**LAB 1 REPORT**

**Problem 1**

* 1. **Convert array to number**

A screen shot of a computer

Description automatically generated

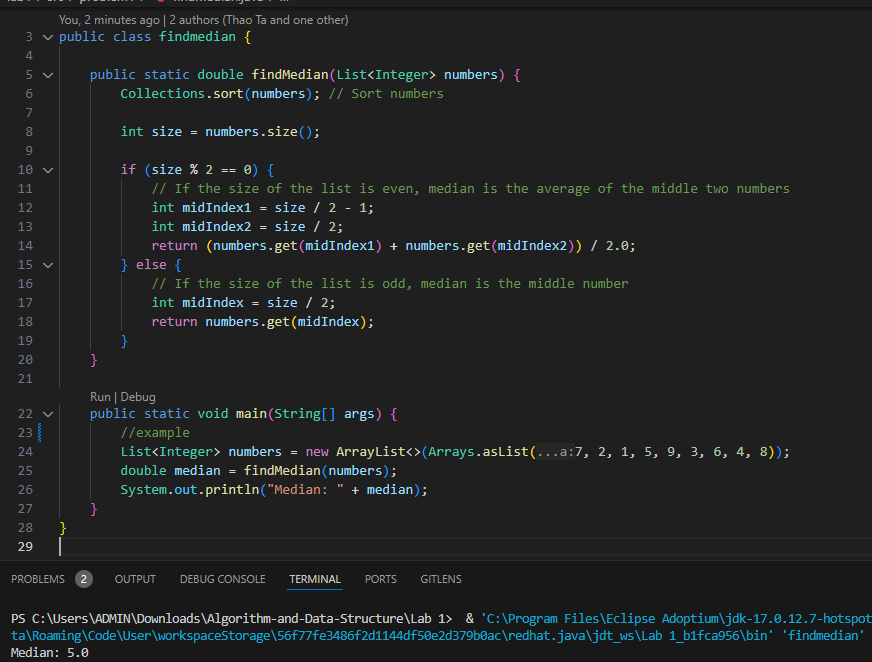
First, we use class “arraytonumber” to define objects and organize code.

* **public**: This method can be accessed from outside the class.
* **static**: It belongs to the class rather than an instance of the class.
* **int arrayToNumber(int[] arr)**: This method takes an array of integers (arr) as input and returns an integer (result)

We can see that int result = 0; its mean that at the beginning, “result” has nothing inside so that it can fill the interger element in the “result”

After that, we use function to run from index[i] to the end of array, each time [i] run, “result” will x10 and + [i] so that “result” always has a slot to contain [i], making it from array to number.

* 1. **Find median**

****

In this work, we use function “findmedian” to find the median in the array.

In this function:

+ First we must sort the array, this is the essential step to find median, so we use “Collection.sort” to sort the array

+ Second, we need to check the array if it size even or odd:

* + If even (size %2 ==0) => mid index = (size /2 +1) + (size /2) / 2
  + If odd => mid index = size/2
  1. **Find the minGap**

**A screenshot of a computer program

Description automatically generated**

In this work, we find the minimum gap in the array between elements, we create class “mingap”

First, if the size of the array < 2 => the gap and the minimum gap = 0

Second, for the array > 2 , we need to find the gap with each elements in array using subtraction, due to that, [i] will locate at second element and keep running (i++) while [i] < n (size)

We could see in the example:

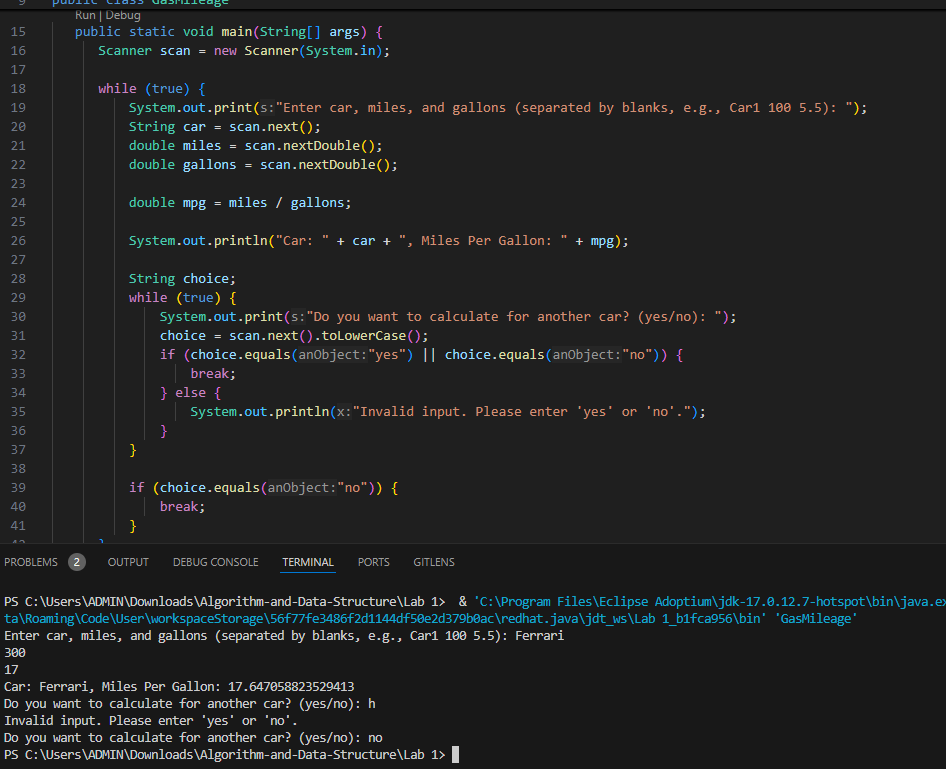
* index [i] = 2 => i = 3, [i-1] = 1 => i = 1
* gap = [i] – [i-1] = 3 -1 = 2; i++ = > [i] = 6 , [i-1] = 3

cont....

After run, we can have the array gap by {2, 3, 1, 5}

To find the minimum gap, we just need to use “Math.min” , so the minimum gap will be 1

* 1. **GasMileage**

****

The program prompts the user to enter the car's name, miles driven, and gallons used.

It calculates and displays the car's MPG = miles / gallons (we use double instead of float to make the result more accurate)

The program then asks if the user wants to calculate MPG for another car.

* If "yes" = >the program repeats. If "no" = > the program terminates.
* If the input is invalid (not "yes" or "no") = > it asks again until valid input is provided.
  1. **Student.java**

**A screenshot of a computer

Description automatically generated**

In this work, to handle the “students.txt” file, we use “throws IOException” which is necessary for handling potential file-related errors. This means if an error occurs while trying to read the file, it will propagate up and terminate the program.

There are some of types of elements that we need to concern:

* first\_name, last\_name: string
* grade, total ,count: integer
* avg of grade: double

First, we need to scan the file input to collect data from it, hasNext() method is crucial for safely iterating through input data. It helps ensure that the program only attempts to read data when there is more data available, preventing runtime errors and making the code more robust.

Second, identify the type of elements

Third, using while loop continues as long as there are more elements to read in the file, Inside the loop:

* The first\_name and last\_name are read as strings.
* The grade is read as an integer.
* After that, the “Student st” will be the place to contain the elements and print it out
* Total = total + grade (total grade of all students) and count = count +1 (number of students)

Avg\_grade = Total / count

Beside that, in “Student class”, there is a program that categorizes students into three performance types based on their grades:

* EXCELLENT: Grades above 89
* OK: Grades between 60 and 89 (inclusive)
* FAILURE: Grades below 60



Static variables:

* excellentCount, okCount, failureCount: Track the count of students in each respective category.
* excellentTotal, okTotal, failureTotal: Keep track of the total grades in each category.

=> When a new student is created, the constructor assigns the student to a category using the determineType() method, which classifies students based on their grade.

Final Output:

A screen shot of a computer

Description automatically generated

**Problem 2:**

* 1. **Low array.java**

A screen shot of a computer code

Description automatically generated

In this work:

First, we create class “LowArray” to manage an array of long integers, encapsulating methods to manipulate the array:

Constructor: Initializes the array with a specified size.

* private long[] a: A private array reference to store long integers.
* setElem(int index, long value): Allows setting a value at a given index in the array.
* getElem(int index): Retrieves the value at the specified index.

A screen shot of a computer screen

Description automatically generated

Second we use class “LowArrayApp” to Demonstrate the functionality of the “LowArray” class by performing various operations.

* Main method actions:
  + Creates an instance of LowArray with a size of 100.
  + Inserts 10 predefined values into the array using the setElem method.
  + Displays all current values in the array.
  + Searches for a specific value (26) and reports whether it was found.
  + Deletes a specific value (55) by shifting subsequent elements down and decrements the count of elements.
  + Displays the remaining values in the array after deletion.

*Output:*

A screen shot of a computer

Description automatically generated

* 1. **High Array app**A screen shot of a computer screen

     Description automatically generated

This is “HighArray” class, it is designed to manage an array of long integers with several functionalities including insertion, deletion, searching, and duplicate removal:

* public static final int length = 0;: This line defines a constant for the length but is not utilized in the class.
* private long[] a;: An array to store long integer values.
* private int nElements;: Keeps track of the number of elements currently in the array.

Constructor: public HighArray(int max): Initializes the array with the specified maximum size and sets nElements to 0, indicating that the array is empty.

Methods:

1. find(long searchKey): Searches for a specified value in the array.
   * Iterates through the array and returns true if the value is found; otherwise, returns false.
2. insert(long value): Inserts a new value at the next available position in the array and increments the count of nElements.
3. delete(long value): Deletes a specified value from the array.
   * Searches for the value, and if found, shifts subsequent elements down to fill the gap, decrementing nElements.
4. getMax(): Returns the maximum value in the array.
   * If the array is empty, it returns -1. Otherwise, it iterates through the array to find and return the maximum value.
5. removeMax(): Removes the maximum value from the array and returns it.
   * Similar to getMax, but also shifts elements to maintain the integrity of the array and decrements nElements.
6. noDups(): Removes duplicate elements from the array.
   * Creates a new temporary array to store unique elements, iterating through the original array, and then replaces the original array with the unique values.
7. show(): Displays all the current elements in the array.

After that, we are going to test the function in “HighArrayApp” class by creating predefined elements into the HighArray object, cheking the function whether working or not:

A screenshot of a computer program

Description automatically generated

*Output:*

A screenshot of a computer

Description automatically generated

**1.3 Ordered App**

As same as “HighArrayApp .java”, “Ordered App.java”I is the program includes operations such as insertion, deletion, searching, merging, and removing duplicates from the array.

In this “OrdArray”class”, the attributes are:

* private long[] a: An array that holds the elements of the ordered array.
* private int nElements:A counter for the number of elements currently in the array.

The methods:

* Constructor: Initializes the array with a specified maximum size and sets the number of elements to zero.
* size(): Returns the current count of elements.
* find(long searchKey): Implements binary search to locate a specified key, returning its index or the size if not found.
* insert(long value): Inserts a value while maintaining order, shifting elements as necessary.
* delete(long value): Removes a specified value from the array, shifting elements to fill the gap.
* get(int index): Retrieves the value at a given index.
* add(long value): Appends a value to the end of the array.
* merge(OrdArray A1, OrdArray A2, OrdArray destination): Merges two ordered arrays into a destination array, maintaining order.
* noDups(): Removes duplicate values from the array.
* show(): Displays all elements in the array.

A screen shot of a computer

Description automatically generated

After that, we are going to test the function in “OrdArray” class by creating predefined elements in the array with a maximum size of 100.

A screen shot of a computer

Description automatically generated

*Output:*

*A computer screen with white text

Description automatically generated*