Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-002-S2024/it114-m2-java-problems/grade/ns87

IT114-002-S2024 - [IT114] M2 Java Problems

Submissions:

Submission Selection
1 Submission [active] 2/5/2024 3:51:20 PM

Instructions

COLLAPSE

Guide:

1. Make sure you're in the main branch locally and `git pull origin main` any pending changes
2. Make a new branch per the recommended branch name below (git checkout -b ...)
3. Grab the template code
   from https://gist.github.com/MattToegel/fdd2b37fa79a06ace9dd259ac82728b6
4. Create individual Java files for each problem and save the files inside a subfolder of your choice
    1. The should end with the file extension in lowercase .java
5. Move the unedited template files to github
    1. `git add .`
    2. `git commit -m "adding template files`
    3. `git push origin <homework branch>` (see below and don't include the < >)
    4. Create and open a pull request from the homework branch to main (leave it open until later steps)
6. Note: As you work, it's recommended to add/commit at least after each solution is done (i.e., 3+ times in this case)
    1. Make sure the files are saved before doing this
7. Fill in the items in the worksheet below (save as often as necessary)
8. Once finished, export the worksheet
9. Add the output file to any location of your choice in your repository folder (i.e., a Module2 folder)
10. Check that git sees it via `git status`
11. If everything is good, continue to submit
    1. Track the file(s) via `git add`
    2. Commit the changes via `git commit` (don't forget the commit message)
    3. Push the changes to GitHub via `git push` (don't forget to refer to the proper branch)
    4. Create a pull request from the homework related branch to main (i.e., main <- "homework branch")
    5. Open and complete the merge of the pull request (it should turn purple)
    6. Locally checkout main and pull the latest changes (to prepare for future work)
12. Take the same output file and upload it to Canvas
    1. *This step is new since GitHub renders the PDF as an image the links aren't clickable so this method works better
    2. *Remember, the github process of these files are encouragement for your tracking of your progress

Branch name: M2-Java-Problems

Branch name: M2-Java-Problems

Tasks: 8 Points: 10.00

COLLAPSE

Problem 1 (3 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of the Problem 1 Solved Code and Output

Details:
Only make edits where the template code mentions.

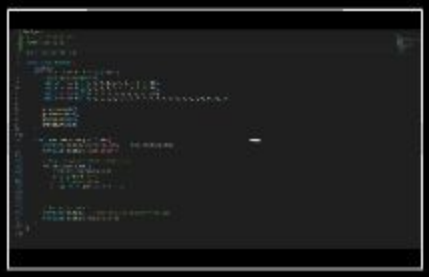
Solution should ensure that any passed in array will have only the odd values output.
Requires at least 2 screenshots (code + output from terminal)

Checklist *The checkboxes are for your own tracking

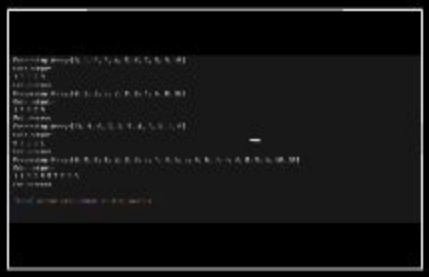
#	Points	Details
#1	1	Edits were done only in the processArray() method and original template code/comments remain untouched
#2	1	Only arr is used (no direct usage of a1, a2, a3, a4)
#3	5	Only odd values output (not odd indexes/keys)
#4	1	Includes code comments with student's ucid and date
#5	1	Terminal output is fully visible

Task Screenshots:

Large Gallery



Checklist Items (0)



Checklist Items (0)

Code | Output

COLLAPSE

Task #2 - Points: 1

Text: Explain your solution

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Clearly explains how the code/logic solves the problem (mentions how the odd values are determined)

Response:

The code solves the problem by iterating through each element in the input array (arr). Inside the processArray method, a for-each loop is used to traverse the array.

For each element (num) in the array, the code checks if it is an odd number by using the condition num % 2 != 0. This condition checks if the remainder when dividing num by 2 is not equal to 0, which is a characteristic of odd numbers. If the condition is true, the number is considered odd, and it is printed to the console.

This process ensures that only the odd values present in the array are outputted. The logic adheres to the requirement of displaying only the odd values and avoids direct usage of specific arrays (a1, a2, a3, a4) as instructed in the problem statement.

In summary, the code iterates through the array, identifies odd numbers based on the modulo operation, and prints only the odd values to the console.

COLLAPSE

Problem 2 (3 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of the Problem 2 Solved Code and Output

Details:

Only make edits where the template code mentions.

Solution should ensure that any passed in array will have its values converted to a positive version of the value AND converted back to the original data type.
Requires at least 2 screenshots (code + output from terminal)

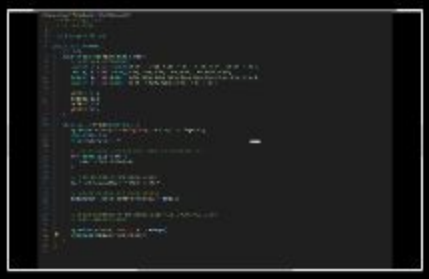
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Edits were done only in the getTotal() method and original template code/comments remain untouched (unless noted)
#2	1	Only arr is used (no direct usage of a1, a2, a3, a4)
#3	5	Passed in array's values get summed AND rounded to two decimal places like currency (i.e., 0.00, 0.10, 1.10)
	1	Includes code comments with student's ucid and date

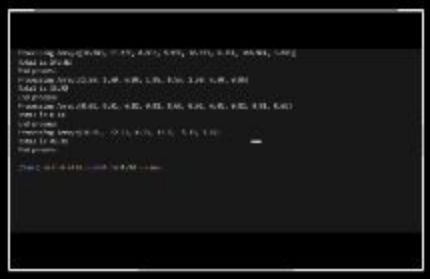
#4	1	Includes code comments with student's solid and date
#5	1	Terminal output is fully visible

Task Screenshots:

Large Gallery



Checklist Items (0)



Checklist Items (0)

Problem 2 code	Problem 2 output

COLLAPSE

Task #2 - Points: 1

Text: Explain your solution

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Clearly explains how the code/logic solves the problem (mentions both how the values get summed and how the rounding is solved correctly)

Response:

The provided code addresses the problem by ensuring that any passed-in array has its values converted to their absolute (positive) versions. This is achieved through the use of Math.abs(value) when iterating through the elements of the array. The absolute values are then summed up to calculate the total.

To address the rounding requirement, the code utilizes Math.round(total * 100.0) / 100.0. This expression rounds the total to two decimal places by multiplying it by 100.0, rounding it to the nearest integer, and then dividing it back by 100.0. This ensures that the total is rounded to two decimal places, simulating currency format (e.g., 0.00, 0.10, 1.00).

The formatted total is then converted to a string using String.format("%.2f", total), and the result is printed out along with the processed array. The code effectively handles both the summation of absolute values and the rounding to two decimal places, meeting the specified requirements for the problem.

COLLAPSE

Problem 3 (3 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of the Problem 2 Solved Code and Output

Details:

Only make edits where the template code mentions.

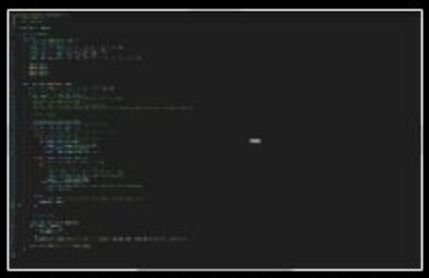
Solution should ensure that any passed in array will have its values converted to a positive version of the value AND converted back to the original data type.
Requires at least 2 screenshots (code + output from terminal)

Checklist *The checkboxes are for your own tracking

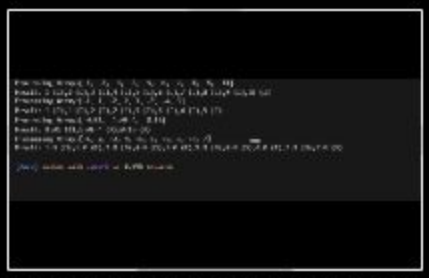
#	Points	Details
☐ #1	1	Edits were done only in the bePositive() method and original template code/comments remain untouched
☐ #2	1	Only arr is used (no direct usage of a1, a2, a3, a4)
☐ #3	5	Passed in array's values will get converted to a positive version AND converted back to the original data type
☐ #4	1	Includes code comments with student's ucid and date
☐ #5	1	Terminal output is fully visible

Task Screenshots:

Large Gallery



Checklist Items (0)



Checklist Items (0)

code | Output

COLLAPSE

Task #2 - Points: 1

Text: Explain your solution

Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Clearly explains how the code/logic solves the problem (mentions both the conversion to positive and conversion to original data type)

Response:

The provided code solves the problem by creating a generic method bePositive that takes an array of any type as input and processes each element according to the following logic:

It initializes an output array of Object type to store the result.
It iterates through each element of the input array (arr) using a loop.
For numeric types (Integer or Double), it converts the element to its positive equivalent using Math.abs() and stores

the result in the corresponding index of the output array.
For String types, it attempts to parse the string as a Double. If successful, it converts the absolute value of the parsed Double to ensure it is positive and stores the result in the output array. If parsing fails, it retains the original string value in the output array.
For non-numeric types, it simply copies the original value to the output array.
The method then prints the original array and the resulting array with elements converted to positive values and maintains the original data types.
This approach ensures that the passed-in array's values are converted to their positive versions and then converted back to the original data type, taking into account various numeric and string scenarios. The final result is printed in a formatted manner showing each element along with its data type.

COLLAPSE

Reflection (1 pt.)

COLLAPSE

Task #1 - Points: 1

Text: Reflect on your experience

Details:

Talk about any issues you had, how you resolved them, and anything you learned during this process.

Provide concrete details/examples.

Response:

Problem 1:

Issue: The "End process" line was inside the loop, which would print it multiple times for each element in the array.

Resolution: Moved the line outside of the loop to ensure it's printed only once after processing all elements.

Lesson Learned: Proper placement of statements is crucial to achieve the desired output.

Problem 2:

Issue: The comment suggested ensuring rounding to two decimal places, but the code rounded to only one decimal place.

Resolution: Adjusted the rounding statement to correctly round to two decimal places.

Lesson Learned: Pay attention to details and comments, ensuring code aligns with the intended functionality.

Problem 3:

Issue: The comments suggested an "end edit section," but the actual edits were spread throughout the method.

Resolution: Grouped the edits in one place for clarity, following the provided comments.

Lesson Learned: Consistency in commenting and code structure enhances readability and maintainability.

COLLAPSE

Task #2 - Points: 1

Text: Include the pull request link for this branch

Details:

The correct link will end with /pull/ and a number.