Com S 227
Spring 2019
Miniassignment 3
30 points

Due Date: Thursday, April 18, 11:59 pm (midnight)

5% bonus for submitting 1 day early (by 11:59 pm April 17)

10% penalty for submitting 1 day late (by 11:59 pm April 19)

No submissions accepted after April 19, 11:59 pm

General information

This assignment is to be done on your own. See the Academic Dishonesty policy in the syllabus, http://www.cs.iastate.edu/~cs227/syllabus.html#ad **, for details.**

You will not be able to submit your work unless you have completed the *Academic Dishonesty policy acknowledgement* on the Homework page on Canvas. Please do this right away.

If you need help, see your instructor or one of the TAs. Lots of help is also available through the Piazza discussions.

Note: This is a miniassignment and the grading is automated. If you do not submit it correctly, you will receive at most half credit.

Overview

Imagine you are going to a Sandwich place for lunch. A sandwich in this restaurant can be customarily built from a number of ingredients, such as bread, meats, cheeses, and so on. Each ingredient has a number of options to choose from. For example there could be three options of bread, 2 options of meats, 2 options of cheeses and so on. Your job in this mini-assignment is to list out all the different combinations of ingredients that a customer can have when building a sandwich.

Specifically, you need to write a single recursive method that computes all the combinations. The number of options for each and every ingredient is stored in an array called `options`. That is, the length of `options` is the same as the number of the ingredients and `options` [i] stores the number of options available for ingredient *i*.

For example, a sandwich that uses three ingredients (bread, meat, and cheese) that can be chosen from three types of bread, 2 types of meats, and 2 types of cheeses can be encoded in the following **`options`** array:

```
int options = {3, 2, 2};
```

All the different combinations of ingredients that a customer can use to build a sandwich are:

`[0, 0, 0],` // use type 0 bread, type 0 meat, type 0 cheese
`[0, 0, 1],` // use type 0 bread, type 0 meat, type 1 cheese
…
`[2, 0, 1],` // use type 2 bread, type 0 meat, type 1 cheese
`[2, 1, 1],` // use type 2 bread, type 1 meat, type 1 cheese

Altogether, there are 3*2*2 = 12 different combinations of ingredients a customer can have. The method you are going to write should return all the different combinations in a 2-D array, with each row of which representing a unique combination of the ingredients. In our example above, the returned 2-D array will have 12 rows and 3 columns (12 combinations of the 3 ingredients).

> *Tip:* it is useful to remember that in Java, what we call a 2-D array is really just an array of arrays. In the above example, the result is a 12-element array, and each of the 12 elements ("rows") is a 3-element **`int`** array. For example, if **`combos`** is the 2-D array returned in this example, an easy way to print it row by row is:
>
> ```
> for (int[] row : combos)
> {
>   System.out.println(Arrays.toString(row));
> }
> ```

Think recursively

To solve this problem recursively, you may think in the following way. Suppose there are n ingredients and there are m_n options to choose from for the last ingredient. What you can do is to find all the combinations to build a sandwich using only the first $n-1$ ingredients, which can be achieved by making a recursive call. Now let **`int[ ][ ] combos =`** … be the returned combinations of the first n-1 ingredients of the recursive call. **`combos`** should be a 2-D array with n-1 columns and $m_1 \times m_2 \times \ldots \times m_{n-1}$ rows, where m_i represents the number of options available for the i^{th} ingredient.

For example, using the options {3, 2, 2} above, your recursive call for just the first two options {3, 2} would return the 2D array **`combos`** consisting of

```
[0, 0]
[0, 1]
[1, 0]
[1, 1]
[2, 0]
[2, 1]
```

Your next step is to include the last ingredient. Recall that there are m_n options for the last ingredient, so you will need **`combos.length`** times m_n rows for the 2-D array that contains all the combinations of all the n ingredients. To fill it, you can take each array within **`combos`** and duplicate it m_n times, increasing the length by 1, each time setting a different choice of the last ingredient. In the example above, each array is copied twice since the number of third options m_3 is 2. The result you return would be the 2D array:

```
[0, 0, 0]
[0, 0, 1]
[0, 1, 0]
[0, 1, 1]
[1, 0, 0]
[1, 0, 1]
[1, 1, 0]
[1, 1, 1]
[2, 0, 0]
[2, 0, 1]
[2, 1, 0]
[2, 1, 1]
```

> *Tip*: The static method **`Arrays.copyOf`** is an easy way to duplicate an array while increasing the length. For example, if **`arr`** is **`[2, 4, 6]`**, then the method call **`Arrays.copyOf(arr, arr.length + 1)`** returns the array **`[2, 4, 6, 0]`.**

Your class should be called **`Combinations`**, placed in a package called **`mini3`**. It should have a single method which **must** be implemented using recursion. The single method should be called:

```
public static int[][] getCombinations(int[] choices) {

}
```

You may assume that the parameter **`choices`** contains at least one **`int`** value and all the values are positive (1 or higher).

Sorting your combinations

In order to produce consistent outputs and also for the SpecChecker to correctly evaluate your method, you need to sort your combinations (a 2-D array) at the end of your method before returning it. You are provided in this mini-assignment with a class called **`ArrayComparator`** that you will need to sort the combinations. To use **`ArrayComparator`** to sort, do the following:

```
Arrays.sort(combinations, new ArrayComparator());
```

where **`combinations`** is the 2-D array you are returning.

The class **`ArrayComparator`** is provided for you and you should not modify it.

Testing and the SpecChecker

A SpecChecker will posted shortly that will perform an assortment of functional tests. As long as you submit the assignment correctly, your score will be exactly the score reported by the specchecker.

However, when you are debugging, it is much more helpful if you have a simpler test case of your own that reproduces the error you are seeing.

Since no test code is being turned in, you are welcome to post your tests on Piazza for others to use and comment on.

Documentation and style

Since this is a miniassignment, the grading is automated and in most cases we will not be reading your code. Therefore, there are no specific documentation and style requirements. However, writing a brief descriptive comment for each method will help you clarify what it is you are trying to do.

If you have questions

For questions, please see the Piazza Q & A pages and click on the folder **`miniassignment3`**. If you don't find your question answered, then create a new post with your question. Try to state the question or topic clearly in the title of your post, and attach the tag **`miniassignment3`**. *But remember, do not post any source code for the classes that are to be turned in.* It is fine to post

source code for general Java examples that are not being turned in. (In the Piazza editor, use the button labeled "pre" to have Java code formatted the way you typed it.)

If you have a question that absolutely cannot be asked without showing part of your source code, make the post "private" so that only the instructors and TAs can see it. Be sure you have stated a specific question; vague requests of the form "read all my code and tell me what's wrong with it" will generally be ignored.

Of course, the instructors and TAs are always available to help you. See the Office Hours section of the syllabus to find a time that is convenient for you. We do our best to answer every question carefully, short of actually writing your code for you, but it would be unfair for the staff to fully review your assignment in detail before it is turned in.

Any posts from the instructors on Piazza that are labeled "Official Clarification" are considered to be part of the spec, and you may lose points if you ignore them. Such posts will always be placed in the Announcements section of the course page in addition to the Q&A page. (We promise that no official clarifications will be posted within 24 hours of the due date.)

What to turn in

Note: You will need to complete the "Academic Dishonesty policy questionnaire," found on the Homework page on Blackboard, before the submission link will be visible to you.

Please submit, on Canvas, the zip file that is created by the SpecChecker. The file will be named **`SUBMIT_THIS_mini3.zip`**. and it will be located in the directory you selected when you ran the SpecChecker. It should contain one directory, **`mini3`**, which in turn contains two files, **`Combinations.java`**, **`ArrayComparator.java.`**

Always LOOK in the zip file the file to check what you have submitted!

Submit the zip file to Canvas using the Miniassignment3 submission link and verify that your submission was successful. If you are not sure how to do this, see the document "Assignment Submission HOWTO" which can be found in the Piazza pinned messages under "Syllabus, office hours, useful links."

We strongly recommend that you just submit the zip file created by the specchecker. If you mess something up and we have to run your code manually, you will receive at most half the points.

We strongly recommend that you submit the zip file as created by the specchecker. If necessary for some reason, you can create a zip file yourself. The zip file must contain the directory **mini3**, which in turn should contain two

files, **`Combinations.java`**, **`ArrayComparator.java`**. You can accomplish this by zipping up the **src** directory of your project. The file must be a zip file, so be sure you are using the Windows or Mac zip utility, and not a third-party installation of WinRAR, 7-zip, or Winzip.