

LAPORAN PRAKTIKUM ALGORITMA PEMOGRAMAN  
STACK PADA JAVA



OLEH:  
**SASKIA ALIFAH**  
**2411531002**

DOSEN PENGAMPU:  
DR. WAHYUDI, S.T, M.T

FAKULTAS TEKNOLOGI INFORMASI  
DEPARTEMEN INFORMATIKA  
UNIVERSITAS ANDALAS  
PADANG 2025

## A. Pendahuluan

Struktur data stack merupakan salah satu bentuk penyimpanan data linear yang mengikuti prinsip LIFO (Last In, First Out), yaitu data yang terakhir masuk akan menjadi data pertama yang keluar. Stack digunakan luas dalam pemrograman karena efisien untuk menangani proses seperti navigasi mundur, undo-redo, hingga evaluasi ekspresi matematika.

Konsep dasar yang digunakan dalam stack adalah operasi **push** dan **pop**. Operasi push berfungsi untuk menambahkan elemen ke atas tumpukan. Setiap data baru akan menempati posisi teratas, dan elemen sebelumnya akan berada di bawahnya. Push dilakukan dengan menaikkan indeks penunjuk stack sebelum menyimpan data. Sebaliknya, **pop** digunakan untuk menghapus elemen paling atas dari tumpukan. Operasi ini akan mengambil elemen terakhir yang masuk, kemudian menurunkan penunjuk indeks stack agar posisi tersebut bisa diisi ulang. Pop juga biasa digunakan untuk "membongkar" stack satu per satu hingga kosong.

Selain push dan pop, ada juga operasi **peek** atau **top**, yaitu melihat elemen paling atas tanpa menghapusnya. Ini berguna untuk mengecek data terakhir yang masuk, tanpa mengubah struktur stack yang ada.

Dalam stack, pengecekan kondisi juga penting, seperti `isEmpty()` yang memeriksa apakah stack kosong, dan `size()` yang menghitung berapa banyak elemen dalam stack. Fungsi-fungsi ini dibutuhkan agar program tidak melakukan pop atau top saat stack sedang kosong, yang bisa menyebabkan error.

Materi lain yang digunakan adalah **array** sebagai media penyimpanan stack secara manual. Dalam implementasi stack berbasis array, perlu diperhatikan batas kapasitas agar tidak terjadi stack overflow, yaitu saat data melebihi kapasitas maksimum yang telah ditentukan.

Selain array, materi **generic** juga digunakan agar stack dapat menyimpan berbagai tipe data seperti Integer, String, dan lainnya. Dengan penggunaan generic, struktur stack menjadi lebih fleksibel dan dapat digunakan dalam berbagai konteks program.

Terakhir, digunakan juga stack untuk menyelesaikan **evaluasi ekspresi postfix**, di mana operator berada setelah operand. Ini menunjukkan penerapan stack di dunia nyata, terutama dalam sistem pemrosesan ekspresi matematika seperti pada kalkulator atau compiler.

## B. Tujuan

Tujuan dilakukannya praktikum ini adalah sebagai berikut:

1. Memahami konsep struktur data Stack (tumpukan) dan prinsip kerja LIFO (Last In, First Out).
2. Mampu mengimplementasikan Stack menggunakan array dan generic class di Java.
3. Mampu menggunakan class bawaan `java.util.Stack` untuk operasi dasar Stack.
4. Mampu menerapkan Stack untuk menyelesaikan permasalahan seperti evaluasi postfix.
5. Mengembangkan kemampuan analisis kode dan interpretasi output program berbasis Stack.

## C. Langkah Kerja Praktikum

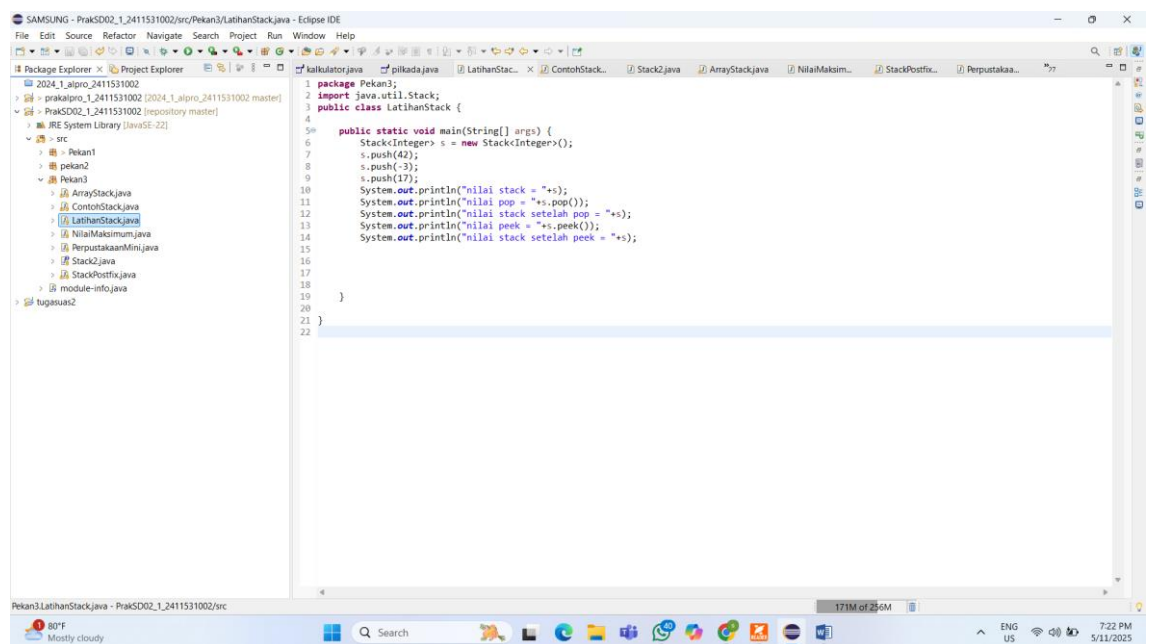
### 1. Cara kerja stack

Class LatihanStack

Perhatikan program berikut:

```
package Pekan3;
import java.util.Stack;
public class LatihanStack {

    public static void main(String[] args) {
        Stack<Integer> s = new Stack<Integer>();
        s.push(42);
        s.push(-3);
        s.push(17);
        System.out.println("nilai stack = "+s);
        System.out.println("nilai pop = "+s.pop());
        System.out.println("nilai stack setelah pop = "+s);
        System.out.println("nilai peek = "+s.peek());
        System.out.println("nilai stack setelah peek = "+s);
    }
}
```



Pada program tersebut, kita dapat menambahkan data kedalam stack dengan cara menggunakan method push(); untuk melihat isi dari stak tersebut secara keseluruhan, cukup panggil variabel stack tersebut. Jika kita ingin memanggil nilai stack paling atas dan menghapus data paling atas, dapat menggunakan

method pop(). Jika kita hanya ingin memanggil nilai stack paling atas saja, dapat menggunakan method peek();

Contoh keluarannya adalah sebagai berikut:

```
<terminated> LatihanStack [Java Application] C:\Users\SAMSUNG\
nilai stack = [42, -3, 17]
nilai pop = 17
nilai stack setelah pop = [42, -3]
nilai peek = -3
nilai stack setelah peek = [42, -3]
```

## 2. Pengenalan lebih dalam mengenai stack

### A. Class ArrayStack

```
package Pekan3;
```

```
public class ArrayStack<E> implements Stack2<E> {
```

```
    public static final int CAPACITY=1000;
```

```
    // default array capacity
```

```
    private E[] data; // generic array used for storage
```

```
    private int t=-1;
```

```
    public ArrayStack() {
```

```
        this (CAPACITY);
```

```
    } // constructs stack with default capacity
```

```
    public ArrayStack (int capacity) {
```

```
        // constructs stack with given capacity
```

```
        data=(E[]) new Object [capacity];
```

```
    }
```

```
    public int size() {
```

```
        return (t+1);
```

```
    }
```

```
    public boolean isEmpty() {
```

```
        return (t== -1);
```

```
    }
```

```
    public void push (E e) throws
```

```
        IllegalStateException {
```

```
        if (size() == data.length)
```

```
            throw new
```

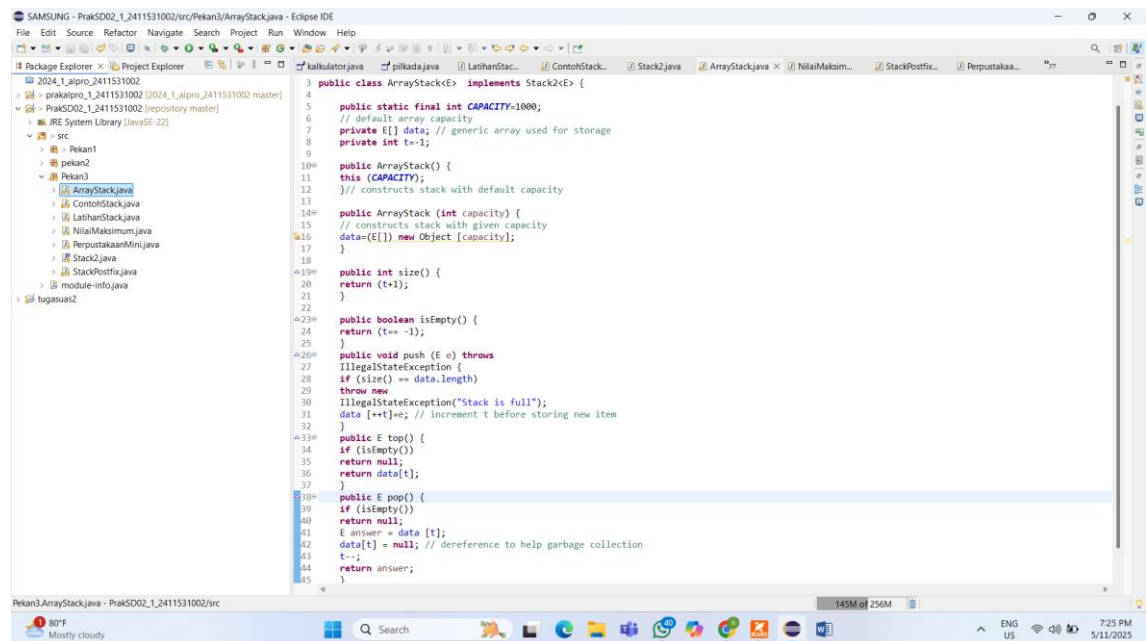
```
                IllegalStateException("Stack is full");
```

```
        data [++t]=e; // increment t before storing new item
```

```

    }
    public E top() {
        if (isEmpty())
            return null;
        return data[t];
    }
    public E pop() {
        if (isEmpty())
            return null;
        E answer = data[t];
        data[t] = null; // dereference to help garbage collection
        t--;
        return answer;
    }
}

```



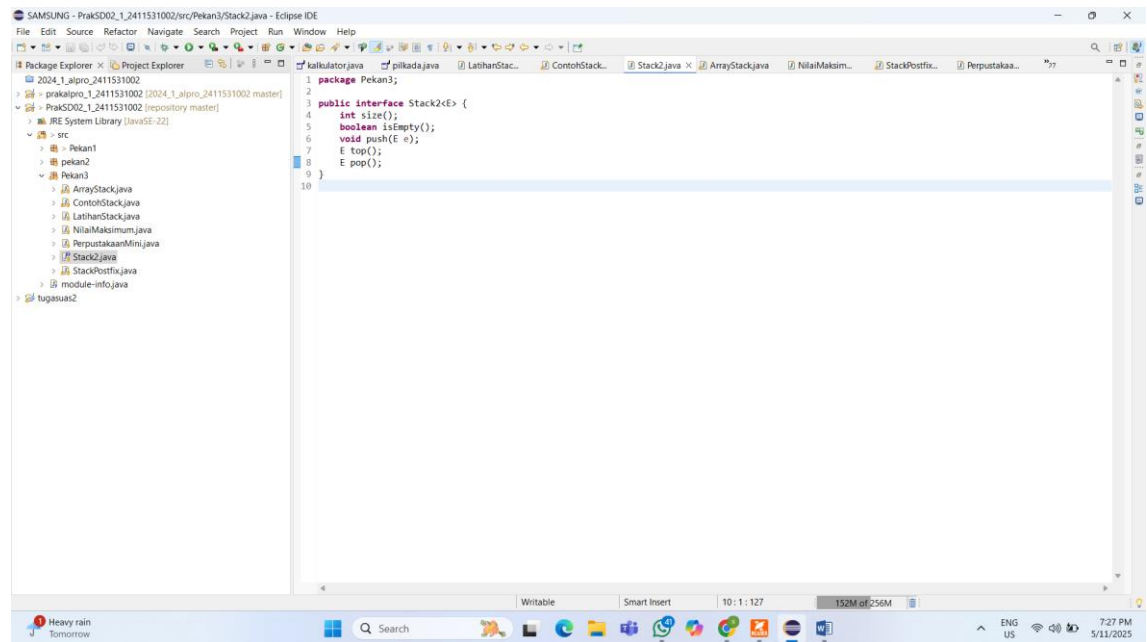
## B. Interface Stack2

**package** Pekan3;

```

public interface Stack2<E> {
    int size();
    boolean isEmpty();
    void push(E e);
    E top();
    E pop();
}

```



Program tersebut adalah rincian mengenai cara kerja stack. Kita bisa membuat sistem stack sendiri tanpa mengimport dari `java.util.Stack`;. Program tersebut bernama `ArrayStack` yang mengimplement interface `Stack2`. Cara kerja `ArrayStack` sama seperti Stak pada umumnya yang mengimplementasi menggunakan `Array`.

Berikut rincian dari setiap method:

- “`public static final int CAPACITY = 1000;`”: berisikan batas maksimal element yang dapat stack tersebut simpan, yaitu 1000 buah.
- “`private E[] data;`”: pada program ini, dasar stacknya yaitu array yang direpresentasikan oleh variabel `data`. Jadi, `data` yang masuk akan tersimpan pada variabel ini.
- “`private int t = -1;`”: method ini berisikan tentang informasi jumlah index pada stack. `-1` menunjukkan bahwa stack tersebut kosong.
- `public ArrayStack()`: Merupakan default constructor dari class `ArrayStack`, yang mana memiliki kapasitas default 1000
- `public ArrayStack(int capacity)`: Merupakan overloaded constructor yang mana jika dipanggil, akan digunakan kapasitas sesuai input yang diberikan.
- `public int size()`: Merupakan method untuk mengeluarkan ukuran dari stack tersebut. Cara method ini melihat ukuran stack yaitu dengan mereturn `t+1`
- `public boolean isEmpty()`: Method ini mengecek apakah `t = -1` yang berarti stack kosong. Jika stack kosong, akan bernilai `true`. Jika stack tidak `-1`, maka bernilai `false`.
- `public void push(E e) throws IllegalStateException`: Method ini memasukkan data kedalam stack. Method ini juga melihat kapasitas yang

disediakan saat pendeklarasian. Jika method dipanggil lebih dari kapasitas, akan mengeluarkan "Stack is full".

- public E top(): Method ini akan mengeluarkan data teratas dari stack. Jika stack kosong, bernilai null
- public E pop(): Method ini mengeluarkan data teratas dan menghapus data teratas di stack. Jika stack kosong, return null. Method ini juga akan mengurangi nilai t.

### 3. Implementasi Stack

#### A. Class ContohStack

Dari program ArrayStack diatas, kita dapat membuat contoh program seperti berikut:

```
package Pekan3;
```

```
import java.util.*;
```

```
public class ContohStack {
```

```
    public static void main(String[] args) {
```

```
        ArrayStack test = new ArrayStack();
```

```
        Integer[] a= {4, 8, 15, 16, 23, 42}; //autoboxing allows this
```

```
        for (int i = 0; i < a.length; i++) {
```

```
            System.out.println("nilai A "+i+" = "+ a[i]);
```

```
            test.push(a[i]);
```

```
        }
```

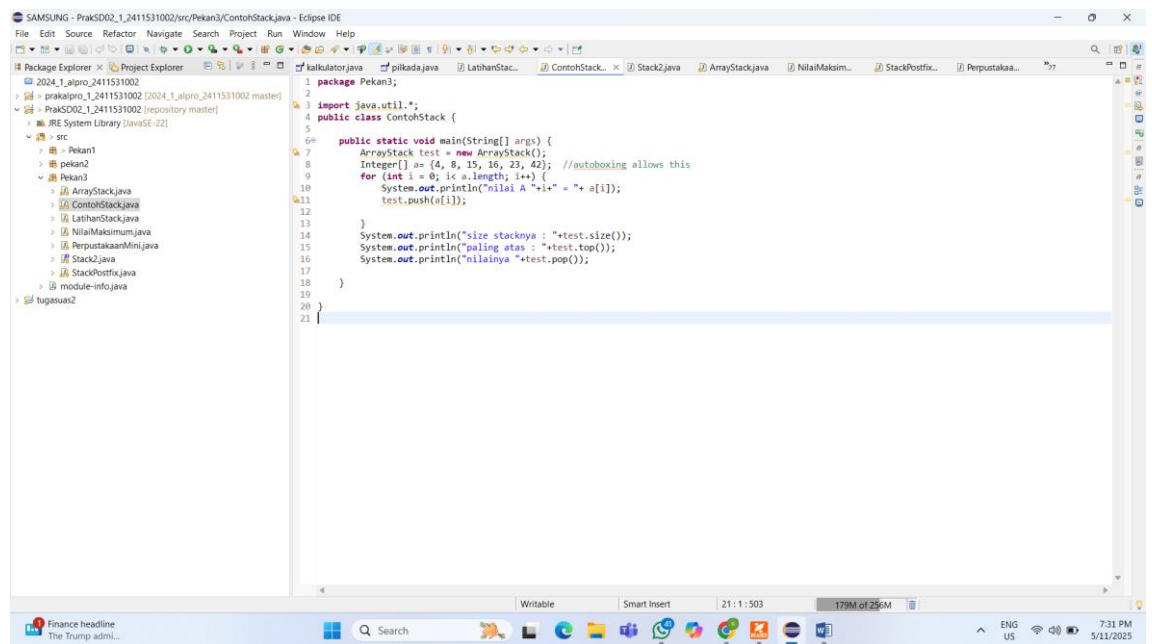
```
        System.out.println("size stacknya : "+test.size());
```

```
        System.out.println("paling atas : "+test.top());
```

```
        System.out.println("nilainya "+test.pop());
```

```
    }
```

```
}
```



Alih-alih menggunakan stack, kita bisa menggunakan class yang telah kita buat sebelumnya yaitu ArrayStack. Untuk sistemnya sama seperti stack pada umumnya, berikut contoh keluarannya:

<terminated> ContohStack [Java Application]

```

nilai A 0 = 4
nilai A 1 = 8
nilai A 2 = 15
nilai A 3 = 16
nilai A 4 = 23
nilai A 5 = 42
size stacknya : 6
paling atas : 42
nilainya 42

```

Berikut contoh lainnya dalam implementasi stack:

## B. Class NilaiMaksimum

```

package Pekan3;
import java.util.*;
public class NilaiMaksimum {
    public static int max (Stack<Integer> s) {
        Stack<Integer> backup=new Stack<Integer>();
        int maxValue=s.pop();
        backup.push (maxValue);
        while (!s.isEmpty()) {
            int next=s.pop();

```



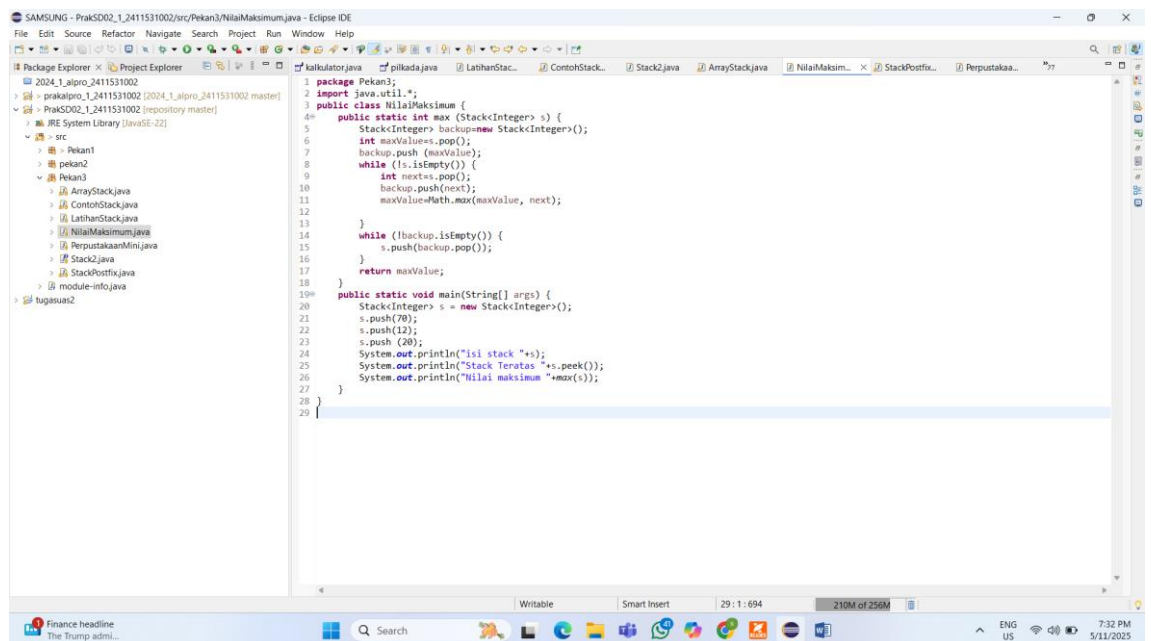
```

        backup.push(next);
        maxValue=Math.max(maxValue, next);

    }
    while (!backup.isEmpty()) {
        s.push(backup.pop());
    }
    return maxValue;
}

public static void main(String[] args) {
    Stack<Integer> s = new Stack<Integer>();
    s.push(70);
    s.push(12);
    s.push(20);
    System.out.println("isi stack "+s);
    System.out.println("Stack Teratas "+s.peek());
    System.out.println("Nilai maksimum "+max(s));
}
}

```



Dari program tersebut, kita membuat 2 stack, yaitu stack utama pada method main, dan stack backup pada method max. Program ini berfungsi untuk mencari nilai maksimum dari sebuah stack. Cara kerjanya yaitu stack akan dimasukkan ke method max(), lalu stack yang telah dimasukkan di pop() dan dimasukkan ke variabel max**Value**. Kita juga membuat variabel kedua yaitu variabel next untuk memasukkan data kedua sebagai perbandingan dari nilai maksimal.

Contoh keluarannya adalah sebagai berikut:

```
<terminated> NilaiMaksimum [Java Application]
isi stack [70, 12, 20]
Stack Teratas 20
Nilai maksimum 70
```

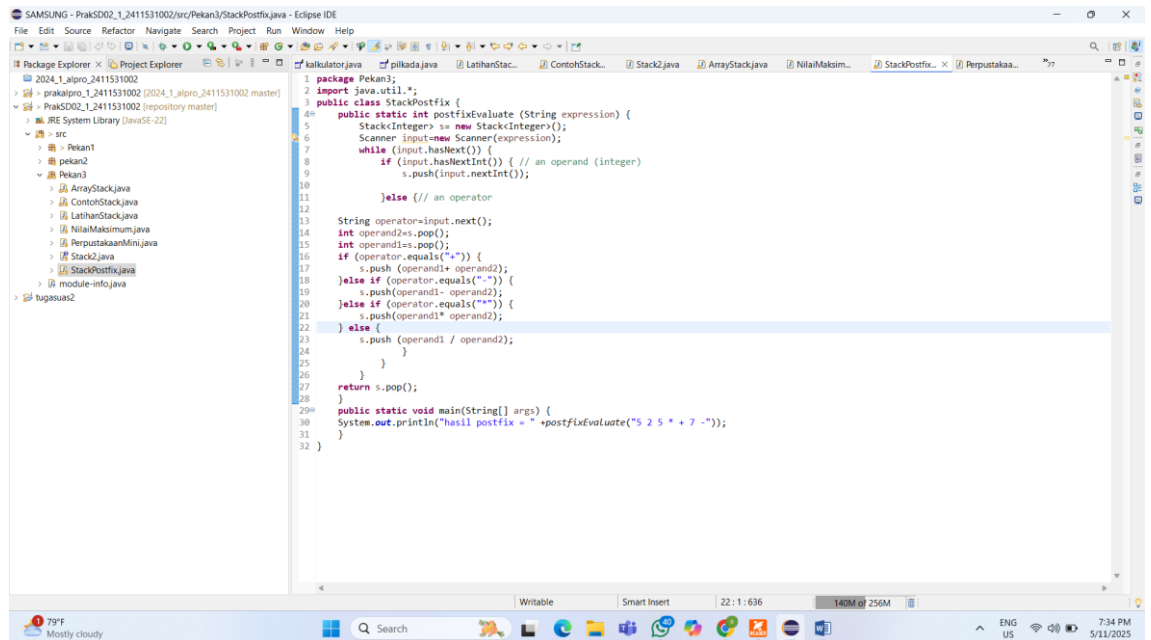
Berikut contoh lainnya dalam penggunaan stack :

### C. Stack postfix

```
package Pekan3;
import java.util.*;
public class StackPostfix {
    public static int postfixEvaluate (String expression) {
        Stack<Integer> s= new Stack<Integer>();
        Scanner input=new Scanner(expression);
        while (input.hasNext()) {
            if (input.hasNextInt()) { // an operand (integer)
                s.push(input.nextInt());

            } else { // an operator

                String operator=input.next();
                int operand2=s.pop();
                int operand1=s.pop();
                if (operator.equals("+")) {
                    s.push (operand1+ operand2);
                } else if (operator.equals("-")) {
                    s.push(operand1- operand2);
                } else if (operator.equals("*")) {
                    s.push(operand1* operand2);
                } else {
                    s.push (operand1 / operand2);
                }
            }
        }
        return s.pop();
    }
    public static void main(String[] args) {
        System.out.println("hasil postfix = " +postfixEvaluate("5 2 5 * +
7 -"));
    }
}
```



```
1 package Pekan3;
2 import java.util.*;
3 public class StackPostfix {
4     public static int postfixEvaluate(String expression) {
5         Stack<Integer> s = new Stack<Integer>();
6         Scanner input = new Scanner(expression);
7         while (input.hasNext()) {
8             if (input.hasNextInt()) { // an operand (integer)
9                 s.push(input.nextInt());
10            }
11            else { // an operator
12                String operator = input.next();
13                int operand2 = s.pop();
14                int operand1 = s.pop();
15                if (operator.equals("+")) {
16                    s.push(operand1 + operand2);
17                }
18                else if (operator.equals("-")) {
19                    s.push(operand1 - operand2);
20                }
21                else if (operator.equals("*")) {
22                    s.push(operand1 * operand2);
23                }
24                else {
25                    s.push(operand1 / operand2);
26                }
27            }
28            return s.pop();
29        }
30        public static void main(String[] args) {
31            System.out.println("hasil postfix = " + postfixEvaluate("2 5 * + 7 -"));
32        }
33    }
34 }
```

Program tersebut akan menerjemahkan operasi postfix menjadi nilai sebenarnya. Seperti yang kita ketahui, operasi yang biasa kita lakukan sehari-hari bernama infix, contohnya seperti berikut:

$$2 * 3 + 5$$

Cara kerja infix yaitu kita membaca dari kiri ke kanan, lalu mengoperasikan perkalian dan pembagian terlebih dahulu, lalu penjumlahan dan pengurangan. Sehingga, outputnya :

```
<terminated> StackPostfix [Java Application]
hasil postfix = 8
```

#### **D. Kesimpulan**

Melalui praktikum ini, dipahami bahwa struktur data stack merupakan salah satu bentuk penyimpanan data yang sangat efisien dalam berbagai situasi pemrograman karena mengikuti prinsip LIFO (Last In, First Out). Operasi dasar seperti push, pop, dan peek/top menjadi inti dari seluruh aktivitas dalam stack. Penggunaan stack dapat dilakukan baik secara manual menggunakan array maupun dengan memanfaatkan library bawaan seperti `java.util.Stack`. Implementasi stack secara manual membantu memahami cara kerja internal struktur data ini, sedangkan penggunaan stack bawaan mempercepat proses pengembangan. Stack juga terbukti bermanfaat dalam menyelesaikan masalah riil seperti evaluasi ekspresi postfix dan pencarian nilai maksimum. Dengan demikian, praktikum ini memberikan pemahaman menyeluruh mengenai penerapan stack dalam konteks teori dan praktik pemrograman Java.