:

**java ./src/Main ${input\_filepath} ${output\_filepath}**

|  |  |
| --- | --- |
| **Sample Input** | **Sample Output** |
| stack push 10  stack push 20  stack push 30  stack pop  stack isempty  stack push 40  stack length  stack pop  stack pop  stack pop  stack isempty  stack push 50  stack clear  stack isempty  stack length | stack pushed: 10  stack pushed: 20  stack pushed: 30  stack pop: 30  stack is not empty  stack pushed: 40  stack length: 3  stack pop: 40  stack pop: 20  stack pop: 10  stack is empty  stack pushed: 50  stack cleared  stack is empty  stack length: 0 |

|  |  |
| --- | --- |
| **Sample Input** | **Sample Output** |
| queue push 20  queue push 30  queue pop  queue isempty  queue push 40  queue length  queue pop  queue pop  queue isempty  queue length  queue push 50  queue clear  queue isempty  queue length | queue pushed: 20  queue pushed: 30  queue pop: 20  queue is not empty  queue pushed: 40  queue length: 2  queue pop: 30  queue pop: 40  queue is empty  queue length: 0  queue pushed: 50  queue cleared  queue is empty  queue length: 0 |

|  |  |
| --- | --- |
| **Sample Input** | **Sample Output** |
| building 10 9 8 7 6 5 4 3 2 1  building 1 2 3 4 5 6 7 8 9 10  building 10 6 5 9 7 8 1 2 3 4 | visible buildings: 10  visible buildings: 1  visible buildings: 4 |

|  |  |
| --- | --- |
| **Sample Input** | **Sample Output** |
| restaurant 3 2 1 0 0 0  restaurant 1 2 3 0 0 4 5 6 7 0 0 0 0 0  restaurant 1 2 3 0 4 5 6 0 7 8 9 | length, stu\_id: 1 3  length, stu\_id: 7 5  length, stu\_id: 2 7 |