Laporan Praktikum

Struktur Data



Disusun Oleh :

**Ahda Rindang Al-Amin (2311531003)**

Dosen Pengampu : Dr. Wahyudi, S.T, M.T

Departemen Informatika

Fakultas Teknologi Informasi

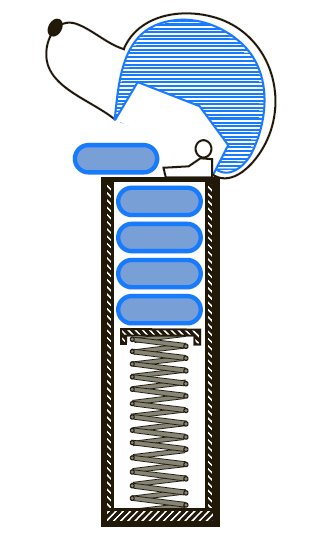
Universitas Andalas

Tahun 2024

**Stack**

1. **Tujuan Praktikum**
2. Memahami penggunaan Stack untuk struktur data pada bahasa pemograman Java
3. **Pendahuluan**

A ***stack*** is a collection of objects that are inserted and removed according to the ***last-in, first-out*** (***LIFO***) principle. A user may insert objects into a stack at any time, but may only access or remove the most recently inserted object that remains (at the so-called “top” of the stack). The name “stack” is derived from the metaphor of a stack of plates in a spring-loaded, cafeteria plate dispenser. In this case, the fundamental operations involve the “pushing” and “popping” of plates on the stack. When we need a new plate from the dispenser, we “pop” the top plate off the stack, and when we add a plate, we “push” it down on the stack to become the new top plate. Perhaps an even more amusing example is a PEZ® candy dispenser, which stores mint candies in a spring-loaded container that “pops” out the topmost candy in the stack when the top of the dispenser is lifted.



The java.util.Stack Class

Stacks are the simplest of all data structures, yet they are also among the most important, as they are used in a host of different applications, and as a tool for many more sophisticated data structures and algorithms. Because of the importance of the stack ADT, Java has included, since its original version, a concrete class named java.util.Stack that implements the LIFO semantics of a stack. However, Java’s Stack class remains only for historic reasons, and its interface is not consistent with most other data structures in the Java library. In fact, the current documentation for the Stack class recommends that it not be used, as LIFO functionality (and more) is provided by a more general data structure known as a double-ended queue.

|  |  |
| --- | --- |
| Stack<**E**>() | constructs a new stack with elements of type **E** |
| push(**value**) | places given value on top of stack |
| pop() | removes top value from stack and returns it;  throws EmptyStackExceptionif stack is empty |
| peek() | returns top value from stack without removing it;  throws EmptyStackExceptionif stack is empty |
| size() | returns number of elements in stack |
| isEmpty() | returns true if stack has no elements |

1. **Metode Praktikum**
2. **Contoh Stack Sederhana**

Program berikut merupakan deklarasi stack sederhana.

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 pop= " + s.pop());

System.***out***.println("nilai stack setelah 2 pop= " + s);

}

Output dari program tersebut adalah sebagai berikut:

nilai stack= [42, -3, 17]

nilai pop= 17

nilai stack setelah pop= [42, -3]

nilai pop= -3

nilai stack setelah 2 pop= [42]

1. **Stack Abstract Data Type (ADT)**

Formally, a stack is an abstract data type (ADT) that supports the following two update methods:

push(*e*): Adds element *e* to the top of the stack.

pop( ): Removes and returns the top element from the stack (or null if the stack is empty).

Additionally, a stack supports the following accessor methods for convenience:

top( ): Returns the top element of the stack, without removing it (or null if the stack is empty).

size( ): Returns the number of elements in the stack.

isEmpty( ): Returns a boolean indicating whether the stack is empty.

By convention, we assume that elements added to the stack can have arbitrary type and that a newly created stack is empty.

A Stack Interface in Java

In order to formalize our abstraction of a stack, we define what is known as its ***application programming interface*** (API) in the form of a Java ***interface***, which describes the names of the methods that the ADT supports and how they are to be declared and used. This interface is defined in following code :

public interface ContohStack1 <E>{

int size();

boolean isEmpty();

void push(E e);

E top();

E pop();

}

We rely on Java’s ***generics framework***, allowing the elements stored in the stack to belong to any object type <E>. For example, a variable representing a stack of integers could be declared with type Stack<Integer>. The formal type parameter is used as the parameter type for the push method, and the return type for both pop and top.

1. **Simple Array-Based Stack Implementation**

Recall, from the discussion of Java interfaces, that the interface serves as a type definition but that it cannot be directly instantiated. For the ADT to be of any use, we must provide one or more concrete classes that implement the methods of the interface associated with that ADT. In the following subsections, we will give such implementation of the Stack interface that uses an array for storage.

public class ArrayStack <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) {

//construct 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

}

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;

}

}

As our first implementation of the stack ADT, we store elements in an array, named data, with capacity *N* for some fixed *N*. We oriented the stack so that the bottom element of the stack is always stored in cell data[0], and the top element of the stack in cell data[t] for index *t* that is equal to one less than the current size of the stack

Recalling that arrays start at index 0 in Java, when the stack holds elements from data[0] to data[*t*] inclusive, it has size *t* +1. By convention, when the stack is empty it will have *t* equal to −1 (and thus has size *t* +1, which is 0).

1. **Contoh penggunaan Array-Based Stack**

Program berikut memanggil kelas ArrayStack yang telah dibuat, dan menggunakan method-method yang ada di kelas tersebut.

public static void main(String[] args) {

ArrayStack test = new ArrayStack();

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

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());

}

Output dari program tersebut adalah sebagai berikut:

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

1. **Mencari Nilai Maksimum pada Stack**

Pada program berikut, ditulis method max yang berfungsi untuk mencari nilai maksimum pada suatu Stack yang diberikan, dan method digunakan pada Stack yang diberikan pada method main.

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

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

int maxValue = s.pop();

backup.push(maxValue);

while(!s.empty()) {

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("Nilai Maksimum " + *max*(s) +

"\nStack Teratas " + s.peek() +

"\nIsi Stack" + s);

Pada method max, nilai elemen-elemen yang ada dalam Stack yang diberikan dipanggil dan dikeluarkan dari Stack menggunakan fungsi pop() satu-persatu dan dicek nilai maksimum dengan menggunakan fungsi Math.max(maxValue, next). Setelah itu, elemen yang telah dikeluarkan dimasukkan ke Stack backup. Setelah Stack yang diberikan telah kosong, dan nilai maksimum telah didapatkan, elemen-elemen yang telah dipindahkan ke dalam Stack backup dikembalikan ke Stack awal. Lalu, nilai variabel maxValue sebagai nilai maksimum dikembalikan dari method tersebut.

Output program tersebut adalah sebagai berikut:

Nilai Maksimum 70

Stack Teratas 20

Isi Stack[70, 12, 20]

1. **Algoritma Postfix**

Rincian dari algoritma postfix adalah sebagai berikut:

* Saat ada operand, push ke dalam stack.
* Saat ada operator :
  + pop dua operand terakhir.
  + lakukan operasi operator pada kedua operand.
  + push hasil operasi ke dalam stack.
* setelah selesai, satu-satunya elemen yang tersisa di stack adalah hasilnya.

public static int postfixEvaluate (String expression) {

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

Scanner input = new Scanner(expression);

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 if (operator.equals("/")) {

s.push(operand1/operand2);

}

}

}

return s.pop();

}

public static void main(String[] args) {

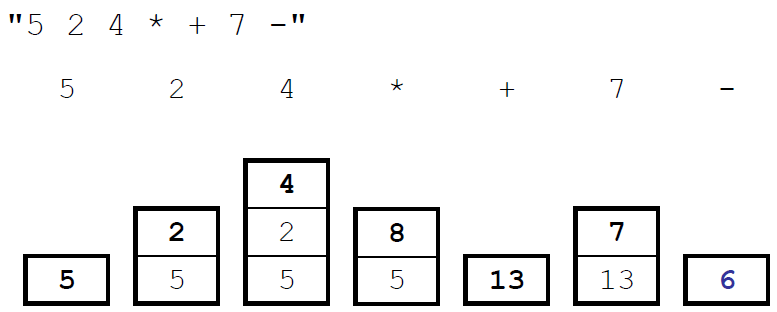
System.***out***.println("Hasil postfix = " +

*postfixEvaluate*("5 2 4 \* + 7 -"));

}

Contoh implementasi algoritma postfix dibuat pada method main. Hasil dari algoritma tersebut adalah ditampilkan pada output program sebagai berikut:

Hasil postfix = 6



1. **Kesimpulan Praktikum**

Stack adalah kumpulan objek yang mengikuti prinsip LIFO (last-in first-out). Stack merupakan struktur data paling sederhana, namun termasuk bagian yang terpenting dalam struktur data. Konsep Stack cukup sederhana, implementasi stack dapat dibuat menggunakan basis array sebagai penyimpanan. Namun, pada bahasa pemograman Java, telah diberikan library khusus Stack dengan berbagai method Stack ADT seperti push(), pop(), peek(), size(), dan isEmpty().