LAPORAN PRAKTIKUM STRUKTUR DATA

PEKAN 3: IMPLEMENTASI DAN APLIKASI STRUKTUR DATA STACK DALAM BAHASA JAVA

OLEH

Abdullah Al Ramadhani (2411533016)

DOSEN PENGAMPU

DR. Wahyudi, S.T, M.T

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2025

A. Pendahuluan

Stack adalah salah satu struktur data yang bekerja dengan prinsip Last In First Out (LIFO), artinya elemen yang dimasukkan terakhir akan menjadi yang pertama dikeluarkan. Struktur ini sering digunakan dalam berbagai aspek komputasi, seperti evaluasi ekspresi matematika, rekursi dalam pemrograman, dan manajemen memori pada sistem operasi.

Biasanya, Stack diimplementasikan menggunakan array atau linked list. Operasi dasar yang umum dilakukan pada Stack meliputi **push** (menambahkan elemen ke stack), **pop** (menghapus elemen dari atas stack), dan **top** (melihat elemen paling atas tanpa menghapusnya). Stack juga sangat berguna dalam pengolahan ekspresi matematika, terutama dalam bentuk postfix, karena sifat LIFO-nya memungkinkan pemrosesan operand dan operator secara efisien.

B. Tujuan

Tujuan dari praktikum ini adalah:

1. Memahami konsep dasar dari struktur data Stack beserta operasi-operasi yang dapat dilakukan di dalamnya.
2. Menerapkan Stack menggunakan array serta menguji fungsinya dalam berbagai kondisi.
3. Menjelaskan penerapan Stack secara nyata dalam menyelesaikan ekspresi matematika bertipe postfix.
4. Mengetahui peran Stack dalam berbagai algoritma dan penerapannya di berbagai permasalahan komputasi.

C. Langkah-langkah Pengerjaan

a. Program Interface Stack2

1. Buatlah package baru dengan nama **"pekan3"** dan tambahkan sebuah class dengan nama **Stack2** di dalamnya.
2. Kemudian, buat sebuah interface yang berfungsi sebagai kerangka dasar dari stack. Interface ini akan digunakan untuk mendefinisikan kontrak dari beberapa method penting, seperti **push**, **pop**, **top**, **size**, dan **isEmpty**. Metode **size()** digunakan untuk mengetahui jumlah elemen dalam stack, **isEmpty()** untuk memeriksa apakah stack dalam keadaan kosong, **push(E e)** untuk menambahkan elemen ke dalam stack, **top()** untuk melihat elemen paling atas, dan **pop()** untuk mengambil sekaligus menghapus elemen paling atas dari stack.

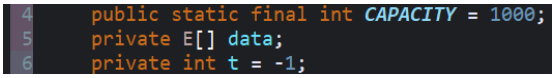
```
1  package Pekan3;
2
3  public interface Stack2<E> {
4          int size();
5          boolean isEmpty();
6          void push (E e);
7          E top();
8          E pop();
9  }
10
```

b. Program ArrayStack

1. Buatlah class baru dengan nama **"ArrayStack"** yang menerapkan interface **Stack2**.
2. Tambahkan deklarasi **generic <E>** pada class **ArrayStack** dan implementasikan interface **Stack2**. Class ini akan digunakan untuk membangun struktur data Stack secara manual menggunakan array. Dengan penggunaan generic **<E>**, class ini bisa menangani berbagai jenis tipe data. Interface **Stack2** sendiri berisi deklarasi method-method dasar dari stack.

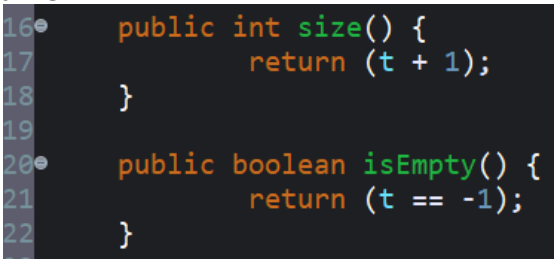
```
1  package Pekan3;
2
3  public class ArrayStack<E> implements Stack2<E> {
```

3. Deklarasikan konstanta untuk kapasitas default dan variabel-variabel utama stack. Buatlah variabel CAPACITY bernilai 1000 sebagai kapasitas default jika pengguna tidak menentukan kapasitas secara eksplisit. Setelah itu, deklarasikan array generic bernama data untuk menyimpan elemen-elemen stack dan sebuah variabel t untuk menunjukkan posisi elemen yang ada di paling atas. Nilai awal t diatur ke -1, yang menandakan bahwa stack masih kosong.

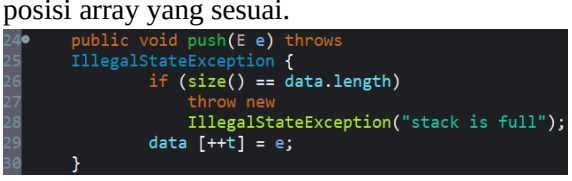


4. Buat dua konstruktor: satu konstruktor default dan satu konstruktor yang menerima parameter kapasitas. Konstruktor default nantinya akan memanggil konstruktor kedua dengan nilai kapasitas default dari CAPACITY. Konstruktor kedua menerima nilai kapasitas sebagai argumen, lalu membuat array data sesuai ukuran tersebut. Tujuannya adalah untuk menentukan kapasitas awal stack sekaligus mengalokasikan memori.

5. Selanjutnya, implementasikan method size() dan isEmpty(). Method size() akan mengembalikan jumlah elemen di stack dengan cara menghitung nilai t + 1, karena indeks array dimulai dari nol. Sementara itu, isEmpty() akan mengembalikan nilai true jika t == -1, yang berarti belum ada elemen di dalam stack.



6. Tambahkan method push() yang bertugas untuk memasukkan elemen baru ke stack. Method ini akan mengecek apakah stack sudah penuh dengan membandingkan jumlah elemen sekarang dengan kapasitas array. Jika penuh, maka akan dilemparkan exception bertipe IllegalStateException. Jika masih ada ruang, nilai t akan dinaikkan dan elemen e disimpan ke posisi array yang sesuai.



7. Implementasikan method top() untuk melihat isi elemen paling atas pada stack. Method ini akan mengembalikan elemen teratas tanpa menghapusnya. Jika stack dalam keadaan kosong, maka akan mengembalikan null.

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

8. Setelah itu, buat method pop() yang fungsinya untuk menghapus elemen teratas dari stack. Pertama-tama, method ini akan memeriksa apakah stack kosong. Jika iya, maka akan mengembalikan null. Jika tidak kosong, elemen teratas akan disimpan ke dalam variabel sementara bernama answer, kemudian elemen tersebut dihapus dari array dengan menyetelnya ke null (untuk membantu proses garbage collection), dan terakhir nilai t dikurangi satu.

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

c. Program contohStack

1. Buatlah class baru dengan nama “contohStack”.

2. Class ini akan berfungsi sebagai program utama untuk menguji bagaimana class ArrayStack bekerja. Di dalamnya akan diperlihatkan proses penambahan elemen ke dalam stack serta bagaimana stack merespons saat dilakukan operasi pop.

3. Pada method main(), buat sebuah objek dari class ArrayStack serta array bertipe Integer. Objek yang dibuat (misalnya bernama test) akan digunakan untuk menampung elemen-elemen yang akan dimasukkan ke stack. Sementara itu, array a akan berisi enam angka bulat yang akan dimasukkan ke dalam stack sebagai data awal.

```
    public static void main(String[] args) {
        ArrayStack test = new ArrayStack();
        Integer[] a = {4, 8, 15, 16, 23, 42};
```

4. Gunakan perulangan for untuk menampilkan dan menambahkan elemen dari array ke stack. Setiap elemen dari array a akan ditampilkan ke layar dan dimasukkan ke stack melalui method push(), sehingga elemen-elemen tersebut masuk ke stack secara berurutan.

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

5. Setelah seluruh elemen berhasil ditambahkan ke stack, tampilkan jumlah elemen saat ini menggunakan method size(). Hal ini dilakukan untuk memastikan bahwa semua elemen dari array telah masuk ke stack dengan benar.

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

6. Lakukan operasi pop() sebanyak dua kali untuk menghapus dua elemen teratas dari stack dan tampilkan hasilnya. Karena stack mengikuti prinsip LIFO (Last In First Out), maka dua elemen terakhir yang dimasukkan, yaitu 42 dan 23, akan menjadi yang pertama keluar.

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

7. Setelah class Stack2, ArrayStack, dan contohStack selesai dibuat, program sudah siap untuk dijalankan.

```
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: 23
```

d. Program LatihanStack

1. Buat class baru dengan nama latihanStack, seperti pada pembuatan class sebelumnya. Class ini digunakan untuk mencoba penggunaan struktur data stack bawaan dari Java yang tersedia dalam library java.util.Stack. Tujuannya adalah untuk menunjukkan bagaimana operasi dasar stack bisa dilakukan tanpa harus membuat class sendiri seperti ArrayStack.

2. Buatlah sebuah objek stack dengan tipe Integer dan tambahkan beberapa elemen menggunakan method push(). Dalam contoh ini, elemen yang akan dimasukkan adalah 42, -3, dan 17.

```
public static void main(String[] args) {
    Stack<Integer> s = new Stack<Integer>();
    s.push(42);
    s.push(-3);
    s.push(17);
```

3. Gunakan System.out.println() untuk menampilkan isi dari stack saat ini. Elemen-elemen akan terlihat mulai dari elemen yang pertama kali dimasukkan hingga yang terakhir berada di bagian atas stack.

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

4. Gunakan method pop() untuk menghapus elemen teratas dari stack, yaitu angka 17. Setelah itu, tampilkan kembali isi stack untuk melihat sisa elemen yang ada.

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

5. Gunakan method peek() untuk melihat elemen paling atas yang baru tanpa menghapusnya. Setelah method peek() dipanggil, tampilkan isi stack lagi untuk menunjukkan bahwa strukturnya tetap sama dan tidak berubah.

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

6. Jalankan program untuk melihat hasil dari penggunaan stack, pop(), dan peek() yang telah dibuat sebelumnya.

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

e. NilaiMaksimum

1. Buatlah class baru dengan nama “NilaiMinimum”. Class ini dirancang untuk mencari nilai maksimum dari elemen-elemen yang ada di dalam stack. Proses pencarian dilakukan tanpa mengubah isi akhir stack, sehingga menunjukkan bagaimana stack bisa dimanipulasi tanpa kehilangan data aslinya.

```
public static int max(Stack<Integer> s) {
    Stack<Integer> backup = new Stack<Integer>();
    int maxValue = s.pop();
    backup.push(maxValue);
```

2. Gunakan method max() untuk menemukan nilai tertinggi dengan cara memindahkan elemen satu per satu ke dalam stack cadangan (backup), sambil membandingkan tiap nilainya. Nilai terbesar akan disimpan dalam variabel maxValue.

```
public static int max(Stack<Integer> s) {
    Stack<Integer> backup = new Stack<Integer>();
    int maxValue = s.pop();
    backup.push(maxValue);
```

3. Lakukan perulangan untuk memindahkan elemen dari stack utama ke stack backup sambil mengecek nilai maksimum. Selama stack masih berisi elemen, ambil satu per satu elemennya, pindahkan ke backup, dan bandingkan dengan nilai maksimum sebelumnya. Proses ini diulang sampai seluruh elemen selesai diperiksa.

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

4. Setelah semua elemen dipindahkan dan dibandingkan, kembalikan elemen dari stack backup ke stack asal. Langkah ini bertujuan untuk mengembalikan kondisi stack seperti semula agar datanya tetap utuh dan bisa digunakan kembali untuk proses lain.

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

5. Di dalam method main(), buat sebuah objek stack dan tambahkan beberapa angka. Setelah itu, tampilkan isi stack dan panggil method max() untuk mencari nilai maksimum dari elemen-elemen yang ada di dalamnya.

```
public static void main(String[] args) {
    Stack<Integer> s = new Stack<Integer>();
    s.push(10);
    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));
}
```

6. Jalankan program dan kamu akan melihat isi stack yang telah dibuat, elemen yang berada di posisi teratas, serta nilai maksimum dari stack tersebut.

```
isi stack [10, 12, 20]
Stack Teratas 20
Nilai maksimum 20
```

f. StackPostFix

1. Buatlah class baru dengan nama “StackPostFix”. Class ini dibuat untuk menghitung nilai dari ekspresi postfix, atau yang dikenal juga sebagai notasi polska balik. Dalam ekspresi postfix, operator diletakkan setelah operand, dan proses evaluasinya sangat cocok menggunakan struktur data stack karena prinsip kerjanya LIFO (Last In, First Out).

2. Buat method bernama postfixEvaluate(String expression) yang menerima input berupa string ekspresi postfix. Method ini akan membaca setiap elemen dalam ekspresi dan melakukan perhitungan sesuai operator yang ditemukan. Jika elemen berupa angka, maka akan dimasukkan ke dalam stack. Jika ditemukan operator, maka dua angka terakhir dari stack diambil, dihitung sesuai operatornya, lalu hasilnya dikembalikan ke stack.

```
public static int postfixEvaluate(String expression) {
    Stack<Integer> s = new Stack<Integer>();
    Scanner input = new Scanner(expression);
```

3. Gunakan perulangan untuk membaca bagian-bagian dari ekspresi. Jika elemen yang terbaca adalah angka, langsung masukkan ke stack. Namun jika berupa simbol operator seperti +, -, *, atau /, ambil dua elemen terakhir dari stack, lakukan operasi sesuai simbol, lalu hasilnya dimasukkan kembali ke stack.

```
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 if (operator.equals("/"))
            s.push(operand1 / operand2);
        else
            s.push(operand1 / operand2);
    }
}
return s.pop();
```

4. Tambahkan method main() untuk menjalankan program, lalu panggil method postfixEvaluate() dengan ekspresi contoh. Misalnya, ekspresi "5 2 5 * + 7 -" berarti: 5 + (2 * 5) - 7, dan hasil akhirnya adalah 8. Program akan mencetak hasil evaluasi ekspresi tersebut ke layer.

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

5. Setelah semua selesai dibuat, jalankan program untuk melihat apakah hasil evaluasi dari ekspresi postfix yang diberikan sudah benar atau belum.

```
<terminated> StackPostFix
hasil postfix = 8
```

D. Kesimpulan

Struktur data Stack merupakan komponen penting dalam pemrograman dan algoritma karena bekerja berdasarkan prinsip Last In First Out (LIFO). Dengan prinsip ini, Stack memungkinkan pengelolaan data yang efisien dan sering digunakan dalam berbagai situasi seperti evaluasi ekspresi matematika, pengelolaan antrian proses, hingga navigasi dalam proses rekursif.

Dalam bahasa Java, Stack bisa dibuat menggunakan array ataupun dengan memanfaatkan class bawaan seperti java.util.Stack. Operasi dasar yang mendukung kerja Stack antara lain push (untuk menambahkan elemen), pop (menghapus elemen teratas), dan peek (melihat elemen paling atas tanpa menghapusnya), yang menjadi dasar dari berbagai implementasi praktis.

Pemahaman tentang konsep dan mekanisme kerja Stack sangat penting sebagai landasan untuk mempelajari struktur data lainnya. Selain itu, penerapan Stack memperlihatkan bagaimana prinsip sederhana seperti LIFO bisa dimanfaatkan untuk menyelesaikan berbagai masalah komputasi secara efektif dan sistematis.