Homework 03 - Inventory Manager

Problem Description

You have decided you want to create an Inventory management system, which can keep track of items, their weights, and their prices.

Solution Description

We have provided you with the skeletons of three classes: Item, Inventory and InventoryManager. Your task is to fill them in so that they meet the following requirements:

Item

- You will create two instance fields in Item:
  - A `double` field named "weight".
  - A `double` field named "price".
- In addition, you will fill in the first constructor. Just fill in the `TODO` lines provided.

Inventory

- You will create an `Item[]` field in Inventory called `contents`.
- You will write the `merge` method. This works as follows:
  - You are given another inventory. You can access its `contents` with `other.contents`.
  - You need to first figure out how big the merged array should be. You must add together the lengths of `contents` and `other.contents`.
  - You need to create a new `Item[]` whose size is the sum we computed above.
  - You need to copy all of `contents` into that `Item[]` created above.
  - You need to copy all of `other.contents` into that `Item[]` created above.
- You need to write the `getItem` method. This works as follows:
  - First, check if the given index is valid. (Think about when an array index is valid versus invalid.) If it is invalid, return null.
  - Otherwise, if it is valid, return the value in `contents` at the given index.
- You need to write the `putItem` method. This works as follows:
  - First, check if the given index is valid. (Think about when an array index is valid versus invalid.) If it is invalid, return false.
  - Otherwise, if it is valid, set the value in `contents` at the given index to `item` and return true.
- You need to write the `getContentsLength` method, which just returns the length of `contents`.

InventoryManager

- You will fill in a `mergeInventories` method, which just takes in two `Inventory`s and merges them using the methods you wrote above.
- You will fill in a `addItem` method, which just takes in an `Inventory` to put the item into, an `Item`, to put in the `Inventory`, and an `index` for where in the `Inventory` to put the `Item`. If it doesn't work, print `"Can't add there."`.
- You will fill in a `printItem` method, which takes in an Item and prints `(name, price, weight)`, where `name`, `price` and `weight` are filled in with the proper values from Item. `price` and `weight` should be printed to two decimal points. If the Item is null, print `(empty, 0, 0)`. Do not print a new-line after this.
- You will fill in a `showInventory` method. If the Inventory is `null`, or if getContentsLength returns a value less than or equal to zero, just print `()`. Otherwise, print a comma separated list of the Items in the Inventory's `contents` array. An example is given below:

  ```
  (A, 0.0, 0.0), (B, 1.0, 0.0), (empty, 0, 0), (empty, 0, 0), ...
  ```

Your formatting (spaces, no extra commas, etc.) must match this exactly. Print a new-line after your list or after the "()".

Allowed Imports

To prevent trivialization of the assignment, you are only allowed to import the following classes:

- java.util.Scanner
- java.util.Random

If you would like to import anything else, ask on Piazza. Based on what we have seen in the past, we'll go ahead and say you **cannot import the following classes:**

- java.util.Arrays
- java.util.ArrayList
  - Or any List

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach. For that reason, do not use any of the following in your final submission:

- var (the reserved keyword)
- System.arraycopy

In addition, you may not use `System.exit` as it will break the autograder.

Rubric

Make any specific notes about the rubric here.

All possible partial credit should be clear in the rubric as a nested bullet. Each bullet (or nested bullet) should correspond to an autograder test or something that TAs have to check manually. Be sure to follow the exact spacing (empty lines and 4 spaces before nested bullet points).

[20] **Item**

- [10] Created `weight` and `price`
  - [5] per-field
- [10] Correct random assignment of `price` and `weight`
  - [5] per-field

[40] **Inventory**

- [5] `contents` field
  - [2.5] `contents` named correctly
  - [2.5] `contents` has correct initial capacity
- [15] `merge` method
  - [15] `merge` functions properly
- [5] `getItem` method
  - [5] `getItem` functions properly
- [10] `putItem` method
  - [5] `putItem` functions properly on its own
  - [5] `putItem` functions with merge as well
- [5] `getContentsLength` method
  - [5] `getContentsLength` functions properly

[40] **InventoryManager**

- [5] `mergeInventories` method

  - [5] merges `i2` into `i1`
- [5] `addItem` method
  - [2.5] adds correctly in normal case
  - [2.5] prints correct error if adding fails
- [15] `printItem` method
  - [5] prints null item correctly
  - [10] prints non-null items correctly
- [15] `printInventory` method
  - [5] prints empty inventory correctly
  - [10] prints non-empty inventory correctly

Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **10** points. Review the style guide and download the checkstyle jar. Run checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.2.jar *.java
Audit done. Errors (potential points off):
0
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the checkstyle cap mentioned above). The Java source files we provide contain no Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Depending on your editor, you might be able to change some settings to make it easier to write style-compliant code. See the customization tips page for more information.

Collaboration

Collaboration Statement

To ensure that you acknowledge a collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Allowed Collaboration

When completing homeworks for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.

Examples of approved/disapproved collaboration:

- **approved**: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"
- **disapproved**: "Yo it's 10:40 on Thursday... Can I see your code? I won't copy it directly I promise"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `Item.java`
- `Inventory.java`
- `InventoryManager.java`

Make sure you see the message stating "HW03 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

1) Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2) Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment are also autograded once the submission deadline has passed. The autograders used are often dependent on specific output formats, so **make sure that your submission passes all test cases marked "FORMAT:"**.

Some of your files will depend on other files in your submission. If one file breaks and also causes another to break, you'll lose points in both files. The public tests will help ensure you don't have these dependency issues, but it is ultimately up to you to ensure it functions correctly.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Ensure you pass all "FORMAT:" tests
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for a note containing all official clarifications