



JOBSHEET 9

ARRAY 1

DAVA RIZKY H

254107020246

Objective

- Students are able to understand one-dimensional Array creation and accessing its elements in Java
- Students are able to make programs using the concept of one-dimensional arrays

Laboratory

Experiment 1: Fill in Array Element

Experiment Time: 20 minutes

1. Open a text editor, create a new Java class with the name **arrayNumbersXX**.
(XX=student ID number)
2. Write the basic structure of the Java programming language which contains the **main()** function
3. Create an array of integer type named **num** with a capacity of 4 elements

```
int[] num = new int[4];
```

4. Fill each element of the array with numbers 5, 12, 7, 20

```
num[0] = 5;  
num[1] = 12;  
num[2] = 7;  
num[3] = 20;
```

5. Display all contents of the elements to the screen

```
System.out.println(num[0]);  
System.out.println(num[1]);  
System.out.println(num[2]);  
System.out.println(num[3]);
```

6. Compile and run the program. Match the results of the running programs that you have created according to the following display



5
12
7
20

7. Commit and push the changes to GitHub.

```

File Edit Selection View Go Run ... < > PRAKTIKUMDASPRO-2 [Administrator]
arrayNumbers07.java SistemHotellengkap.java
jobsheet9 arrayNumbers07.java SistemHotellengkap.java
1 import java.util.Scanner;
2
3 public class arrayNumbers07 {
4     Run | Debug
5         public static void main(String[] args) {
6             Scanner sc = new Scanner(System.in);
7
8                 int [] num = new int [4];
9                 num[0] = 5;
10                num[1] = 12;
11                num[2] = 7;
12                num[3] = 20;
13
14                 System.out.println(num[0]);
15                 System.out.println(num[1]);
16                 System.out.println(num[2]);
17                 System.out.println(num[3]);
18
19             sc.close();
}
PROBLEMS OUTPUT TERMINAL ...
PS C:\Users\ASUS ROG\Documents\PRAKTIKUMDASPRO\PRAKTIKUMDASPRO-2> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\ASUS ROG\AppData\Roaming\Code\User\workspaceStorage\2fa80fc21927ae4156d2f9ab12e0554\redhat.java\jdt_ws\PRAKTIKUMDASPRO_2_68585b5b\bin' 'arrayNumbers07'
5
12
7
20
O PS C:\Users\ASUS ROG\Documents\PRAKTIKUMDASPRO\PRAKTIKUMDASPRO-2>
Ln 6, Col 9 Spaces: 4 UTF-8 CRLF Java Chat quota reached Go Live Prettier
08:03 04/11/2025

```

Questions!

- If the contents of each element of the array **num** are changed with numbers 5.0, 12867, 7.5, 2000000. What happens? How can it be like that?

The program will produce an **error** because the **int** data type cannot store decimal numbers such as 5.0 or 7.5. To use decimal values, the array data type must be changed to **double**.

Correct example:

```
double[] num = {5.0, 12867, 7.5, 2000000};
```

Explanation:

The error occurs due to a data type mismatch. Java cannot automatically convert a double value to int without explicit casting.



2. Modify the program code by initializing the array elements at the same time when declaring the array.

```

File Edit Selection View Go Run ... ← → 🔍 PRAKTIKUMDASPRO-2 [Administrator]
EXPLORER OPEN EDITORS arrayNumbers07.java jobsheet9 M
PRAKTIKUMDASPRO-2
  > DASPRO
  > jobsheet3
  > jobsheet4
  > jobsheet5
  > jobsheet6
  > jobsheet7
  > jobsheet8
  > jobsheet9
    > arrayNumbers07.java
    README.md
SISTEMHOTELLENGKAP
arrayNumbers07.java M
1 import java.util.Scanner;
2
3 public class arrayNumbers07 {
4     Run | Debug
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7         int[] num = {5, 12, 7, 20};
8
9         System.out.println(num[0]);
10        System.out.println(num[1]);
11        System.out.println(num[2]);
12        System.out.println(num[3]);
13
14        sc.close();
15    }
16
17

```

PROBLEMS OUTPUT TERMINAL ... Run: arrayNumbers07 + ⚡ 🗑️ ...

```

PS C:\Users\ASUS ROG\Documents\PRAKTIKUMDASPRO\PRAKTIKUMDASPRO-2> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\ASUS ROG\AppData\Roaming\Code\User\workspaceStorage\24fa86fc21927ae4156d2f9ab12e0554\redhat\java\jdt-ws\PRAKTIKUMDASPRO-2_688856b\bin'\arrayNumbers07
● 5
● 12
● 7
● 20

```

PS C:\Users\ASUS ROG\Documents\PRAKTIKUMDASPRO\PRAKTIKUMDASPRO-2>

Chat quota reached Go Live Prettier Agent Auto

08:14 04/11/2025

3. Change the statement in step 6 to be like this

```

for (int i = 0; i < 4; i++) {
    System.out.println(num[i]);
}

```

What is the result? How can it be like that?

The **for** loop prints all the elements of the array sequentially from index 0 to 3
The program displays every value stored in the array in order.



```

import java.util.Scanner;

public class arrayNumbers07 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] num = {5, 12, 7, 20};
        for (int i = 0; i < 4; i++) {
            System.out.println(num[i]);
        }
        sc.close();
    }
}

```

The screenshot shows a GitHub Desktop application window. On the left, there's a sidebar with 'SOURCE CONTROL' showing a repository named 'PRAKTIKUMDAS'. Under 'CHANGES', a commit is pending with the message 'arrayNumbers07.java jobsheet9'. Below it is a 'GRAPH' section showing a commit history for 'modify1'. The main area displays a Java code editor with the file 'arrayNumbers07.java'. The code contains a for-loop that prints elements from index 0 to 3 of an array 'num'. A syntax error is highlighted at the end of the loop. The output terminal shows the command run: 'arrayNumbers07' and its execution. The status bar at the bottom indicates the file is ready.

- If the condition in the for-loop statement is changed to `i <= 4`, what is the output of the program? Why is the result like that?

Answer:

The program will cause a **runtime error** called **ArrayIndexOutOfBoundsException**.

Explanation:

The array only has indexes from 0 to 3 (four elements in total). When the loop condition is changed to `i <= 4`, the program tries to access `num[4]`, which does not exist, resulting in an exception.

- Commit and push the changes to GitHub.

Experiment 2: Requesting User Input to Fill in an Array Element

Experiment Time: 40 minutes

- Open a text editor, create a Java file then save it with the name **arrayValueXX**. (XX=student ID number)
- Write the basic structure of the Java programming language which contains the **main()** function



3. Add the Scanner library
4. Create an array of integer type with the name **finalScore**, with a capacity of 10 elements

```
int[] finalScore = new int[10];
```

5. Using a loop, create an input to fill in the **finalScore** array element

```
for (int i = 0; i < 10; i++) {  
    System.out.print("Enter the final score " + i + ": ");  
    finalScore[i] = sc.nextInt();  
}
```

6. Using a loop, display all the contents of the elements from the **finalScore** array

```
for (int i = 0; i < 10; i++) {  
    System.out.println("Final score " + i + " is " + finalScore[i]);  
}
```

7. Compile and run the program. Match the results of the running programs that you have created according to the following display

```
Enter the final score 0: 78  
Enter the final score 1: 89  
Enter the final score 2: 94  
Enter the final score 3: 85  
Enter the final score 4: 79  
Enter the final score 5: 87  
Enter the final score 6: 93  
Enter the final score 7: 72  
Enter the final score 8: 86  
Enter the final score 9: 91  
Final score 0 is 78  
Final score 1 is 89  
Final score 2 is 94  
Final score 3 is 85  
Final score 4 is 79  
Final score 5 is 87  
Final score 6 is 93  
Final score 7 is 72  
Final score 8 is 86  
Final score 9 is 91
```

8. Commit and push the changes to GitHub.



```

import java.util.Scanner;
public class arrayValue07 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] finalScore = new int[10];

        for (int i = 0; i < 10; i++) {
            System.out.print("Enter the final score " + i + ": ");
            finalScore[i] = sc.nextInt();
        }

        for (int i = 0; i < 10; i++) {
            System.out.println("Final score " + i + " is " + finalScore[i]);
        }

        sc.close();
    }
}

```

The screenshot shows the IntelliJ IDEA interface with the following details:

- EXPLORER**: Shows files like `arrayNumbers07.java`, `jobsheet9`, and `arrayValue07.java`.
- EDITOR**: Displays the `arrayValue07.java` code.
- TERMINAL**: Shows the output of the program execution.
- OUTPUT**: Shows standard Java build logs.
- PROBLEMS**: Shows a warning about an unresolved reference to `sc`.
- CHAT**: Shows a message about reaching monthly chat messages quota.
- WORKSPACE**: Shows usage statistics for GitHub Copilot Pro.

Questions!

1. Change the statement in step 5 to be like this

```

for (int i = 0; i < finalScore.length; i++) {
    System.out.print("Enter the final score " + i + ": ");
    finalScore[i] = sc.nextInt();
}

```

Run the program. Have there been any changes? How can it be like that?



The screenshot shows the GitHub Copilot extension in Visual Studio Code. The code editor displays a Java file named `arrayValue07.java` with the following content:

```

public class arrayValue07 {
    public static void main(String[] args) {
        // Display all students and mark which one passed, and
        System.out.println("Final Results:");
        for (int i = 0; i < finalScore.length; i++) {
            if (finalScore[i] > 70) {
                System.out.println("Student " + i + " with score " + finalScore[i]);
            } else {
                System.out.println("Student " + i + " with score " + finalScore[i]);
            }
        }
        sc.close();
    }
}

```

The terminal below shows the output of the program:

```

Student 4 with score 80 - Passed!
Student 5 with score 79 - Passed!
Student 6 with score 70 - Did not pass
Student 7 with score 90 - Passed!
Student 8 with score 80 - Passed!
Student 9 with score 78 - Passed!

```

2. Apa yang dimaksud dengan kondisi `i < finalScore.length`?

`finalScore.length` is a property that returns the number of elements in the array

For our array with 10 elements, `finalScore.length` equals 10

The condition `i < finalScore.length` means "continue the loop while `i` is less than the array size"

This ensures we access only valid indices: 0, 1, 2, ..., 9 (never 10, which would cause an error)

3. Change the statement in step 6 to be like this, so that the program only displays the grades of students who passed, students who have a score > 70

```

for (int i = 0; i < finalScore.length; i++) {
    if (finalScore[i] > 70) {
        System.out.println("Student " + i + " Passed!");
    }
}

```

Run the program and describe the flow of the program!



The screenshot shows a GitHub Copilot interface within the Visual Studio Code (VS Code) environment. The code editor displays a Java file named `arrayValue07.java`. The code reads student scores from an array and prints them along with their status (Passed or Failed). A tooltip from GitHub Copilot suggests upgrading to Pro for better performance.

```

public class arrayValue07 {
    public static void main(String[] args) {
        // Display all students and mark which one passed, and which did not!
        System.out.println("Final Results:");
        for (int i = 0; i < finalScore.length; i++) {
            if (finalScore[i] > 70) {
                System.out.println("Student " + i + " with score " + finalScore[i] + " Passed!");
            } else {
                System.out.println("Student " + i + " with score " + finalScore[i] + " Failed!");
            }
        }
        sc.close();
    }
}

```

4. Modify the program so that it displays all students, and mark which one passed, and which did not!

```

Enter the final score 0: 87
Enter the final score 1: 65
Enter the final score 2: 78
Enter the final score 3: 95
Enter the final score 4: 92
Enter the final score 5: 58
Enter the final score 6: 89
Enter the final score 7: 67
Enter the final score 8: 85
Enter the final score 9: 78
Student 0 Passed!
Student 1 Failed!
Student 2 Passed!
Student 3 Passed!
Student 4 Passed!
Student 5 Failed!
Student 6 Passed!
Student 7 Failed!
Student 8 Passed!
Student 9 Passed!

```



The screenshot shows a GitHub Copilot Pro interface within a code editor. The code is a Java program named `arrayValue07.java` which reads student scores from a file and prints them out. The GitHub interface includes a sidebar for repositories and changes, and a chat window on the right.

```

public class arrayValue07 {
    public static void main(String[] args) {
        // Display all students and mark which one passed, and ...
        System.out.println("Final Results:");
        for (int i = 0; i < finalScore.length; i++) {
            if (finalScore[i] > 70) {
                System.out.println("Student " + i + " with score " + finalScore[i]);
            } else {
                System.out.println("Student " + i + " with score " + finalScore[i]);
            }
        }
        sc.close();
    }
}

```

- Commit and push the changes to GitHub.

2.3 Experiment 3: Perform Arithmetic Operations on Array Elements

Experiment Time: 75 minutes

This experiment is done to add array elements. The program will accept input of 10 student scores. Then the program will display the average score of 10 students.

- Open a text editor, create a Java file then save it with the name **arrayAverageScoreXX**.
(XX=student ID number)
- Write the basic structure of the Java programming language which contains the **main()** function
- Add the Scanner library and make a **Scanner** declaration for input purposes
- Create an array of integer types with the name **score** with a capacity of 10.

Then declare the variables total and average

```

int[] score = new int[10];
double total = 0;
double average;

```



-
5. Using a loop, create an input to fill in the **score** array element

```
for (int i = 0; i < score.length; i++) {  
    System.out.print("Enter student score " + (i + 1) + ": ");  
    score[i] = sc.nextInt();  
}
```

6. Using a loop, calculate the total number of scores.

```
for (int i = 0; i < score.length; i++) {  
    total += score[i];  
}
```

7. Calculate the average value by dividing **total** by the number of elements of **score**

```
average = total / score.length;  
System.out.println("The class average score is " + average);
```

8. Compile and run the program. Match the results of the running programs that you have created according to the following display

```
Enter student score 1: 98  
Enter student score 2: 73  
Enter student score 3: 86  
Enter student score 4: 82  
Enter student score 5: 95  
Enter student score 6: 68  
Enter student score 7: 90  
Enter student score 8: 71  
Enter student score 9: 78  
Enter student score 10: 84  
The class average score is 82.5
```

9. Commit and push the changes to GitHub



```

public class arrayAverageScore07 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] score = new int[10];
        double total = 0;
        double average;
        for (int i = 0; i < score.length; i++) {
            System.out.print("Enter student score " + (i + 1) + ": ");
            score[i] = sc.nextInt();
        }
        for (int i = 0; i < score.length; i++) {
            total += score[i];
        }
        average = total / score.length;
        System.out.println("The class average score is " + average);
        sc.close();
    }
}

```

The screenshot shows the Java code running in the terminal. The output is:

```

Enter student score 7: 89
Enter student score 8: 89
Enter student score 9: 90
Enter student score 10: 88
The class average score is 83.4

```

Questions!

1. Modify the program in Experiment 3 so that the program can display the number of students who passed, students who have a score greater than 70 (>70)

```

public class arrayAverageScore07 {
    public static void main(String[] args) {
        int[] score = new int[10];
        double total = 0;
        double average;
        int passedCount = 0;

        // Step 5: Using a loop, create an input to fill in the scores
        for (int i = 0; i < score.length; i++) {
            System.out.print("Enter student score " + (i + 1) + ": ");
            score[i] = sc.nextInt();
        }

        // Step 6: Using a loop, calculate the total number of scores
        for (int i = 0; i < score.length; i++) {
            total += score[i];
        }

        // Count students who passed (score > 70)
        for (int i = 0; i < score.length; i++) {
            if (score[i] > 70) {
                passedCount++;
            }
        }
    }
}

```

The screenshot shows the modified Java code running in the terminal. The output is:

```

Enter student score 8: 88
Enter student score 9: 90
Enter student score 10: 84
The class average score is 84.1
Number of students who passed (score > 70): 9

```



2. Modify the program in Experiment 3 so that it can produce output like the following display

```
Enter the number of students: 5
Enter the final score 0: 81
Enter the final score 1: 76
Enter the final score 2: 90
Enter the final score 3: 68
Enter the final score 4: 63
The average score of students who passed is 82.33333333333333
The average score of students who failed is 65.5
```

```

PRAKTIKUMDASPRO-2 [Administrator]
File Edit Selection View Go Run ... ← → PRAKTIKUMDASPRO-2 [Administrator]
SOURCE CONTROL ... bers07.java arrayValue07.java arrayAverageScore07.java M ...
REPOSITORIES PRAKTIKUMDAS main* ✓ ...
CHANGES Message (Ctrl+Enter to commit on "...") ✓ Commit 1 Changes arrayAverageScore07.java jobsheet9
GRAPH modify1 rizkydava1233-ops main
modify2 rizkydava1233-ops
modify 1 rizkydava1233-ops
percobaan2 rizkydava1233-ops
modify1 rizkydava1233-ops
modify2 rizkydava1233-ops
jobsheet rizkydava1233-ops
assignment 5 rizkydava1233-ops
modify rizkydava1233-ops
tugas rizkydava1233-ops
tugas rizkydava1233-ops
tugas rizkydava1233-ops
Assignment 4 rizkydava1233-ops
Assignment 3 rizkydava1233-ops
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS Run: arrayAverageScore07 + ...
arrayAverageScore07.java -XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Users\ASUS\ROG\AppData\Roaming\Code\User\workspaceStorage\24fa80fc21927ae4156d2f9ab12e0554\redhat-.java\jdt_ws\PRAKTIKUMDASPRO-2_68585b5b\bin" "arrayAverageScore07"
am Files\Java\jdk-24\bin\java.exe
Enter the final score 2: 68
Enter the final score 3: 66
Enter the final score 4: 88
The average score of students who passed is 82.0
The average score of students who failed is 67.0
PS C:\Users\ASUS\ROG\Documents\PRAKTIKUMDASPRO\PRAKTIKUMDASPRO-2>

```

3. Commit and push the changes to GitHub

2.4 Experiment 4: Searching

Experiment Time: 45 minutes

- Open a text editor, create a Java file then save it with the name

linearSearchXX. (XX=student ID number)

- Add the following code



```

6     public static void main(String[] args) {
7         int[] arrayInt = { 34, 18, 26, 48, 72, 20, 56, 63 };
8         int key = 20;
9         int result = 0;
10        for (int i = 0; i < arrayInt.length; i++) {
11            if (arrayInt[i] == key) {
12                result = i;
13                break;
14            }
15        }
16        System.out.println("The key in the array is located at index position " + result);
17    }
18 }
```

3. Compile and run the program. Match the results of the running programs that you have created according to the following display

The key in the array is located at index position 5

The screenshot shows the Visual Studio Code (VS Code) interface. On the left, the Explorer sidebar lists several Java files in the 'PRAKTIKUMDASPRO-2' workspace, including 'linearSearch07.java'. The 'linearSearch07.java' file is open in the center editor, displaying the provided Java code for a linear search algorithm. To the right of the editor is the 'Terminal' panel, which shows the command-line output of running the program. The output indicates that the key '20' was found at index position 5. A tooltip or status message above the terminal says 'The key in the array is located at index position 5'. The bottom status bar shows the current date and time as 04/11/2025.

4. Commit and push the changes to GitHub

Questions!

- Explain the meaning of the **break;** statement on line 13 of the program code in Experiment 4.

The **break;** statement makes the linear search algorithm efficient by stopping the search as soon as the target value is found, rather than continuing through the entire array unnecessarily.



Think of it like looking for your keys: once you find them on the table, you don't keep searching in the drawers!

2. Modify the program code in experiment 4 so that the program can receive input in the form of the number of array elements, the contents of the array, and the key you want to search for. Then, print to the screen the index of the element positions of the searched key. Example of program results:

```
Enter the number of array elements: 8
Enter the array element 0: 12
Enter the array element 1: 18
Enter the array element 2: -6
Enter the array element 3: 10
Enter the array element 4: 6
Enter the array element 5: 15
Enter the array element 6: 11
Enter the array element 7: 9
Enter the key you want to search for: 10
The key in the array is located at index position 3
```

The screenshot shows the GitHub Copilot interface with a Java file named `linearSearch07.java` open. The code implements a linear search algorithm. A tooltip from GitHub Copilot suggests upgrading to Copilot Pro due to reaching monthly message quota.

```
public class linearSearch {
    public static void main(String[] args) {
        // Tampilkan hasil
        if (result != -1) {
            System.out.println("The key in the array is located at index position " + result);
        } else {
            System.out.println("Key not found in the array");
        }
        sc.close();
    }
}
```

The terminal window shows the execution of the program with user input and output matching the example above.

3. Modify the program in experiment 4 so that the program will give the message "key not found" if the key is not in the array. Example of program results:



```
Enter the number of array elements: 6
Enter the array element 0: 19
Enter the array element 1: 23
Enter the array element 2: 29
Enter the array element 3: 31
Enter the array element 4: 37
Enter the array element 5: 43
Enter the key you want to search for: 11
Key not found
```

The screenshot shows a GitHub Desktop application window. On the left, the repository structure is visible, including branches like 'main' and 'origin/main'. The central part of the window displays the code for 'linearSearch07.java'. The right side shows the GitHub interface with a message about workspace resolution, a workspace summary, and a notification about reaching monthly chat messages quota.

```
public class linearSearch07 {
    public static void main(String[] args) {
        // Tampilkan hasil
        if (result != -1) {
            System.out.println("The key in the array is located at index " + result);
        } else {
            System.out.println(x + ": Key not found in the array");
        }
        sc.close();
    }
}
```

Assignment

1. You are asked to create a program that can store and manage student grades.

The grades are integers. The program must provide features for:

- entering the number of student grades to be entered,
- entering each student's grade,
- calculating the average grade,
- displaying the highest and lowest grades, and
- displaying all grades entered.



```

1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.Scanner;
4
5 public class StudentGradeManager07 {
6     Run | Debug
7     public static void main(String[] args) {
8         Scanner sc = new Scanner(System.in);
9         System.out.println("===== STUDENT GRADE MANAGEMENT SYSTEM =====");
10        // Enter the number of students
11        int numStudents = 0;
12        while (true) {
13            try {
14                System.out.print("Enter the number of students: ");
15                numStudents = Integer.parseInt(sc.nextLine());
16                if (numStudents > 0) {
17                    break;
18                }
19            } catch (Exception e) {
20                System.out.println("Please enter a valid integer value.");
21            }
22        }
23        int totalGrade = 0;
24        int highestGrade = 0;
25        int lowestGrade = 100;
26        for (int i = 1; i <= numStudents; i++) {
27            System.out.print("Enter grade " + i + ": ");
28            int grade = Integer.parseInt(sc.nextLine());
29            totalGrade += grade;
30            if (grade > highestGrade) {
31                highestGrade = grade;
32            }
33            if (grade < lowestGrade) {
34                lowestGrade = grade;
35            }
36        }
37        System.out.println("Total number of students: " + numStudents);
38        System.out.println("Average grade: " + (totalGrade / numStudents));
39        System.out.println("Highest grade: " + highestGrade);
40        System.out.println("Lowest grade: " + lowestGrade);
41        System.out.println("All grades entered:");
42        for (int i = 1; i <= numStudents; i++) {
43            System.out.println(i + ". " + grade);
44        }
45    }
46 }

```

The screenshot shows a Java development environment with the following details:

- Code Editor:** Displays `StudentGradeManager07.java` with code for managing student grades.
- Terminal:** Shows the execution of the program and its output. The output includes the total number of students (5), average grade (77.20), highest grade (90), lowest grade (68), and a list of all entered grades.
- Graph Browser:** On the left, it shows a dependency graph with nodes like `main`, `modify1`, `percobaan4`, etc., and their relationships.
- IDE UI:** Includes tabs for File, Edit, Selection, View, Go, Run, etc., and various status indicators at the bottom.

2. Create a program that can manage food and beverage orders at a cafe. The program will allow users to enter orders, calculate the total cost of the order, and display a list of the orders that have been placed.

- Input:
 - number of orders (input by the user).
 - name of the food/drink and the price for each order (input by the user).
- Process:
 - store the order data in a one-dimensional array for the order names; and a separate one-dimensional array for the prices.
 - calculate the total cost of all orders entered.
 - display a list of orders that have been entered along with the total cost.
- Output:
 - list of orders and the total cost of all orders.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the current workspace, including `CafeOrderManagement07.java`, `StudentGradeManager07.java`, `arrayNumbers07.java`, `arrayValue07.java`, `arrayAverageScore07.java`, and `linearSearch07.java`.
- Editor:** Displays the `CafeOrderManagement07.java` file with code for reading order details from the console.
- Terminal:** Shows the command run in the terminal:

```
PS C:\Users\ASUS ROG\Documents\PRAKTIKUMDASPRO\PRAKTIKUMDASPRO-2> & 'C:\Program Files\Java\jdk-24\bin\java.exe' -XX:+ShowCodeDetailsInExceptionMessages' -cp 'C:\Users\ASUS ROG\AppData\Roaming\Code\User\workspaceStorage\24fa80fc21927ae4156d2f9ab12e0554\redhat\data\jdt_ws\PRAKTIKUMDASPRO-2_68585bb\bin'\Assignment.CafeOrderManagement07'
```
- Output:** The terminal output shows the program's response:

```
List of Orders:
-----
No. Item Price
-----
1 seblak $2,00
2 basreng $30,00
3 udang keju $20,00

TOTAL COST: $52,00
```
- Chat:** A sidebar message indicates "You've reached your monthly chat messages quota. Upgrade to Copilot Pro (30-day free trial) or wait for your allowance to renew." It also shows a link to "Upgrade to GitHub Copilot Pro".

3. Continuing with the example of ordering food at a cafe, create a program that allows users to order food from the cafe's menu. The program must store a list of food items in an array and provide an option to search for the desired item using a linear search method.

- Input:

- a predefined menu item in array form. The item names are initialized during the array declaration. For example:

```
String[] menu = {"Fried Rice", "Fried Noodles", "Toasted Bread", "Fried Potatoes", "Teh Tarik", "Cappuccino", "Chocolate Ice"};
```

- the name of the item to be searched for (user input).

- Process:

- the program searches for the item entered by the user using a linear search algorithm.
 - if the item is found, the program informs the user that it is available. If not, the program informs the user that the item is not on the menu.

- Output:

- display the search results to the user.



The screenshot shows a Java development environment with the following details:

- File Explorer:** Shows various Java files including `arrayNumbers07.java`, `arrayValue07.java`, `arrayAverageScore07.java`, `linearSearch07.java`, `StudentGradeManager07.java`, `CafeOrderManagement07.java`, and `CafeMenuSearch07.java`.
- Code Editor:** Displays the `CafeMenuSearch07.java` file with the following code:import java.util.Scanner;
public class CafeMenuSearch07 {
 public static int linearSearch(String[] menu, String searchItem) {
 for (int i = 0; i < menu.length; i++) {
 // Case-insensitive comparison
 if (menu[i].equalsIgnoreCase(searchItem)) {
 return i; // Return index if found
 }
 }
 return -1; // Return -1 if not found
 }
}
- Terminal:** Shows the command-line interface output:PS C:\Users\ASUS ROG\Documents\PRAKTIKUMDASPRO\PRAKTIKUMDASPRO-2> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' -cp 'C:/Users/ASUS ROG/AppData/Roaming/Code/User/workspaceStorage/24fa80fc21927ae4156d2f9ab12e0554/redhat.java/jdt_ws/PRAKTIKUMDASPRO-2_68585b/bin' Assignment.CafeMenuSearch07'
Enter the food/drink item to search for (or 'exit' to quit): teh tarik

SEARCH RESULT

? ITEM FOUND!
"Teh Tarik" is available on our menu.
Position in menu: 5
- Right Panel:** Includes a GitHub Copilot Pro upgrade notice, workspace usage information, and a GitHub Copilot status bar.