Project 2: Stocks Go Up!

DUE: March 3rd , 2023 at 11:59pm

Extra Credit Available for Early Submissions!

Basic Procedures

You must:

- Fill out a readme.txt file with your information (goes in your user folder, an example readme.txt file is provided).
- Have a style (indentation, good variable names, etc.) and pass the provided style checker (See P0).
- Comment your code well in JavaDoc style and pass the provided JavaDoc checker (See P0).
- Have code that compiles with the command: **`javac *.java`** in your user directory without errors or warnings.
- For methods that come with a big-O requirement (check the provided template Java files for details), make sure your implementation meet the requirement.
- Implement all required methods to match the expected behavior as described in the given template files.
- Have code that runs with the commands: **`java StockMonitor InputFile [-d]`**

You may:

- Add additional methods and class/instance variables (unless specified otherwise by the template files), however they **must be private**.

You may NOT:

- Make your program part of a package.
- Add additional public methods or public class/instance variables, remember that local variables are not the same as class/instance variables!
- Use any built in Java Collections Framework classes anywhere in your program (e.g. no ArrayList, LinkedList, HashSet, etc.).
- Declare/use any arrays anywhere in your program (except the provided **`buckets`** field in **`ThreeTenHashMap`**).
  - You may not call toArray() methods to bypass this requirement.
- Alter any method signatures defined in this document of the template code. Note: "throws" is part of the method signature in Java, don't add/remove these.
- Alter provided classes that are complete (e.g. **`Node`**).
- Add any additional import statements (or use the "fully qualified name" to get around adding import statements).
- Add any additional libraries/packages which require downloading from the internet.

Setup

- Download the **`p2.zip`** and unzip it. This will create a folder **`section-yourGMUUserName-p2`**;
- Rename the folder replacing **`section`** with the **`004`** or **`006`** based on the lecture section you are in;
- Rename the folder replacing **`yourGMUUserName`** with the first part of your GMU email address;
- After renaming, your folder should be named something like: **`000-yzhong-p2.`**
- Complete the **`readme.txt`** file (an example file is included: **`exampleReadmeFile.txt`**).

Submission Instructions

- Make a backup copy of your user folder!
- Remove all test files, jar files, class files, etc. You should just submit your java files and your readme.txt
- Zip your user folder (not just the files) and name the zip **`section-username-p2.zip`** (no other type of archive) following the same rules for **`section`** and **`username`** as described above. For example:
  - **`000-yzhong-p2.zip --> 000-yzhong-p2 -->`** **`JavaFile1.java`**, **`JavaFile2.java`**, **`JavaFile3.java ...`**
- Submit to blackboard.

Grading Rubric

Due to the complexity of this assignment, an accompanying grading rubric pdf has been included with this assignment. Please refer to this document for a complete explanation of the grading, including extra credit for early submissions.

Overview

Investors always want stock prices to go up! For this project, we will build a stock price monitor and analyze the ups and downs of stock prices. In particular, we will inspect a sequence of daily prices p_0, p_1, … p_n, assuming p_i is the stock price we have for the i^{th} day. For each daily price p_i, we will determine for how many consecutive days before that day (and including that day) we have a price smaller than or equal to p_i. This is considered as the ***span*** of that p_i. The table below gives an example:

Example sequence of 7 days of stock price:

Days	0	1	2	3	4	5	6
Prices	39	32	41	25	25	22	40
Spans	1	1	3	1	2	1	4

Example explanations:

- Price 39 for Day 0 has a span of 1 as there is no price to its left (i.e. no previous days exit).
- Price 32 for Day 1 has a span of 1 as there is no price smaller than it to its left.
- Price 41 for Day 2 has a span of 3 as there are two smaller prices to its left (i.e. 39, 32).
- Price 25 for Day 3 has a span of 1 as immediate left neighbor is greater than 25.
- Price 25 for Day 4 has a span of 2 as there is one price to its left that is no greater than it (25==25).
- …
- You should get the idea now!

To determine the span of each day efficiently, we will use a stack to store the prices in monotonically decreasing order. For each price in stack, we will also record on which day we see that price. Hence each item in stack would be a pair of <day, price>. **Algorithm** we follow for span calculation and stack maintenance is as below:

1. For Day 0: span = 1, initialize the stack with this first day information.
2. For all following days, suppose we are processing a pair of <current_day, current_price>:
    a. Keep popping items from the stack top until either the stack is empty or the stack top has a price greater than current_price;
    b. If the stack is empty, that implies all previous days have a price no greater than the latest one, so we know span = current_day+1;
    c. Otherwise, stack top represents the latest day with a price greater than current_price. Therefore, we can calculate span = current_day – latest_day;
    d. Push the pair <current_day, current_price> onto the stack.

Span calculation and stack maintenance step by step example from the sequence above:

Price Queue	Current Day	Current Price	Span	Stack (bottom to top)	Notes
39 32 41 25 25 22 40	0	39	1	<0, 39>	Nothing prior to day 0; push <current day, current price> onto stack
39 **32** 41 25 25 22 40	1	32	1	<0, 39> <1,32>	Rule 2.c applied: day 0 (stack top) is the latest day with a price greater than current price, hence span=1-0 = 1; push <1,32>
39 32 **41** 25 25 22 40	2	41	3	<2,41>	All items in stack are popped out (all smaller than 41), then Rule 2.b applied: span = 2+1 = 3; push <2,41>
39 32 41 **25** 25 22 40	3	25	1	<2,41> <3, 25>	Rule 2.c applied: day 2 is the latest day with a price greater than current price, hence span=3-2 = 1; push <3,25>

39 32 41 25 **25** 22 40	4	25	2	<2,41> <4, 25>	<3,25> popped out (since it equals to current price 25); then Rule 2.c applied: day 2 is the latest day with a price greater than current price, hence span=4-2 = 2; push <4,25>
39 32 41 25 25 **22** 40	5	22	1	<2,41> <4, 25> <5,22>	Stack top (25) is greater than current price (22), nothing popped out; then Rule 2.c applied: day 4 is the latest day with a price greater than current price, hence span=5-4 = 1; push <5,22>
39 32 41 25 25 22 **40**	6	40	4	<2,41> <6, 40>	Two items smaller than 40 popped out (25 and 22); then Rule 2.c applied: day 2 is the latest day with a price greater than current price, hence span=6-2 = 4; push <6,40>

One other analysis we will perform is to use a hash map to help us keep track of the max span we have observed for each price value. A final report will be provided when we complete the processing of all prices. See example below and sample runs in Appendix for details.

Max span for each price in the example sequence:

Price from sequence	39	32	41	25	22	40
Max Span of each price	1	1	3	**2**	1	4

- Note: 25 occurred two times in the sequence and we only remember the higher span.

Implementation/Classes

This project will be built using a number of classes representing a doubly linked lists, a stack, a hashmap, and the stock price analyzer. The three data structure classes are all generic. Here we provide a description of these classes. Template files are provided for each class in the project package and these contain further comments and additional details. You must follow the instructions included in those files.

- **Node (Node.java)**: The node used in linked list class **`ThreeTenDLList`**. This class is provided to you and you should NOT change the file.
- **ThreeTenDLList (ThreeTenDLList.java)**: The implementation of a generic doubly linked lists. We will use this to construct other classes of this project.
- **ThreeTenStack (ThreeTenStack.java)**: The implementation of a stack built upon **`ThreeTenDLList`**.
- **ThreeTenHashMap (ThreeTenHashMap.java)**: The implementation of a hash map using separate chaining for conflict resolution. It is also built upon **`ThreeTenDLList`**.
- **StockMonitor (StockMonitor.java)**: The implementation of a stock price monitor and analyzer. It reads in a sequence of daily stock prices from an input file, then calculate the span of each day using the stack-based algorithm as described above, and keep track of the max span value for each price in the sequence.
- There are a number of methods already implemented and provided to you in some of the java classes above. Do not change the provided methods but you will need to add JavaDocs for them.

Requirements

An overview of the requirements is listed below, please see the grading rubric and template files for more details.

- **Implementing the classes** - You will need to implement required classes and fill the provided template files.
- **JavaDocs** - You are required to write JavaDoc comments for all the required classes and methods.
- **Big-O** - Template files provided to you contains instructions on the REQUIRED Big-O runtime for many methods. Your implementation of those methods should NOT have a higher Big-O.

How To Handle a Multi-Week Project

While this project is given to you to work on for about two weeks, you are unlikely to be able to complete everything in one weekend. We recommend the following schedule:

- Step 1 (**`ThreeTenDLList`**): Before the first weekend (by 02/17)
  - Implement and test methods of **`ThreeTenDLList`**.
- Step 2 (**`ThreeTenStack,StockMonitor`**): First weekend (02/18-02/19)
  - Implement and test methods of **`ThreeTenStack.`**
  - Implement and test methods of **`StockMonitor`**, everything not related to recording/reporting max span should work.
- Step 3 (**`ThreeTenHashMap`**): Before second weekend (02/20-02/24)
  - Implement and test methods of **`ThreeTenHashMap`**.
- Step 4 (**`StockMonitor`**): Second weekend (02/25-02/26)
  - Complete and test all methods of **`StockMonitor`**.
- Step 5 (**`Wrapping-up`**): Last week (02/27-03/03)
  - Additional testing, debugging, get additional help.
  - ☺ Also, notice that if you get it done early in the week, you can get extra credit! Check our grading rubric PDF for details.

Testing

The main methods provided in the template files contain useful example code to test your project as you work. You can use command like "**`java ThreeTenDLList`**" or "**`java ThreeTenStack`**" to run the testing defined in **`main()`**.

- ***Note:*** passing all yays does NOT guarantee 100% on the project! Those are only examples for you to start testing and they only cover limited cases. Make sure you add many more tests in your development. You could edit **`main()`** to perform additional testing.
- JUnit test cases will not be provided for this project, but feel free to create JUnit tests for yourself. A part of your grade *will* be based on automatic grading using test cases made from the specifications provided.

We have provided the framework in **`StockMonitor`** so that you can run the span measurement simulation with an input file to specify the sequence of prices. The simulation will have two modes: normal and debug. In normal mode, the simulation will complete processing all prices in the sequence and display only the final report. In debug mode, the simulation will perform one "step" at a time. The **`StockMonitor`** is run with in the following ways:

```
java StockMonitor [path-to-input] [-d]
```

For example, when I run the **`StockMonitor`** *without* debug mode on one of the provided sample programs (in1.txt in this case), I type and see the following:

```
java StockMonitor ../input/in1.txt

Prices:     39 32 41 25 25 22 40
Spans:      1 1 3 1 2 1 4
Max spans:  39:1 32:1 41:3 25:2 22:1 40:4
```

And when I run the StockMonitor *with* debug mode, I type and see the following:

```
java StockMonitor ../input/in1.txt -d

Prices:     39 32 41 25 25 22 40

######### Step 0 ###############

----------Step Output----------
Day = 0, Price = 39, Span = 1
-------------------------------
--Record Stack (bottom to top)--
```

```
<0,39>
-------------Spans-------------
1
----------Prices Remaining----
32 41 25 25 22 40

Press Enter to Continue
```

When I press enter I see:

```
######### Step 1 ###############

----------Step Output----------
Day = 1, Price = 32, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<0,39> <1,32>
-------------Spans-------------
1 1
----------Prices Remaining----
41 25 25 22 40

Press Enter to Continue
```

The debug mode and printing have already been done for you, but your job will be to build the supporting data structures and processing the list of prices. You can do this in stages:

- **Step 1**: Read the input from a file into a linked list. You will practice a bit of file I/O and complete a method called **`fileToPriceList()`**. Using the Java Scanner class, you'll need to turn an input text file into a linked list of integers that we will check one by one later to determine the span for each.
    - You can assume every input file includes one or more positive integers, separated by whitespaces.
- Once you have completed this stage, you should be able to run your program as follows and see the input displayed. The span checking is not working yet, but this is a good start!

```
java StockMonitor input/in1.txt
Prices:  39 32 41 25 25 22 40
```

- **Step 2**: Use a stack to support the StockMonitor's computations and store spans in a linked list. You will need to fill in the implementation of a method **`stepProcess()`** which is called by the provided **`runProgram()`** to check the prices one step at a time and perform needed maintenance on supporting data structures.
- **Step 3**: Use a hashmap to support the StockMonitor in keeping track of the max span for each price processed. You will need to further enhance **`stepProcess()`** and complete max span related methods to generate the complete report at the end of processing.

To help with testing, we provide a number of input files that you can use with **`StockMonitor`** under the folder **input/**. We also include multiple sample-runs in the Appendix of this document to show you the expected behavior/printing.

- ***Note:*** as always, matching all provided sample runs does NOT guarantee 100% on the project!
- You should test with more scenarios and files of your own.

Appendix: Sample Runs

Sample Run 1

Initialize with in1.txt; no debug mode.

```
java StockMonitor input/in1.txt
Prices:     39 32 41 25 25 22 40
Spans:      1 1 3 1 2 1 4
Max spans:  39:1 32:1 41:3 25:2 22:1 40:4
```

- First row: sequence of prices;
- Second row: span of each price, follow the original order of prices;
- Third row: the max span of each price in the form of price:maxSpan, follow the same order but skip repeated prices.

Sample Run 2

Initialize with in1.txt; debug mode.

Print the sequence of prices.

Step by step simulation.

Report day, price, and the span measure for that day.

<day, price> add to top of the initial empty stack.

Span added to the end of a list.

Report prices yet to be processed.

```
java StockMonitor input/in1.txt -d
Prices:     39 32 41 25 25 22 40

######### Step 0 ###############

----------Step Output----------
Day = 0, Price = 39, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<0,39>
-------------Spans-------------
1
----------Prices Remaining----
32 41 25 25 22 40

Press Enter to Continue


######### Step 1 ###############

----------Step Output----------
Day = 1, Price = 32, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<0,39> <1,32>
-------------Spans-------------
1 1
----------Prices Remaining----
41 25 25 22 40

Press Enter to Continue
```

Next step: this is the same sequence as our example in description before.

```
######### Step 2 ###############

----------Step Output----------
Day = 2, Price = 41, Span = 3
-------------------------------
--Record Stack (bottom to top)--
<2,41>
-------------Spans-------------
1 1 3
----------Prices Remaining----
25 25 22 40

Press Enter to Continue


######### Step 3 ###############

----------Step Output----------
Day = 3, Price = 25, Span = 1
-------------------------------
--Record Stack (bottom to top)--
<2,41> <3,25>
-------------Spans-------------
1 1 3 1
----------Prices Remaining----
25 22 40

Press Enter to Continue


######### Step 4 ###############

----------Step Output----------
Day = 4, Price = 25, Span = 2
-------------------------------
--Record Stack (bottom to top)--
<2,41> <4,25>
-------------Spans-------------
1 1 3 1 2
----------Prices Remaining----
22 40

Press Enter to Continue


######### Step 5 ###############

----------Step Output----------
Day = 5, Price = 22, Span = 1
-------------------------------
--Record Stack (bottom to top)--
<2,41> <4,25> <5,22>
-------------Spans-------------
1 1 3 1 2 1
----------Prices Remaining----
40

Press Enter to Continue


######### Step 6 ###############

----------Step Output----------
Day = 6, Price = 40, Span = 4
-------------------------------
--Record Stack (bottom to top)--
<2,41> <6,40>
-------------Spans-------------
1 1 3 1 2 1 4
----------Prices Remaining----


Press Enter to Continue

Prices:     39 32 41 25 25 22 40
Spans:      1 1 3 1 2 1 4
Max spans:  39:1 32:1 41:3 25:2 22:1 40:4
```

Final report of all spans and max span of each price. This would be the same for both debug or non-debug mode.

Sample Run 3

Initialize with in2.txt; no debug mode.

```
java StockMonitor input/in2.txt
Prices:     3 16 28 9 7 19 19 5 9 3 9 22 17 2 30 12 10 29 5 1
Spans:      1 2 3 1 1 3 4 1 2 1 4 9 1 1 15 1 1 3 1 1
Max spans:  3:1 16:2 28:3 9:4 7:1 19:4 5:1 22:9 17:1 2:1 30:15 12:1 10:1 29:3 1:1
```

Sample Run 4

Initialize with in2.txt; debug mode.

```
java StockMonitor input/in2.txt -d
Prices:     3 16 28 9 7 19 19 5 9 3 9 22 17 2 30 12 10 29 5 1

######### Step 0 ###############

----------Step Output----------
Day = 0, Price = 3, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<0,3>
-------------Spans-------------
1
----------Prices Remaining----
16 28 9 7 19 19 5 9 3 9 22 17 2 30 12 10 29 5 1

Press Enter to Continue


######### Step 1 ###############

----------Step Output----------
Day = 1, Price = 16, Span = 2
--------------------------------
--Record Stack (bottom to top)--
<1,16>
-------------Spans-------------
1 2
----------Prices Remaining----
28 9 7 19 19 5 9 3 9 22 17 2 30 12 10 29 5 1

Press Enter to Continue


######### Step 2 ###############

----------Step Output----------
Day = 2, Price = 28, Span = 3
--------------------------------
--Record Stack (bottom to top)--
<2,28>
-------------Spans-------------
1 2 3
----------Prices Remaining----
9 7 19 19 5 9 3 9 22 17 2 30 12 10 29 5 1
```

```
Press Enter to Continue


######### Step 3 ###############

----------Step Output----------
Day = 3, Price = 9, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<2,28> <3,9>
-------------Spans-------------
1 2 3 1
----------Prices Remaining----
7 19 19 5 9 3 9 22 17 2 30 12 10 29 5 1

Press Enter to Continue


######### Step 4 ###############

----------Step Output----------
Day = 4, Price = 7, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<2,28> <3,9> <4,7>
-------------Spans-------------
1 2 3 1 1
----------Prices Remaining----
19 19 5 9 3 9 22 17 2 30 12 10 29 5 1

Press Enter to Continue


######### Step 5 ###############

----------Step Output----------
Day = 5, Price = 19, Span = 3
--------------------------------
--Record Stack (bottom to top)--
<2,28> <5,19>
-------------Spans-------------
1 2 3 1 1 3
----------Prices Remaining----
19 5 9 3 9 22 17 2 30 12 10 29 5 1

Press Enter to Continue


######### Step 6 ###############

----------Step Output----------
Day = 6, Price = 19, Span = 4
--------------------------------
--Record Stack (bottom to top)--
<2,28> <6,19>
-------------Spans-------------
1 2 3 1 1 3 4
----------Prices Remaining----
5 9 3 9 22 17 2 30 12 10 29 5 1

Press Enter to Continue


######### Step 7 ###############

----------Step Output----------
Day = 7, Price = 5, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<2,28> <6,19> <7,5>
-------------Spans-------------
1 2 3 1 1 3 4 1
----------Prices Remaining----
9 3 9 22 17 2 30 12 10 29 5 1

Press Enter to Continue


######### Step 8 ###############

----------Step Output----------
Day = 8, Price = 9, Span = 2
--------------------------------
--Record Stack (bottom to top)--
<2,28> <6,19> <8,9>
-------------Spans-------------
1 2 3 1 1 3 4 1 2
----------Prices Remaining----
3 9 22 17 2 30 12 10 29 5 1

Press Enter to Continue


######### Step 9 ###############

----------Step Output----------
Day = 9, Price = 3, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<2,28> <6,19> <8,9> <9,3>
-------------Spans-------------
1 2 3 1 1 3 4 1 2 1
----------Prices Remaining----
9 22 17 2 30 12 10 29 5 1
```

```
Press Enter to Continue


######### Step 10 ###############

----------Step Output----------
Day = 10, Price = 9, Span = 4
--------------------------------
--Record Stack (bottom to top)--
<2,28> <6,19> <10,9>
-------------Spans-------------
1 2 3 1 1 3 4 1 2 1 4
----------Prices Remaining----
22 17 2 30 12 10 29 5 1

Press Enter to Continue


######### Step 11 ###############

----------Step Output----------
Day = 11, Price = 22, Span = 9
--------------------------------
--Record Stack (bottom to top)--
<2,28> <11,22>
-------------Spans-------------
1 2 3 1 1 3 4 1 2 1 4 9
----------Prices Remaining----
17 2 30 12 10 29 5 1

Press Enter to Continue


######### Step 12 ###############

----------Step Output----------
Day = 12, Price = 17, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<2,28> <11,22> <12,17>
-------------Spans-------------
1 2 3 1 1 3 4 1 2 1 4 9 1
----------Prices Remaining----
2 30 12 10 29 5 1

Press Enter to Continue


######### Step 13 ###############

----------Step Output----------
Day = 13, Price = 2, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<2,28> <11,22> <12,17> <13,2>
-------------Spans-------------
1 2 3 1 1 3 4 1 2 1 4 9 1 1
----------Prices Remaining----
30 12 10 29 5 1

Press Enter to Continue


######### Step 14 ###############

----------Step Output----------
Day = 14, Price = 30, Span = 15
--------------------------------
--Record Stack (bottom to top)--
<14,30>
-------------Spans-------------
1 2 3 1 1 3 4 1 2 1 4 9 1 1 15
----------Prices Remaining----
12 10 29 5 1

Press Enter to Continue


######### Step 15 ###############

----------Step Output----------
Day = 15, Price = 12, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<14,30> <15,12>
-------------Spans-------------
1 2 3 1 1 3 4 1 2 1 4 9 1 1 15 1
----------Prices Remaining----
10 29 5 1

Press Enter to Continue


######### Step 16 ###############

----------Step Output----------
Day = 16, Price = 10, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<14,30> <15,12> <16,10>
-------------Spans-------------
1 2 3 1 1 3 4 1 2 1 4 9 1 1 15 1 1
----------Prices Remaining----
29 5 1
```

```
Press Enter to Continue


######### Step 17 ###############

----------Step Output----------
Day = 17, Price = 29, Span = 3
--------------------------------
--Record Stack (bottom to top)--
<14,30> <17,29>
-------------Spans-------------
1 2 3 1 1 3 4 1 2 1 4 9 1 1 15 1 1 3
----------Prices Remaining----
5 1

Press Enter to Continue


######### Step 18 ###############

----------Step Output----------
Day = 18, Price = 5, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<14,30> <17,29> <18,5>
-------------Spans-------------
1 2 3 1 1 3 4 1 2 1 4 9 1 1 15 1 1 3 1
----------Prices Remaining----
1

Press Enter to Continue


######### Step 19 ###############

----------Step Output----------
Day = 19, Price = 1, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<14,30> <17,29> <18,5> <19,1>
-------------Spans-------------
1 2 3 1 1 3 4 1 2 1 4 9 1 1 15 1 1 3 1 1
----------Prices Remaining----


Press Enter to Continue

Prices:     3 16 28 9 7 19 19 5 9 3 9 22
17 2 30 12 10 29 5 1
Spans:      1 2 3 1 1 3 4 1 2 1 4 9 1 1
15 1 1 3 1 1
Max spans:  3:1 16:2 28:3 9:4 7:1 19:4
5:1 22:9 17:1 2:1 30:15 12:1 10:1 29:3
1:1java
```

Sample Run 5

Initialize with in3.txt; no debug mode.

```
java StockMonitor input/in3.txt
Prices:     310 310 310 310 310 310 310 310 310 310 310 310
Spans:      1 2 3 4 5 6 7 8 9 10 11 12
Max spans:  310:12
```

All prices are the same.

Sample Run 6

Initialize with in3.txt; debug mode.

```
java StockMonitor input/in3.txt -d
Prices:     310 310 310 310 310 310 310 310 310 310 310 310

######### Step 0 ###############

----------Step Output----------
Day = 0, Price = 310, Span = 1
--------------------------------
--Record Stack (bottom to top)--
<0,310>
-------------Spans-------------
1
----------Prices Remaining----
310 310 310 310 310 310 310 310 310 310 310

Press Enter to Continue


######### Step 1 ###############

----------Step Output----------
Day = 1, Price = 310, Span = 2
--------------------------------
--Record Stack (bottom to top)--
<1,310>
-------------Spans-------------
1 2
----------Prices Remaining----
310 310 310 310 310 310 310 310 310 310

Press Enter to Continue


######### Step 2 ###############

----------Step Output----------
Day = 2, Price = 310, Span = 3
--------------------------------
--Record Stack (bottom to top)--
<2,310>
-------------Spans-------------
1 2 3
```

```
----------Prices Remaining----
310 310 310 310 310 310 310 310 310

Press Enter to Continue


######### Step 3 ###############

----------Step Output----------
Day = 3, Price = 310, Span = 4
--------------------------------
--Record Stack (bottom to top)--
<3,310>
-------------Spans-------------
1 2 3 4
----------Prices Remaining----
310 310 310 310 310 310 310 310

Press Enter to Continue


######### Step 4 ###############

----------Step Output----------
Day = 4, Price = 310, Span = 5
--------------------------------
--Record Stack (bottom to top)--
<4,310>
-------------Spans-------------
1 2 3 4 5
----------Prices Remaining----
310 310 310 310 310 310 310

Press Enter to Continue


######### Step 5 ###############

----------Step Output----------
Day = 5, Price = 310, Span = 6
--------------------------------
--Record Stack (bottom to top)--
<5,310>
-------------Spans-------------
1 2 3 4 5 6
----------Prices Remaining----
310 310 310 310 310 310

Press Enter to Continue


######### Step 6 ###############

----------Step Output----------
Day = 6, Price = 310, Span = 7
--------------------------------
--Record Stack (bottom to top)--
<6,310>
-------------Spans-------------
1 2 3 4 5 6 7
----------Prices Remaining----
310 310 310 310 310

Press Enter to Continue


######### Step 7 ###############

----------Step Output----------
Day = 7, Price = 310, Span = 8
--------------------------------
--Record Stack (bottom to top)--
<7,310>
-------------Spans-------------
1 2 3 4 5 6 7 8
----------Prices Remaining----
310 310 310 310

Press Enter to Continue


######### Step 8 ###############

----------Step Output----------
Day = 8, Price = 310, Span = 9
--------------------------------
--Record Stack (bottom to top)--
<8,310>
-------------Spans-------------
1 2 3 4 5 6 7 8 9
----------Prices Remaining----
310 310 310

Press Enter to Continue


######### Step 9 ###############

----------Step Output----------
Day = 9, Price = 310, Span = 10
--------------------------------
--Record Stack (bottom to top)--
<9,310>
-------------Spans-------------
1 2 3 4 5 6 7 8 9 10
----------Prices Remaining----
```

```
310 310

Press Enter to Continue


######### Step 10 ###############

----------Step Output----------
Day = 10, Price = 310, Span = 11
--------------------------------
--Record Stack (bottom to top)--
<10,310>
-------------Spans-------------
1 2 3 4 5 6 7 8 9 10 11
----------Prices Remaining----
310

Press Enter to Continue
```

```
######### Step 11 ###############

----------Step Output----------
Day = 11, Price = 310, Span = 12
--------------------------------
--Record Stack (bottom to top)--
<11,310>
-------------Spans-------------
1 2 3 4 5 6 7 8 9 10 11 12
----------Prices Remaining----


Press Enter to Continue

Prices:     310 310 310 310 310 310 310
310 310 310 310 310
Spans:      1 2 3 4 5 6 7 8 9 10 11 12
Max spans:  310:12
```

Sample Run 7

Initialize with in4.txt; no debug mode.

```
java StockMonitor input/in4.txt
Prices:     17 14 58 45 51 31 4 68 79 80 81 33 94 86 6 85 49 51 92 12 17 93 48 11 54 99
91 40 87 73 4 51 28 53 51 95 35 81 22 51 86 65 51 47 35 47 66 78 37 100 63 20 9 73 47 8
48 89 9 79 99 32 6 91 24 15 15 2 21 75 43 16 100 47 89 21 54 33 24 20 5 50 96 85 84 31 78
72 18 53 15 57 29 61 81 97 89 1 34 29
Spans:      1 1 3 1 2 1 1 8 9 10 11 1 13 1 1 2 1 2 6 1 2 9 1 1 3 26 1 1 2 1 1 2 1 4 1 10
1 2 1 2 5 1 1 1 1 3 6 7 1 50 1 1 1 4 1 1 3 8 1 2 11 1 1 3 1 1 2 1 4 6 1 1 73 1 2 1 2 1 1
1 1 5 10 1 1 1 2 1 1 2 1 4 1 6 10 23 1 1 2 1
Max spans:  17:2 14:1 58:3 45:1 51:2 31:1 4:1 68:8 79:9 80:10 81:11 33:1 94:13 86:5 6:1
85:2 49:1 92:6 12:1 93:9 48:3 11:1 54:3 99:26 91:3 40:1 87:2 73:4 28:1 53:4 95:10 35:1
22:1 65:1 47:3 66:6 78:7 37:1 100:73 63:1 20:1 9:1 8:1 89:8 32:1 24:1 15:2 2:1 21:4 75:6
43:1 16:1 5:1 50:5 96:10 84:1 72:1 18:1 57:4 29:1 61:6 97:23 1:1 34:2
```

Debug mode printings are too long to be included. The input in4.txt has 100 prices and is formatted to be 5 numbers per row if you'd like to manually verify the result.