LAPORAN PRAKTIKUM STRUKTUR DATA

STACK

Oleh :

HABIBI IKRAMUL HUDA

NIM: 2411532008

MATA KULIAH STRUKTUR DATA

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

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

PADANG, MEI, 2025

A. Pendahuluan

Di Java, stack adalah struktur data yang umum digunakan yang beroperasi dengan prinsip Last-In-First-Out (LIFO), di mana elemen ditambahkan (push) dan dihapus (pop) hanya dari bagian atas stack. Java menyediakan kelas Stack bawaan dalam paket java.util, yang memudahkan developer untuk melakukan operasi stack standar seperti push(), pop(), peek(), dan isEmpty(). Stack sangat berguna dalam skenario yang membutuhkan pemrosesan terbalik, seperti mengevaluasi ekspresi matematika, memeriksa tanda kurung yang seimbang, mekanisme pembatalan, dan algoritme pencarian mendalam. Developer juga bisa membuat implementasi stack khusus, seperti menggunakan array atau senarai terkait, untuk mengontrol penggunaan memori dengan lebih baik atau menambahkan fitur tambahan.

B. Tujuan

1. Mengenal struktur data stack pada java.
2. Mengimplementasikan struktur data stack.

C. Langkah-Langkah

1. Latihan Stack
   1) Buat kelas baru dengan nama latihanStack dan kelas main. Setelah itu, import stack dengan import java.util.Stack;.

```
package pekan3;

import java.util.Stack;

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

   2) Deklarasikan stack s dengan tipe data integer.

```
        Stack<Integer> s = new Stack<Integer>();
```

   3) Tambahkan data integer yaitu 42, -3, dan 17 dengan cara menge-push stack.

```
        s.push(42);
        s.push(-3);
        s.push(17);
```

   4) Tampilkan nilai stack dengan cara menge-print stack s.
   5) Untuk menampilkan nilai pop, print s.pop.
   6) Untuk menampilakn nilai peek, print s.peek.
   7) Untuk menampilakn nilai stack setelah peek, print s.

```
        System.out.println("nilai stack = "+s);
        System.out.println("nilai pop = "+s.pop());
        System.out.println("nilai peek = "+s.peek());
        System.out.println("nilai stack setelah peek = "+s);
    }
}
```

8) Ketika dijalankan, outputnya akan sebagai berikut.

```
nilai stack = [42, -3, 17]
nilai pop = 17
nilai peek = -3
nilai stack setelah peek = [42, -3]
```

2. Stack2

Kode Java ini mendefinisikan sebuah interface yang disebut Stack2<E>. Interface di Java mendefinisikan sekumpulan method yang harus diimplementasikan oleh sebuah kelas jika kelas tersebut mengimplementasikan interface tersebut.

1) Ketikkan public interface Stack2<E>. Interface ini akan mendefinisikan interface yang bernama Stack2<E> dengan tipe parameter E.
2) E bertindak sebagai tipe untuk elemen yang akan di simpan di stack, misalnya Integer atau String.
3) Setiap kelas yang mengimplementasikan interface ini harus mendefinisikan bagainaman method-method ini bekerja untuk tipte-tipe tertentu.

```
public interface Stack2<E> {
```

4) Ketikkan int size();. Method ini akan me-return jumlah element yang ada pada stack.

```
    int size();
```

5) Ketikkan boolean isEmpty();. Method ini akan mengambil elemen e dari tipe E dan menambahkannya ke top stack.

```
    boolean isEmpty();
```

6) Ketikkan void push (E e). Method ini akan mengembalikan nilai true jika stack kosong, dan false sebaliknya.

```
    void push (E e);
```

7) Ketikkan E top();. Method ini me-return elemen pada top stack tanpa menghapusnya. Method ini memungkinkan pengguna untuk melihat top stack tanpa mengganti stack.

```
    E top();
```

8) Ketikkan method E pop();. Method ini akan me-remove dan me-return elemen yan ada apada top stack.

```
    E pop();
}
```

3. contohStack
   1) Buatlah kelas baru dengan nama contohStack.

```
package pekan3;

public class contohStack {
```

   2) Ketiklah sebuah instance baru yaitu ArrayStack test yang mengimplementasikan interface Stack2<E> yang sudah dibuat sebelumnya. Instance ini harus mendefinisikan method push, top, pop, dan size.

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

   3) Buatlah array a dengan tipe data integer dengan array: {4, 8, 15, 16, 23, 42}.

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

   4) Ketikkan for loop yang akan terjadi di array a. Loop ini akan menge-print setiap nilai dan menge-pushnya ke stack test.

```
        for(int i = 0; i < a.length; i++) {
            System.out.println("nilai A"+i+"="+ a[i]);
            test.push(a[i]);
        }
```

   5) Print kode untuk me-return berapa banyak elemen yang ada pada stack dengan test.size().

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

   6) Print kode untuk me-return element teratas pada stack dengan test.pop() tanpa menghapusnya.

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

   7) Print kode untuk menghapus elemen teratas pada stack dengan test.pop().

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

   8) Berikut adalah output dari kode ini.

```
nilai A3=16
nilai A4=23
nilai A5=42
size stacknya: 6
paling atas: 42
nilainya 42
```

4. ArrayStack

Kode ini mendefinisikan implementasi stack generik menggunakan array di Java.

1) Buatlah kelas generic baru yaitu ArrayStack<E> yang mengimplementasikan interface Stack2<E> sehingga kelas ini harus mendefinisikan size(), isEmpty(), push(), top(), dan pop().

```
package pekan3;

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

2) Ketikkan kode untuk menentukan kapasitas default dari stack, yaitu sebanyak 1000 elemen.

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

3) Inisialisasi array E untuk menyimpan elemen pada stack

```
private E[] data;
```

4) Ketikkan kode untuk index pada elemen teratas. -1 artinya stack tidak berisi apapun.

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

5) Ketikkann default konstruktor yang akan membuat stack dengan kapasitas 1000.

```
public ArrayStack() {
    this (CAPACITY);
}
```

6) Ketikkan konstruktor yang memungkinkan pengguna untuk menentukan kapasitas. Karena pengahapusan tipe Java, Object[capacity] yang baru harus di-cast ke E[].

```
public ArrayStack(int capacity) {

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

7) Ketikkan method yang akan mengembalikan jumlah elemen yang ada pada stack. Karen t adalah index teratas, t+1 adalah ukuran stack.

```
public int size() {
return (t + 1);
}
```

8) Ketikkan method yang akan mengecek apakah stack kosong.

```
public boolean isEmpty() {
    return (t == -1);
}
```

9) Ketikkan method yang akan menambahkan elemen ke stack (paling atas). Kode ini akan mengecek apakah stack penuh, jika penuh, aka nada pengecualian. ++t akan menambahkan index sebelum menambahkan element.

```
public void push(E e) throws
IllegalStateException {
    if (size() == data.length)
        throw new
        IllegalStateException("Stack is full");
    data[++t] = e;
}
```

10) Ketikkan method selanjutnya. Method ini akan mengembalikan elemen teratas tanpa menghapusnya. Method ini akan mengembalikan nilai null jika stack kosong.

```
public E top() {
    if (isEmpty())
        return null;
    return data[t];
}
```

11) Ketikkan method yang akan menghapus dan mengembalikan elemen teratas stack. Kode data[t] = null akan membantu dengan pengumpulan sampah dan menghapus referensi. Lalu, method ini akan mengurangi t atau index teratas.

```
public E pop() {
    if (isEmpty())
        return null;
    E answer = data[t];
    data[t] = null;

    t--;
    return answer;
}
}
```

5. NilaiMaksimum

Kode ini dirancang untuk menemukan nilai maksimum dalam Stack<Integer> tanpa mengubah isinya secara permanen. Kode ini akan menyimpan bilangan bulat di dalam stack. Lalu, kode ini akan menemukan nilai maksimum menggunakan suatu method max dank ode ini akan mempertahankan stack asli setelah pencarian.

1) Buatlah kelas baru dengan nama kelas Nilai Maksimum lalu buat juga method main. Lakukan import stack dengan mengetikkan import java.util.Stack;.

```
package pekan3;

import java.util.Stack;

public class NilaiMaksimum {
```

2) Buatlah method max(Stack<Integer> s). Method ini akan mengambil sebuah Stack<Integer> dan mengembalikan nilai maksimum di dalamnya.

```
public static int max(Stack<Integer> s) {
```

3) Buatlah stack backup. Stack sementara ini akan membantu mengembalikan stack asli setelah selesai.

```
Stack<Integer> backup = new Stack<Integer>();
```

4) Inisialisasi maxValue. Variabel ini akan mengeluarkan atau menge-pop elemen pertama dari stack. Lalu, variabel ini akan menambahkan atau menge-pushnya ke stack backup. Asumsikan ini adalah nilai maksimum awal untuk saat ini.

```
int maxValue = s.pop();
backup.push(maxValue);
```

5) Ketikkan loop while pada stack. Loop ini akan terus menge-pop sisa stack. Lalu, loop akan membandingkan setiap nilai yang muncul dengan nilai maksimum dan akan memperbarui jika perlu. Lalu, akan disimpan setiap elemen yang muncul dalam stack backup dan menyimpan stack asli nanti.

```
while (!s.isEmpty()) {
    int next = s.pop();
    backup.push(next);
    maxValue = Math.max(maxValue, next);
}
```

6) Ketikkan loop while. Loop ini akan memindahkan semuanya kembali stack backup ke stack asli yang akan mengembalikannya ke isi asli dan urutan asli.
7) Ketikkan kode pengembalian untuk nilai maksimum yang ditemukan.

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

8) Buatlah main method program ini. Inisialisasi sebuah stack s dengan elemen yang akan di-push yaitu [70, 12, 20] dengan 20 sebagai top.

```
public static void main(String[] args) {
    Stack<Integer> s = new Stack<Integer>();
    s.push(70);
    s.push(12);
    s.push(20);
```

9) Print isi stack.

```
System.out.println("isi stack "+ s);
```

10) Print top stack tanpa menge-pop stack.

```
System.out.println("stack teratas "+ s.peek());
```

11) Print nilai maksimum yang akan mengembalikan nilai maksimum yaitu 70 dan mengembalikan stack seperti semula setelahnya.

```
System.out.println("nilai maksimum "+ max(s));
    }

}
```

12) Berikut adalah output dari kode ini:

```
isi stack [70, 12, 20]
stack teratas 20
nilai maksimum 70
```

6. StackPostfix

Program Java ini mengevaluasi ekspresi postfix menggunakan stack. Notasi Postfix (juga disebut Reverse Polish Notation, atau RPN) adalah cara menulis ekspresi matematika tanpa tanda kurung.

1) Buatlah kelas baru dengan nama StackPostfix. Import stack dan juga stack pada java util.

```
package pekan3;

import java.util.Scanner;
import java.util.Stack;
```

2) Buatlah method postfixEvaluate(String expression). Method ini akan mengambil sebuah string yang akan merepresentasikan sebuah ekspresi postfix. Method akan mengembalikan hasil akhir sebagai integer.

```
public static int postfixEvaluate(String expression) {
```

3) Inisialisasi stack s dengan tipe data integer yang akan menyimpan operand. Scanner juga digunakan untuk membagi ekspresi menjadiangka dan operator.

```
Stack<Integer> s = new Stack<Integer>();
Scanner input = new Scanner(expression);
```

4) Ketikkan loop while untuk terus membaca token sampai input habis. Jika input adalah angka, ineteger akan di-push ke stack. Jika input adalah operator, operator akan dibaca. Lalu, dua operand dari stack akan dikeluarkan (pop). Lalu, ekspresi akan dievaluasi dan hasil akan di-push kembali ke stack. Setelah semua token diproses, hasil akhir hanya angka yang tersisa di stack.

```
        while (input.hasNext()) {
            if (input.hasNextInt()) {
                s.push(input.nextInt());
            } else {
                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();
    }
```

5) Buatlah main method.. Print hasil postfix.

```
    public static void main(String[] args) {
        System.out.println("hasil postfix = " + postfixEvaluate("5 2 5 * + 7 -"));
    }
}
```

6) Berikut adalah hasil dari program ini.

```
hasil postfix = 8
```

D. Kesimpulan

Stack adalah struktur data linear yang mengikuti prinsip Last-In-First-Out (LIFO), yang berarti item terakhir yang ditambahkan adalah item pertama yang dihapus. Pada kode sebelumnya, stack diimplementasikan menggunakan kelas Stack bawaan Java dan kelas khusus yang disebut ArrayStack yang secara internal menggunakan array untuk menyimpan elemen dan pointer top (t) untuk melacak item terbaru. Implementasi khusus ini mencakup operasi-operasi stack yang penting seperti push, pop, top, isEmpty, dan size. Stack ini kemudian digunakan dalam contoh-contoh aplikasi seperti menemukan nilai maksimum dalam stack (NilaiMaksimum) dan mengevaluasi ekspresi postfix (StackPostfix) yang mendemonstrasikan bagaimana elemen ditambahkan dan dihapus menurut aturan LIFO sambil mempertahankan atau memanipulasi isi stack.