Introduction

Arrays are *the* fundamental data structure of many programming languages including Java. A clear, complete and deep understanding of their strengths and weaknesses is a prerequisite to becoming an effective programmer. This lab is devoted to trying to develop that strength in you – it is not directly related to any of the lectures although one of the issues (noted below) will arise later.

In this lab we will work with some special objects called Young tableaux, which are naturally represented as doubly indexed arrays of integers. These are very important objects in advanced algebra and discrete mathematics, but that will not concern us – we're just using the concept as a convenient foundation for some exercises on array manipulation.

A *Young tableau* is shown below:

1	4	5	10	11
2	6	8		
3	9	12		
7				

The key features of a Young tableau are as follows:

- it consists of cells which are filled with integers, and arranged in left-justified rows,
- no row is longer than a preceding row,
- from left to right in any row, and down any column the integers are increasing,
- the set of integers used is $\{1, 2, \ldots, n\}$ where n is the number of cells.

In Java, an obvious way to represent a tableau is as a doubly indexed array of integers, i.e, `int[][]`. The tableau above might be represented as:

```
int[][] t = {{1, 4, 5, 10, 11}, {2, 6, 8}, {3, 9, 12}, {7}};
```

A basic problem is that doubly indexed arrays of integers need have none of the properties that define tableaux. One of the objects of this lab is to write code that will check when such an array really is a tableau.

Problem description

The main objective of this lab is to write a function `isTableau` which takes an `int[][]` as input and returns a `boolean` as output, where the result is `true` if the array represents a tableau, and `false` otherwise. Because the defining properties of a tableau separate naturally into various individual properties, this is a good opportunity to practice bottom up design, and some form of test driven development – building and testing your code a piece at a time.

The conditions that a tableau must satisfy are listed above. The first is represented automatically by the representation as a doubly indexed array of integers, so the main issue is to test the remaining three conditions, noting that the third of the four conditions splits into two parts.

Problems

There is a skeleton class `TableauChecker.java` provided. This includes a basic utility function for `String` representations of doubly indexed arrays of integers as if they were tableaux.

Complete the static functions to the skeleton code class which implement condition two and the first part of condition three:

`rowLengthsDecrease(int[][] t)` A method that returns `true` if no row is longer than a preceding row, otherwise `false`.

`rowValuesIncrease(int[][] t)` A method that returns `true` if from left to right in any row, the integers are increasing, otherwise `false`.

Implement each one separately, testing as you go. Your methods need not work sensibly if passed a `null` argument, but should work properly on the empty tableau, i.e.,

```
int[][] t = {};
```

which those peculiar mathematicians insist is a proper tableau (with $n = 0$).

Complete two more static functions to the skeleton code class which implement the second part of condition three, and condition four:

`columnValuesIncrease(int[][] t)` A method that returns `true` if from top to bottom in any column, the integers are increasing, otherwise `false`.

`isSetOf1toN(int[][] t)` A method that returns `true` if the set of integers used is $\{1, 2, \ldots, n\}$ where n is the number of cells, otherwise `false`. So, for example, if there are 10 cells the numbers would be $1, 2, \ldots, 10$ with no missing or duplicate numbers.

Finally, use the separate methods above to complete the `isTableau` method for determining whether an `int[][]` represents a tableau.

Reflection and extension

- What is the complexity of your method for checking that the entries of the tableau form the set $\{1, 2, \ldots, n\}$? One natural implementation is of quadratic (i.e., $O(n^2)$) but not linear ($O(n)$) complexity. Can you think of a linear one – or convince yourself that yours is?

  This is related to an issue that we will begin exploring in Lecture 11 – the representation and manipulation of *sets*.

- The `tableauToString` function works nicely if the numbers involved have three digits or fewer including sign. Of course, for reasonably-sized actual tableaux this is fine, but more generally if we were allow arbitrary entries (perhaps skipping the 1 to n condition) things get messy. How could it be improved so that the result is nicely aligned even if some of the numbers are quite large? That is, how could it dynamically take into account the size of the entries in the doubly indexed array?

- Your mathematician friend who wants to work with tableaux only now gets around to telling you that the tableaux she is interested in are generated by a sequence of additions of cells. Her plan is to test some conjectures about tableaux which will involve building up millions or possibly billions of tableaux in this way. Why is the tableau representation we have been using here *not* a good idea for this project?

- If you look at a tableau on its side (i.e., read each column from top to bottom as a row from left to right) you get another tableau called its *conjugate*. How would you write a method that, given a tableau, returned the conjugate tableau?