**Assembly Line**

|  |  |  |
| --- | --- | --- |
| 3 |  |  |
| 5 |  | 8 |
| 6 |  | 2 |
| 10 |  | 1 |
| "regular" |  | "inverted" |

This program is a simulation of an *assembly line* for assembling *pyramids* made of *stacks* of *disks*. All pyramids are stacks of disks arranged in order by size. In a "regular" pyramid, the sizes (the radii) of the disks increase from top to bottom, but in an "inverted" pyramid the sizes (the radii) decrease, as shown here:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| 3 |  |  |
| 5 |  | 1 |
| 6 |  | 2 |
| 10 | 7 | 8 |

Disks arrive on an assembly line--a queue--and are assembled into pyramids by a robot. The robot can temporarily build and hold an inverted pyramid of disks. When the robot cannot add another disk while maintaining the size ordering, the robot "flips over" the pyramid--the stack--it is holding and places it into the output assembly line, also a queue. Thus, the output queue always contains regular pyramids of disks, as illustrated. In our program, the input queue of [3 5 6 10 7 1 2 8] produces an output queue which is displayed as [[10, 6, 5, 3], [7], [8, 2, 1]]

The simulation is implemented in the class Assembly Line:

public class AssemblyLine

{

private Queue<Disk> assemblyLineIn;

private Queue<Pyramid> assemblyLineOut;

private Pyramid robotArm;

/\*\*

\* initializes this object so the assemblyLineIn contains   
\* nDisks Disk objects with randomly generated radii;

\* assemblyLineOut is initialized to an empty Queue;

\* robotArm is initialized to an empty Pyramid.

\*\*/

public AssemblyLine( int nDisks, int maxRadius )

{ /\* to be implemented in Part A \*/ }

/\*\*

\* "flips over" the pyramid in the robotArm and adds it to the

\* assemblyLineOut queue.

\* Precondition: robotArm is not empty and holds an inverted

\* pyramid of disks

\*\*/

private void unloadRobot()

{ /\* to be implemented in Part B \*/}

/\*\*

\* processes all disks from assemblyLineIn; a disk is processed

\* as follows: if robotArm is not empty and the next disk does

\* not fit on top of robotArm (which must be an inverted

\* pyramid) then robotArm is unloaded first; the disk from

\* assemblyLineIn is added to robotArm; when all the disks

\* have been retrieved from assemblyLineIn, robotArm is unloaded.

\* Precondition: robotArm is empty;

\* assemblyLineOut is empty

\*\*/

public void process()

{ /\* to be implemented in part C \*/ }

}

(A) Write a constructor for the AssemblyLine class.

public AssemblyLine( int nDisks, int maxRadius )

{

(B) Write the unloadRobot method of the AssemblyLine class.

private void unloadRobot()

{

(C) Write the process method of the AssemblyLine class.

public void process()

{

In order to turn this into a lab, you will also have to write the Disk and Pyramid classes. (The driver is written for you.) Disk is standard and straightforward. Pyramid is sly. Use these headers:

class Disk implements Comparable<Disk>

class Pyramid extends Stack<Disk>