Southeast University

Department of Computer Science and Engineering (CSE)

School of Sciences and Engineering

Semester: (Summer, Year: 2025)

LAB REPORT NO: **06**

Course Title: Introduction to Programming Language II (Java) Lab

Course Code: CSE282.2

Batch: 65

Lab Experiment Name: Introducing one/two-dimensional Array, arraylist & linklist in JAVA.

Student Details

	Name	ID
1.	Md. Mahmud Hossain	2023200000799

Submission Date : **28-08-25**

Course Teacher's Name : **Dr. Mohammed Ashikur Rahman**

Lab Report Status

Marks: ………………………………… **Signature:.....................**

Comments:... **Date:.............................**

Lab Task 6: Introducing one/two-dimensional Array, arraylist & linklist in JAVA.

PROBLEM:

1. Write a program to sort an array using bubble sort, selection sort and merge sort.
2. Write a program to multiply two matrices.
3. Solve problem 4 by using ArrayList, LinkedList and their various methods
4. Solve practice problems from Lab Sheet #3 using array, ArrayList and LinkedList.

Solution:

1.

Problem Analysis:

This program demonstrates three basic sorting algorithms in Java: Bubble Sort, Selection Sort, and Merge Sort. Each algorithm follows a different approach—Bubble Sort repeatedly swaps adjacent elements, Selection Sort finds the smallest element and places it in order, while Merge Sort uses divide-and-conquer to recursively sort and merge subarrays. The main() method applies all three algorithms on the same input array and displays their results for comparison.

Background Theory:

Sorting Algorithms: Sorting is the process of arranging data in a specific order, usually ascending or descending. It improves efficiency in searching and organizing data.

Bubble Sort: A simple comparison-based algorithm that repeatedly swaps adjacent elements if they are in the wrong order. It is easy to implement but inefficient for large datasets with time complexity $O(n^2)$.

Selection Sort: This algorithm repeatedly selects the smallest element from the unsorted portion of the array and places it in the correct position. It has time complexity $O(n^2)$ but performs fewer swaps than bubble sort.

Merge Sort: An efficient divide-and-conquer algorithm that splits the array into halves, recursively sorts them, and merges the sorted halves. It has a better time complexity of $O(n \log n)$, making it suitable for large datasets.

Arrays Class: The program uses the Arrays.toString() method from the Java util package to print arrays in a readable format.

main() Method: Demonstrates the functionality by applying all three sorting algorithms on the same input array and printing the results.

Algorithm Design:

1. Define a bubbleSort() method:

   - Loop through the array multiple times.
   - Compare each pair of adjacent elements.
   - Swap them if they are out of order.

2. Define a selectionSort() method:

   - Iterate through the array.
   - Find the smallest element in the unsorted portion.
   - Swap it with the first unsorted element.

3. Define a mergeSort() method:

   - Recursively divide the array into two halves until a single element remains.
   - Merge the two halves using a helper merge() method that arranges elements in sorted order.

4. In the main() method:

   - Create an array of integers.
   - Clone it for each sorting algorithm to ensure fairness in testing.
   - Apply Bubble Sort, Selection Sort, and Merge Sort separately.
   - Print the original array and the results after each sorting algorithm.

Code:

```
import java.util.Arrays;
public class SortingAlgorithms{
   public static void bubbleSort(int[] arr) {
      int n = arr.length;
      for(int i = 0; i < n-1; i++) {
         for(int j = 0; j < n-i-1; j++) {
            if(arr[j] > arr[j+1]) {
```

```
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}

public static void selectionSort(int[] arr) {
    int n = arr.length;
    for(int i = 0; i < n-1; i++) {
        int minIndex = i;
        for(int j = i+1; j < n; j++) {
            if(arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }

        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}

public static void mergeSort(int[] arr, int left, int right) {
    if(left < right) {
        int mid = (left + right) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid+1, right);
        merge(arr, left, mid, right);
    }
}

public static void merge(int[] arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int[] L = new int[n1];
    int[] R = new int[n2];

    for(int i=0; i<n1; i++) L[i] = arr[left + i];
    for(int j=0; j<n2; j++) R[j] = arr[mid + 1 + j];

    int i=0, j=0, k=left;
```

```
        while(i<n1 && j<n2) {
            if(L[i] <= R[j]) {
                arr[k++] = L[i++];
            } else {
                arr[k++] = R[j++];
            }
        }
        while(i<n1) arr[k++] = L[i++];
        while(j<n2) arr[k++] = R[j++];
    }

    public static void main(String[] args) {
        int[] arr1 = {64, 25, 12, 22, 11};
        int[] arr2 = arr1.clone();
        int[] arr3 = arr1.clone();

        System.out.println("Sample Input:"+Arrays.toString(arr1));
        bubbleSort(arr1);
        System.out.println("Bubble Sort: " + Arrays.toString(arr1));

        selectionSort(arr2);
        System.out.println("Selection Sort: " + Arrays.toString(arr2));

        mergeSort(arr3, 0, arr3.length-1);
        System.out.println("Merge Sort: " + Arrays.toString(arr3));
    }
}
```

Output:

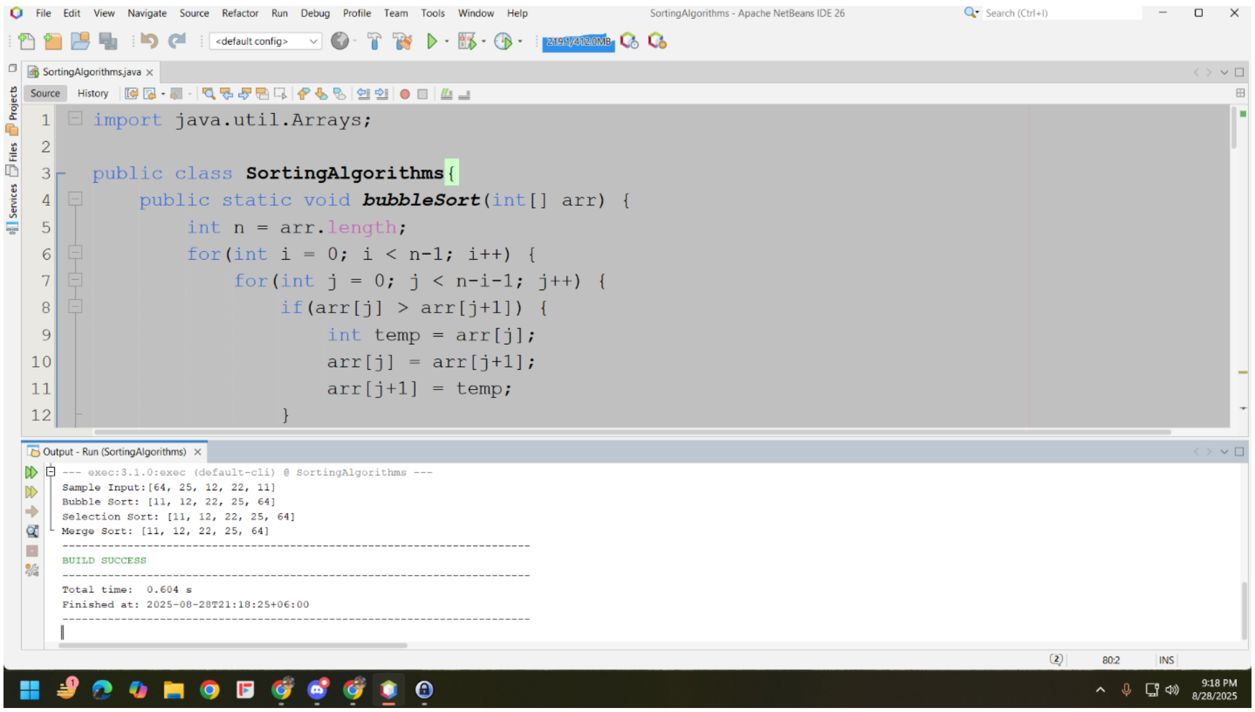

Figure 1: Output

2.

Problem Analysis:

This program performs matrix multiplication between two matrices A and B. The multiplication is valid only if the number of columns in the first matrix equals the number of rows in the second matrix. The program checks this condition, computes the product using nested loops, and stores the result in a new matrix C. Finally, it prints the resulting matrix in row–column format.

Background Theory:

Matrix Multiplication: If matrix A has dimensions m × n and matrix B has dimensions n × p, then the product matrix C will have dimensions m × p. Each element C[i][j] is calculated by multiplying elements of the i-th row of A with the j-th column of B and summing them up.

Validity Condition: For multiplication to be possible, the number of columns in the first matrix (A) must be equal to the number of rows in the second matrix (B).

Nested Loops: The program uses three nested loops—

- Outer loop for iterating through rows of A.
- Middle loop for iterating through columns of B.
- Inner loop for computing the sum of products for each element of the resulting matrix C.

main() Method: Declares two matrices A and B, checks if multiplication is possible, computes the result, and prints the product matrix.

Algorithm Design:

1. Initialize two matrices A and B with predefined values.
2. Determine the number of rows and columns of both matrices.
3. Check if multiplication is possible (colsA == rowsB). If not, print an error message and exit.
4. Create a new matrix C with dimensions rowsA × colsB to store the result.
5. Use three nested loops:

   - Outer loop: iterate over rows of A.
   - Middle loop: iterate over columns of B.
   - Inner loop: compute the sum of products for the current element.

6. Store each computed value in C[i][j].
7. Print the resulting matrix C.

Code:

```
public class MatrixMultiplication {
    public static void main(String[] args) {
        int[][] A = {
            {1, 2, 3},
            {4, 5, 6}
        };

        int[][] B = {
            {7, 8},
            {9, 10},
            {11, 12}
        };
```

```
        int rowsA = A.length;
        int colsA = A[0].length;
        int rowsB = B.length;
        int colsB = B[0].length;

        if (colsA != rowsB) {
            System.out.println("Matrix multiplication not possible!");
            return;
        }

        int[][] C = new int[rowsA][colsB];

        for (int i = 0; i < rowsA; i++) {
            for (int j = 0; j < colsB; j++) {
                for (int k = 0; k < colsA; k++) {
                    C[i][j] += A[i][k] * B[k][j];
                }
            }
        }

        System.out.println("Result of Matrix Multiplication:");
        for (int i = 0; i < rowsA; i++) {
            for (int j = 0; j < colsB; j++) {
                System.out.print(C[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Output:

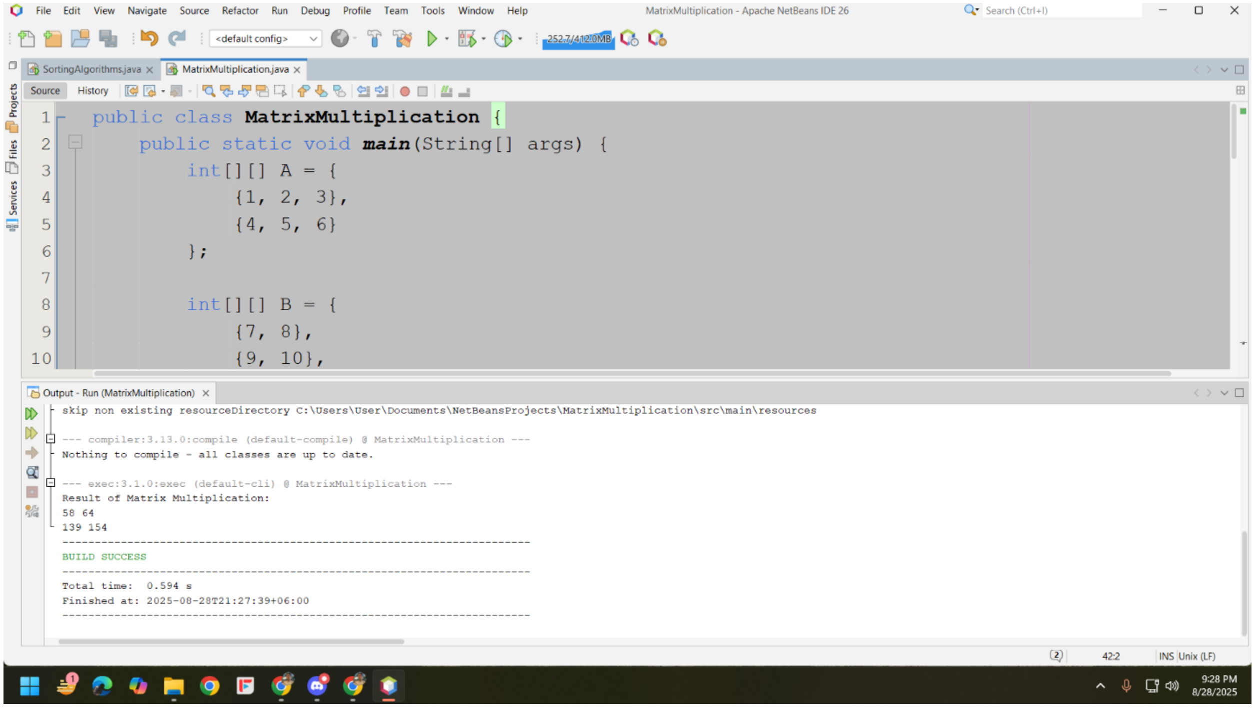

Figure 2: Output

3.

Problem Analysis:

This program demonstrates the use of Java collections (ArrayList and LinkedList) to store and manage objects of the Student class. Each student has attributes such as ID, name, department, and CGPA. The program creates a list of students using both ArrayList and LinkedList, displays their details, and also showcases special methods of LinkedList like addFirst() and addLast(). This highlights how collection classes can be used effectively for storing and manipulating object data.

Background Theory:

ArrayList: A resizable array implementation of the List interface in Java. It allows dynamic storage of elements, provides indexed access, and is efficient for search and traversal operations.

LinkedList: A doubly linked list implementation of the List interface. It is efficient for insertion and deletion at both ends and provides methods such as addFirst() and addLast() that are not present in ArrayList.

main() Method: Creates ArrayList and LinkedList objects, inserts Student objects, and displays them to demonstrate the functionality of both collection classes.

Algorithm Design:

1. Initialize two matrices A and B with predefined values.
2. Determine the number of rows and columns of both matrices.
3. Check if multiplication is possible (colsA == rowsB). If not, print an error message and exit.
4. Create a new matrix C with dimensions rowsA × colsB to store the result.
5. Use three nested loops:
    - Outer loop: iterate over rows of A.
    - Middle loop: iterate over columns of B.
    - Inner loop: compute the sum of products for the current element.
6. Store each computed value in C[i][j].
7. Print the resulting matrix C.

Code:

```
import java.util.*;

class Student {
    int id;
    String name;
    String department;
    double cgpa;

    Student(int id, String name, String department, double cgpa) {
        this.id = id;
        this.name = name;
        this.department = department;
        this.cgpa = cgpa;
    }

    void display() {
        System.out.println("ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Department: " + department);
        System.out.println("CGPA: " + cgpa);
```

```
        System.out.println("-------------------");
    }
}

public class StudentListExample {
    public static void main(String[] args) {

        ArrayList<Student> arrayList = new ArrayList<>();
        arrayList.add(new Student(101, "Mahmud", "CSE", 3.75));
        arrayList.add(new Student(102, "Faujul", "EEE", 3.50));

        System.out.println("ArrayList Students:");
        for (Student s : arrayList) {
            s.display();
        }

        LinkedList<Student> linkedList = new LinkedList<>();
        linkedList.add(new Student(103, "Ayesha", "BBA", 3.80));
        linkedList.add(new Student(104, "Zahid", "CSE", 3.60));

        System.out.println("LinkedList Students:");
        for (Student s : linkedList) {
            s.display();
        }

        linkedList.addFirst(new Student(105, "Ismail", "CSE", 3.90));
        linkedList.addLast(new Student(106, "Rima", "English", 3.40));

        System.out.println("After addFirst() and addLast():");
        for (Student s : linkedList) {
            s.display();
        }
    }
}
```

Output:

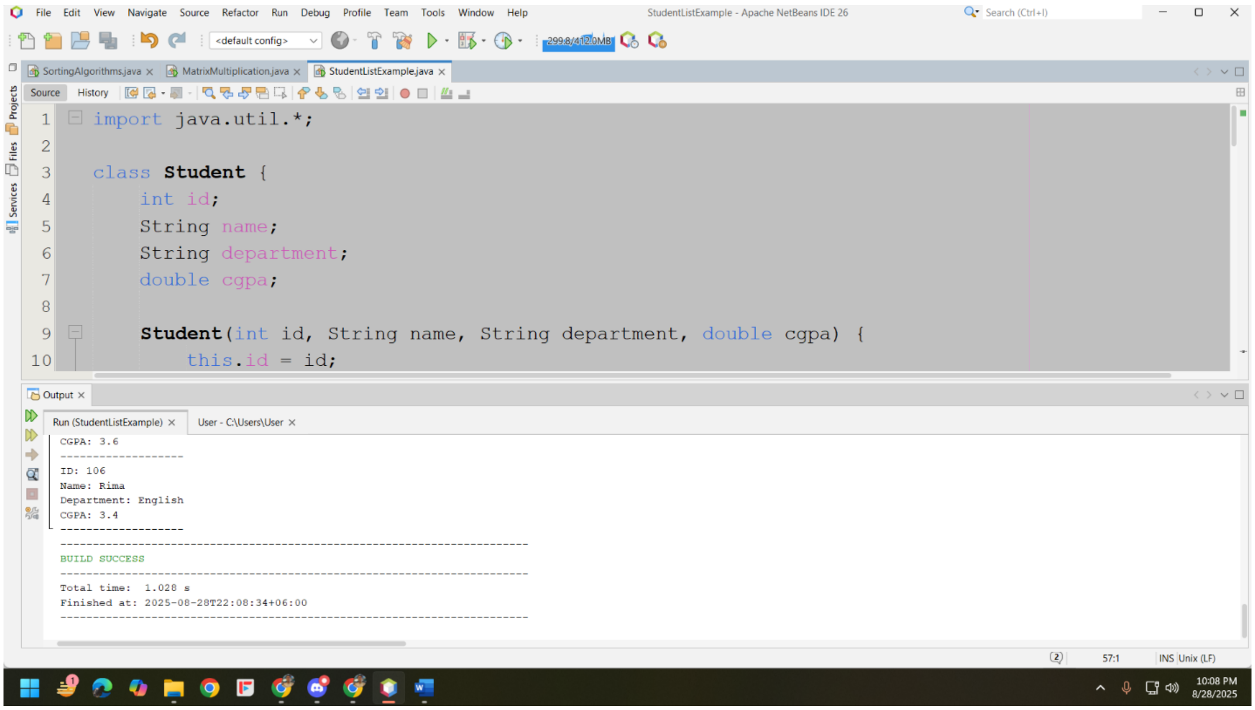

Figure 3: Output

4.

Problem Analysis:

This program demonstrates the use of arrays and Java collection classes (ArrayList and LinkedList) to store and manage objects of the Book class. Each book contains attributes such as title, author, year, and a static field genre that is shared across all objects. The program first stores books in a fixed-size array, then in an ArrayList, and finally in a LinkedList where additional operations like addFirst() are demonstrated. The goal is to show how arrays and collections can be used for organizing and handling object data in Java.

Background Theory:

Arrays: Arrays are fixed-size sequential data structures that allow fast indexed access. In this program, an array of Book objects is used to store and display book information.

ArrayList: A resizable array implementation of the List interface in Java. Unlike arrays, it grows dynamically and provides methods to add and iterate through elements conveniently.

LinkedList: A doubly linked list implementation of the List interface. It allows efficient insertion and deletion, particularly at the beginning or end, and provides methods like addFirst() to insert elements at the front.

main() Method: Demonstrates three storage approaches—using arrays, ArrayList, and LinkedList. It creates Book objects, inserts them into these data structures, and displays the stored information using loops.

Algorithm Design:

1. Define a Book class with attributes: title, author, and year.
2. Declare a static variable genre shared by all books.
3. Implement a constructor to initialize book details.
4. Define a display() method to print book information along with the genre.
5. In the main() method:

- Create an array of Book objects, initialize it with sample books, and display them.
- Create an ArrayList<Book>, add books to it, and display them.
- Create a LinkedList<Book>, add books to it, and use addFirst() to insert a book at the beginning.
- Display the final list of books stored in the LinkedList.

Code:

```
import java.util.*;

class Book {
    String title;
    String author;
    int year;
    static String genre = "Fiction";

    Book(String title, String author, int year) {
        this.title = title;
        this.author = author;
        this.year = year;
    }

    void display() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Year: " + year);
        System.out.println("Genre: " + genre);
        System.out.println("--------------------");
```

```java
    }
}

public class BookDemo {
    public static void main(String[] args) {

        Book[] arr = {
            new Book("Dune", "Frank Herbert", 1965),
            new Book("Neuromancer", "William Gibson", 1984)
        };

        System.out.println("Books using Array:");
        for (Book b : arr) {
            b.display();
        }

        ArrayList<Book> arrayList = new ArrayList<>();
        arrayList.add(new Book("Foundation", "Isaac Asimov", 1951));
        arrayList.add(new Book("1984", "George Orwell", 1949));

        System.out.println("Books using ArrayList:");
        for (Book b : arrayList) {
            b.display();
        }

        LinkedList<Book> linkedList = new LinkedList<>();
        linkedList.add(new Book("The Hobbit", "J.R.R. Tolkien", 1937));
        linkedList.add(new Book("Brave New World", "Aldous Huxley", 1932));

        linkedList.addFirst(new Book("Snow Crash", "Neal Stephenson", 1992));

        System.out.println("Books using LinkedList:");
        for (Book b : linkedList) {
            b.display();
        }
    }
}
```

Output:

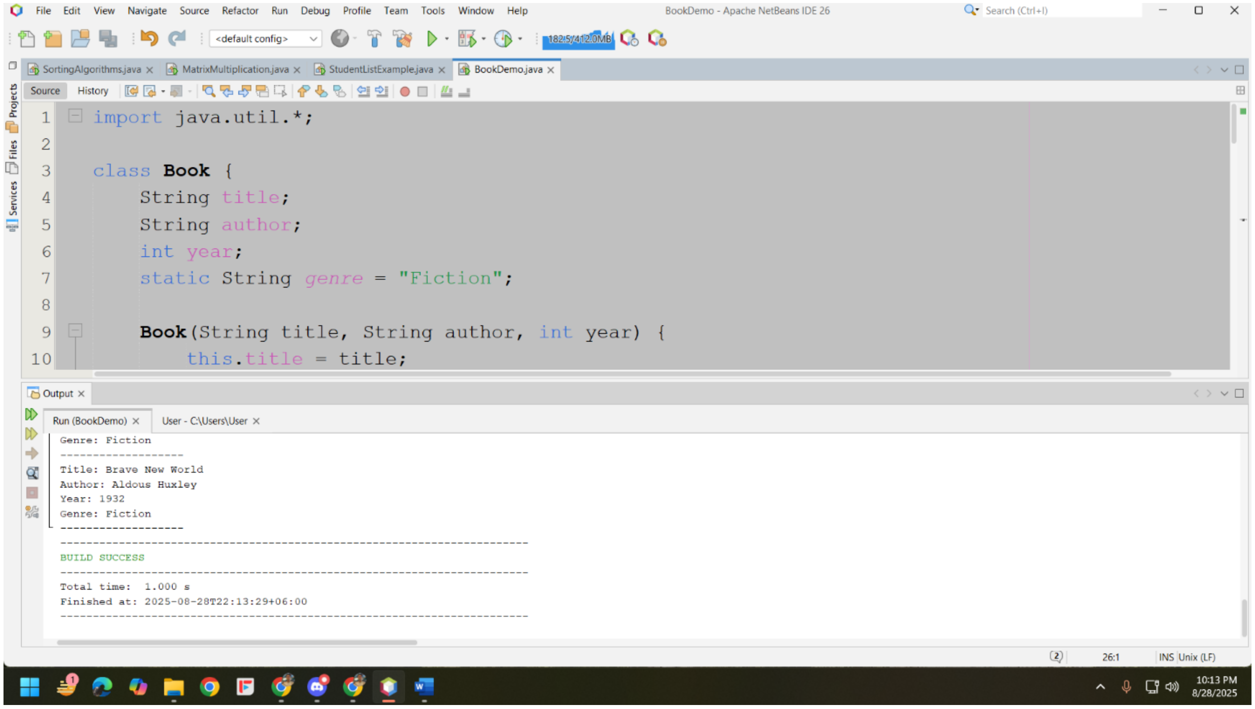

Figure 4: Output