P01 Wardrobe Manager

Overview

Now that the holiday season is drawing to a close, it's time to catalog all of the horrible (and fantastic!) clothing you've received from friends and distant relatives. It'll be much easier to justify getting rid of things come summer if you can verify that you've never actually worn them, right?

For this programming assignment, you'll be creating a collection of utility methods to help manage a listing of clothing in your wardrobe. You'll be able to add items, record when you last wore them, remove them, and purge your whole closet of everything you haven't worn. Each item is listed as follows:

description	brand name	last worn date

Until you update an item's last worn date, it'll just be listed as "never".

This assignment is intended to be completed using primarily your prior knowledge from **BEFORE CS300!** Plan to get started as soon as possible, and don't hesitate to ask for help if you need it.

Grading Rubric

5 points	**Pre-assignment Quiz**: accessible through Canvas until 11:59PM on **01/28**.
+5%	**Bonus Points**: students whose ***final*** submission to Gradescope is before **5:00 PM Central Time** on **WED 01/31** *and* who pass ALL immediate tests will receive an additional 2.5 points toward this assignment, **up to a maximum total of 50 points**.
20 points	**Immediate Automated Tests**: accessible by submission to Gradescope. You will receive feedback from these tests *before* the submission deadline and may make changes to your code in order to pass these tests. Passing all immediate automated tests does **not** guarantee full credit for the assignment.
15 points	**Additional Automated Tests**: these will also run on submission to Gradescope, but you will not receive feedback from these tests until after the submission deadline.
10 points	**Manual Grading Feedback**: TAs or graders will manually review your code, focusing on algorithms, use of programming constructs, and style/readability.
50 points	**MAXIMUM TOTAL SCORE**

Learning Objectives

After completing this assignment, you should be able to:

- Comfortably **implement** a procedural program using static methods in Java
- **Explain** the protocols used with oversize arrays
- Verify the correctness of a static method which uses arrays by **designing** and **implementing** a boolean tester method in Java

Additional Assignment Requirements and Notes

Keep in mind:

- Pair programming is **NOT ALLOWED** for this assignment. You must complete and submit P01 individually.
- The ONLY external libraries you may use in your program are:
  `java.util.Arrays` (ONLY in the tester class)
- Use of *any* other packages (outside of java.lang) is NOT permitted.
- You are allowed to define any **local** variables you may need to implement the methods in this specification (inside methods). You are NOT allowed to define any additional instance or static variables or constants beyond those specified in the write-up.
- You are allowed to define additional **private** helper methods.
- Only the `WardrobeManagerTester` class may contain a main method.
- All classes and methods must have their own Javadoc-style method header comments in accordance with the CS 300 Course Style Guide.
- Any source code provided in this specification may be included verbatim in your program without attribution.
- All other sources must be cited explicitly in your program comments, in accordance with the Appropriate Academic Conduct guidelines.
- Any use of ChatGPT or other large language models ***must be cited*** AND your submission MUST include a file called `log.txt` containing the full transcript of your usage of the tool. Failure to cite or include your logs is considered academic misconduct and will be handled accordingly.
- **Run your program locally before you submit to Gradescope**. If it doesn't work on your computer, *it will not work on Gradescope*.

Need More Help?

Check out the resources available to CS 300 students here:
https://canvas.wisc.edu/courses/398763/pages/resources

CS 300 Assignment Requirements

You are responsible for following the requirements listed on both of these pages on all CS 300 assignments, whether you've read them recently or not. Take a moment to review them if it's been a while:

- Appropriate Academic Conduct, which addresses such questions as:
  - How much can you talk to your classmates?
  - How much can you look up on the internet?
  - How do I cite my sources?
  - and more!
- Course Style Guide, which addresses such questions as:
  - What should my source code look like?
  - How much should I comment?
  - and more!

Getting Started

1. Create a new project in Eclipse, called something like **P01 Wardrobe Manager**.
   a. Ensure this project uses Java 17. Select "JavaSE-17" under "Use an execution environment JRE" in the New Java Project dialog box.
   b. Do **not** create a project-specific package; use the default package.
2. Download one (1) Java source file from the assignment page on Canvas:
   a. **WardrobeManagerTester.java** (includes a main method)
3. Create one (1) Java source file within that project's src folder:
   a. **WardrobeManager.java** (does NOT include a main method)

All methods in this program will be **static** methods, as this program focuses on procedural programming.

Implementation Requirements Overview

Your **WardrobeManager.java** program must contain the following methods, described in detail on this javadoc page.

- `public static boolean containsClothing(String description, String brand, String[][] wardrobe, int wardrobeSize)`
- `public static int addClothing(String description, String brand, String[][] wardrobe, int wardrobeSize)`
- `public static int indexOfClothing(String description, String brand, String[][] wardrobe, int wardrobeSize)`
- `public static boolean wearClothing(String description, String brand, String date, String[][] wardrobe, int wardrobeSize)`
- `public static int getBrandCount(String brand, String[][] wardrobe, int wardrobeSize)`
- `public static int getNumUnwornClothes(String[][] wardrobe, int wardrobeSize)`
- `public static int getMostRecentlyWorn(String[][] wardrobe, int wardrobeSize)`
- `public static int removeClothingAtIndex(int index, String[][] wardrobe, int wardrobeSize)`
- `public static int removeAllUnworn(String[][] wardrobe, int wardrobeSize)`

All of these methods use an ***oversize array***, defined by a two-dimensional (n x 3) array of Strings and an integer value denoting the compact number of initialized (non-null) values in the first dimension of the array. That is, if the integer value `wardrobeSize` is 5, I would expect there to be length-3 arrays of Strings at `wardrobe[0]` through `wardrobe[4]`, and any other indexes in `wardrobe` will be `null`.

An Aside: Using Javadocs to build a program

The Javadoc page linked above is automatically generated by the IDE, using the contents of Javadoc-style comments in the program source code. When creating this assignment, I wrote WardrobeManger with comments for the class (a "class header" comment) and each method (a "method header" comment) in the following style:

```
/**
 * Finds the location (index) of a provided clothing item in an oversize array defined by
 * the provided two-dimensional array of strings and its size. If the item is NOT present
 * in the array, returns -1.
```

```
 *
 * @param description  a general description of the clothing item
 * @param brand       the brand of the clothing item, or "handmade"
 * @param wardrobe    a two-dimensional array of Strings, which stores wardrobe entries.
 *                    wardrobe[i][0] contains a description of item i,
 *                    wardrobe[i][1] contains its brand name, and
 *                    wardrobe[i][2] contains its last-worn date formatted as
 *                    "YYYY-MM-DD", or "never"
 * @param wardrobeSize number of items currently stored in the wardrobe
 * @return            the index of the clothing item if it is present, or -1 if it is not
 */
```

Both class and method header comments begin with a `/**` (two asterisks) and end with a `*/` – note that if you only use one asterisk at the beginning, this creates a multiline comment, not a Javadoc comment.

This is important documentation, and ***you will be required to do the same*** for all classes and methods you write in this course. You are welcome to use any text from the writeups, verbatim, in your comments. You are not required to generate HTML javadoc pages, but you can if you want to!

At the top of the javadoc page you will find a summary of the data fields and methods in the class, and if you either click on their names or scroll down, you will find the full detailed description of what these methods are supposed to do, what parameters they expect, and what they should return.

Most – if not all – of our programming assignments this semester will use these generated Javadoc pages to communicate the requirements of the assignment to you, so if you have any questions about how to use them, this is the time to ask.

Beginning the program: writing the tester methods

Add the methods listed on the javadoc page as ***stub methods*** – methods containing only a default return statement – to your class file, and then turn to the tester file before you go any further.

A link to the **WardrobeManagerTester.java** starter file can be found on the assignment page, so we haven't provided a Javadoc file for it. Several test methods have been provided for you in their entirety, along with some implementation-level comments to give you a guide in constructing your test methods.

While there are a LOT of tester methods, they should all be relatively straightforward. Use the provided methods as a model for setting up your own, and don't be afraid to use Arrays class methods as a shortcut for verifying the state of the 2-dimensional array after a method call.

My naming conventions for this assignment's test methods are as follows:

- `testMETHODEmpty()` - this method should test the behavior of the corresponding method on an empty oversize array; that is, all values in the 2-dimensional array should be null and the integer size value should be 0.
- `testMETHODFull()` - this method should test the behavior of the corresponding method on a completely full oversize array; that is, all values in the 2-dimensional array should be initialized to

valid clothing entries with no null values, and the integer size value should be equal to the length of the array.

- `testMETHODNoMatch()` - none of the values in the provided oversize array should match your query. If the method cares about description/brand, the query should not match any of the description/brand pairs; if the method cares about whether something has never been worn, none of the clothes should have a last-worn date of "never".
- `testMETHODAllMatch()` - *all* of the values in the provided oversize array should match your query.
- `testMETHODSomeMatch()` - some, but not all, of the values in the provided oversize array should match your query.
- `testMETHODFirst()` - the *first* item in the oversize array should match your query.
- `testMETHODLast()` - the *last* item in the oversize array should match your query. (This has sometimes been combined with the previous method.)
- `testMETHODMiddle()` - an item in the oversize array which is neither the first nor last item should match your query.
- `testMETHODBadIndex()` - the query index should be outside of the bounds of the oversize array, either too small or too big (or one test for each case).
- `testMETHODTrue()` - your test inputs for this boolean method should be such that you expect the method to return true.
- `testMETHODFalse()` - your test inputs for this boolean method should be such that you expect the method to return false.

Yes, there are a lot of things to test – but there are a lot of things that could go wrong! Don't rely on our provided tests here or on Gradescope to verify the functionality of your programs this semester.

We ***strongly recommend*** that you implement *at least one* tester method for each method in WardrobeManager BEFORE you write the WardrobeManager method itself.

Implementation Details and Suggestions

For this iteration of the wardrobe manager program (yes, it *will* be coming back), we will assume that the data is stored in arrays created by and stored in an external main class – for you, the tester class. Your WardrobeManager will use these arrays as provided, and you can assume everything is correctly formatted and there are no null values between indexes 0 and `wardrobeSize`-1.

Is this clothing already present in the wardrobe?

Begin by adding the `containsClothing()` method to WardrobeManager as a **stub method**:

```
public static boolean containsClothing(String description, String brand,
    String[][] wardrobe, int wardrobeSize) {
  return false;
}
```

In a stub method, you include only a return statement with some default value to satisfy the compiler, which expects this method to return a **boolean**.

Before going any further with this implementation, check out the `testContainsTrue()` method in the tester class. We've provided it for you in its entirety.

All test methods you write this semester will look something like this (this is a **stub** version):

```
/**
 * Checks whether method() works as expected
 * @return true if method functionality is verified, false otherwise
 */
public static boolean testMethod() { return false; }
```

Note there are NO parameters, the method is STATIC, and it returns a BOOLEAN value.

You can test as many things as you like in a single method, but as soon as any one check **fails**, your method should **stop** and return false – especially if later checks in the test assume that a previous check was successful. For example, consider these lines from the provided `testAddToEmpty()` method:

```
// (3) verify that the provided array was updated correctly
if (empty[0] == null) return false;
if (!empty[0][0].equalsIgnoreCase("green crop top") ||
    !empty[0][1].equalsIgnoreCase("H&M") || !empty[0][2].equals("never"))
    return false;
```

If you continue with the second if-statement test once the first if-statement has failed – meaning that `empty[0]` is null – this will cause a NullPointerException as you try to index into an array which does not exist (`null` does not have an index 0). There is no requirement in CS300 that methods have only one exit point; you are welcome to return in as many different places in a single method as you wish.

At this point, finish out the other tester methods in the CONTAINS section, and then implement the containsClothing() method back over in WardrobeManager. If you were successful, running the tester class should produce output that starts with:

```
CONTAINS:
  pass, pass, pass
```

Tester method pro-tips

- Keep your tester methods simple. Writing complex tester methods is a great way to introduce MORE bugs, which will make things even more complicated.
- If you have a problem with your code, implement a test method to quantify what IS happening vs. what you think SHOULD be happening. Sharing tests like this with course staff can help us help you more efficiently.
- Write helper methods – if you find yourself doing certain things over and over in different tester methods, don't be afraid to add a **private** helper method to the tester class to do that thing.

Now draw the rest of the owl[1]

Using the previous section as a guide and the Javadoc page as a reference, implement *at least one* test method for each WardrobeManager method and then implement the method itself. (Ideally, implement **all** of the associated test methods first.)

We've provided some sample wardrobes in the tester file, which you're welcome to reuse or modify as you see fit in your own tests. Your methods can assume that the input array is set up correctly (the 2-dimensional array is Nx3, with the lowest-index entry at index 0 and the highest at wardrobeSize-1, with no nulls between those indexes and only nulls after them, and no null values in the length-3 arrays), and your test methods should provide arrays that also follow those rules.

Reference: Valid Wardrobes

For reference, here are some examples of valid Nx3 wardrobe arrays. Note that all values outside of the index range 0 to (wardrobeSize-1) are set to `null`, and all `null` values must be at the END of the wardrobe array.

This list has **length** 8 but **size** 5:

```
{ {"green crop top", "H&M", "2024-01-02"},
  {"black t-shirt", "Gildan", "2023-10-31"},
  {"cargo pants", "GAP", "2023-12-29"},
  {"christmas sweater", "handmade", "2023-12-25"},
  {"black halter", "asos", "never"},
  null, null, null
}
```

This list has **length** 8 but **size** 0, and is what we'd call "empty":

```
{ null, null, null, null, null, null, null, null }
```

[1] https://www.reddit.com/r/funny/comments/eccj2/how_to_draw_an_owl/

This list has **length** 5 and **size** 5, with NO room for more, and is what we'd call "full" (size == length, contains no `null` values):

```
{ {"green crop top", "H&M", "2024-01-02"},
  {"black t-shirt", "Gildan", "2023-10-31"},
  {"cargo pants", "GAP", "2023-12-29"},
  {"christmas sweater", "handmade", "2023-12-25"},
  {"black halter", "asos", "never"}
}
```

Note that while the *initialized* contents and sizes of the first and third lists are the same, they would NOT be considered equal by `Arrays.deepEquals()` because they have different **lengths**.

Assignment Submission

Hooray, you've finished this CS 300 programming assignment!

Once you're satisfied with your work, both in terms of adherence to this specification and the academic conduct and style guide requirements, submit your source code through Gradescope.

For full credit, please submit the following files (**source code**, *not* .class files):

- WardrobeManager.java
- WardrobeManagerTester.java

Additionally, if you used any generative AI at any point during your development, you must include the full transcript of your interaction in a file called

- log.txt

Your score for this assignment will be based on the submission marked "**active**" prior to the deadline. You may select which submission to mark active at any time, but by default this will be your most recent submission.

Students whose final submission (which must pass ALL immediate tests) is made before 5pm on the Wednesday before the due date will receive an additional 5% bonus toward this assignment. Submissions made after this time are NOT eligible for this bonus, but you may continue to make submissions until 10:00PM Central Time on the due date with no penalty.

Copyright notice