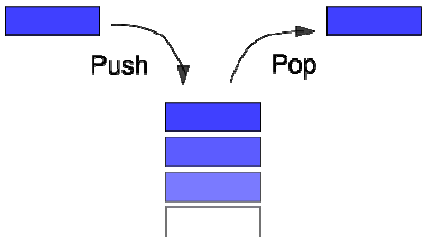
PRAKTIKUM 21
STACK

A. TUJUAN PEMBELAJARAN

1. Memahami konsep dan operasi pada Stack.
2. Mampu mengimplementasikan struktur data Stack pada array dan List.

B. DASAR TEORI

Salah satu konsep yang efektif untuk menyimpan dan mengambil data adalah "terakhir masuk sebagai pertama yang keluar" (Last In First Out/FIFO). Dengan konsep ini, pengambilan data akan berkebalikan urutannya dengan penyimpanan data. Stack(tumpukan) adalah sebuah kumpulan data dimana data yang diletakkan di atas data yang lain. Dengan demikian stack adalah struktur data yang menggunakan konsep LIFO. Elemen terakhir yang disimpan dalam stack menjadi elemen pertama yang diambil. Dalam proses komputasi, untuk meletakkan sebuah elemen pada bagian atas stack disebut dengan push. Dan untuk memindahkan dari tempat teratas tersebut, kita melakukan pop.



Gambar 1. Ilustrasi Stack

Ada 2 operasi paling dasar dari stack yang dapat dilakukan, yaitu :

1. Operasi push yaitu operasi menambahkan elemen pada urutan terakhir (paling atas).
2. Operasi pop yaitu operasi mengambil sebuah elemen data pada urutan terakhir dan menghapus elemen tersebut dari stack.

1. Java Stack Collection

Package Java juga menyediakan class Stack pada `java.util.Stack`, yang merupakan subclass dari Vector yang menggunakan standar *last-in first-out* (LIFO). Class Stack hanya digunakan untuk menentukan default constructor, untuk membuat stack kosong. Berikut ini beberapa metode yang digunakan dalam stack seperti terlihat pada Tabel 1.

Tabel 1. Metode pada java.util.stack

Metode-metode pada kelas Stack	Deskripsi
`boolean empty( )`	Menghasilkan nilai **True** jika stack kosong, dan nilai **False** jika stack berisi elemen
`Object peek( )`	Menghasilkan elemen pada top stack, tetapi tidak me-*remove*.
`Object pop( )`	Menghasilkan elemen pada top stack, dan mengambil/menghapus (*remove*) elemen tersebut.
`Object push(Object element )`	Menambahkan elemen pada stack.
`search (Object Element)`	Mencari elemen dalam stack. Jika ditemukan,menghasilkan offset dari top stack . Sebaliknya jika tidak menghasilkan nilai –1.

2. Implementasi Stack dengan Array dan ArrayList

Selain menggunakan *java stack collection*, kita dapat mengimplementasikan stack dengan menggunakan arraylist. Untuk mengimplementasikan stack digunakan interface Stack yang berisi fungsi-fungsi berikut:

Tabel 2. Interface Stack

`Interface Stack`	
`boolean`	`isEmpty()` mengembalikan true jika stack kosong dan false jika stack berisi elemen.
`T`	`peek()` mengambil nilai dari atas stack, jika stack kosong maka melempar (throw) EmptyStackException
`T`	`pop()` menghapus elemen dari atas stack dan mengembalikan nilainya. Jika stack kosong maka melempar(throw) EmptyStackException.
`void`	`push(T item)` menambahkan item di atas stack
`int`	`size()` mengembalikan jumlah elemen yang terdapat di stack.

Implementasi stack dapat menggunakan array atau arraylist. Untuk mengimplementasikan stack menggunakan array seperti di bawah ini.

```
public class StackArr<T> implements Stack {
      T value[] ;
      int topOfStack ;

      public boolean isEmpty(){…}
      public T pop(){…}
      public void push(T item){…}
      public T peek(){…}
      public int size() {…}
}
```

Sedangkan untuk mengimplementasikan stack menggunakan arraylist seperti di bawah ini.

```
 public class ALStack<T> implements Stack {
      // storage structure
      private ArrayList<T> stackList = null;
      // create an empty stack by creating  an empty ArrayList
      public ALStack(){
            stackList = new ArrayList<T>();
      }

      public boolean isEmpty(){…}
      public T pop(){…}
      public void push(T item){…}
      public T peek() {…}
      public int size() {…}
 }
```

C. TUGAS PENDAHULUAN

Jawablah pertanyaan berikut ini :

1. Jelaskan pengertian tentang Stack
2. Sebutkan dan jelaskan dua operasi dasar stack
3. Jelaskan operasi-operasi pada stack yaitu
   a. boolean empty( )
   b. Object peek( )
   c. Object pop( )
   d. Object push(Object element

D. PERCOBAAN

Percobaan 1 : ***Menggunakan Stack Collection pada java.util.stack***

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

```
public class StackExample {
    public static void main(String args[]) {
    Stack s = new Stack();
    s.push("Java");
    s.push("Source");
    s.push("and");

    System.out.println("Next: " + s.peek());
    s.push("Support");
    System.out.println(s.pop());
    s.push(".");
    int count = s.search("Java");
    while (count != -1 && count > 1) {
      s.pop();
      count--;
    }
    System.out.println(s.pop());
    System.out.println(s.empty());
  }
}
```

Percobaan 2 : ***Menggunakan Stack Collection pada java.util.stack dan iterator***

```
import java.util.Iterator;
import java.util.Stack;

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

        Stack<String> sk=new Stack<String>();

        sk.push("a");
        sk.push("c");
        sk.push("e");
        sk.push("d");

        Iterator it=sk.iterator();

        System.out.println("Size before pop() :"+sk.size());

        while(it.hasNext())
        {
            String iValue=(String)it.next();
            System.out.println("Iterator value :"+iValue);
        }

        // get and remove last element from stack
        String value =(String)sk.pop();

        System.out.println("value :"+value);

        System.out.println("Size After pop() :"+sk.size());
```

```
    }
}
```

Percobaan 3 : *Implementasi stack dengan Array*

```
public interface Stack<T> {
    abstract boolean isEmpty();
    abstract T peek();
    abstract T pop();
    abstract void push(T item);
    abstract int size();
}
```

```
import java.util.EmptyStackException;
public class StackArr<T> implements Stack<T> {
```

```
    T value[];
    int topOfStack;

    public StackArr(int size){
        value = (T[]) new Object[size];
    }
    @Override
    public boolean isEmpty() {
        return topOfStack == 0;
    }
    @Override
    public T pop() {
        if (isEmpty()) {
            throw new EmptyStackException();
        }
        topOfStack--;
        return value[topOfStack];
    }
    @Override
    public void push(T item) {
        value[topOfStack] = item;
        topOfStack++;
    }
    @Override
    public T peek() {
        if (isEmpty()) {
            throw new EmptyStackException();
        }
        topOfStack--;
        T temp = value[topOfStack];
        topOfStack++;
        return temp;
    }
    @Override
    public int size() {
        return topOfStack;
```

```
    }
    @Override
    public String toString() {
        String str = "";
        for(int i=0; i<topOfStack; i++) {
            str += value[i] + " ";
        }
        return str;
    }
}
```

```
public class TestStackArr {

    public static void main(String[] args) {
        StackArr<String> sa = new StackArr<String>(10);

        sa.push("Pink");
        sa.push("Purple");
        sa.push("Red");
        System.out.println("Push Stack : " + sa.toString());
        System.out.println("Size Stack : " + sa.size());
        sa.pop();
        System.out.println("Pop Stack : " + sa.toString());
        System.out.println("Peek Stack : " + sa.peek());
        System.out.println("Size Stack : " + sa.size());
    }
}
```

Percobaan 3 : *Implementasi stack dengan ArrayList*

```
public interface Stack<T> {
    abstract boolean isEmpty();
    abstract T peek();
    abstract T pop();
    abstract void push(T item);
    abstract int size();

}
```

```
import java.util.ArrayList;
import java.util.EmptyStackException;
import java.util.Iterator;

public class ALStack<T> implements Stack<T> {

    private ArrayList<T> stackList = null;

    public ALStack() {
        stackList = new ArrayList<T>();
    }
```

```
    @Override
    public boolean isEmpty() {
        return stackList.size() == 0;
    }

    @Override
    public T pop() {
        if (isEmpty()) {
            throw new EmptyStackException();
        }
        return stackList.remove(stackList.size() - 1);
    }

    @Override
    public void push(T item) {
        stackList.add(item);
    }

    @Override
    public T peek() {
        if (isEmpty()) {
            throw new EmptyStackException();
        }
        return stackList.get(stackList.size() - 1);

    }

    @Override
    public int size() {
        return stackList.size();

    }
    public Iterator<T> iterator() {
      return stackList.iterator();
    }
}
```

```
import java.util.Iterator;

public class TestALStack {
        public static void main(String[] args) {
        ALStack<String> sa = new ALStack<String>();

        sa.push("Pink");
        sa.push("Purple");
        sa.push("Red");
        System.out.println("Size Stack : " + sa.size());
        sa.pop();

        System.out.println("Peek Stack : " + sa.peek());
        System.out.println("Size Stack : " + sa.size());

        Iterator it=sa.iterator();
```

```
        while(it.hasNext()){
            System.out.println("Iterator Value :  " + it.next());
        }
    }
}
```

E. LATIHAN

1. Dengan menggunakan package stack collection, buatlah program/class untuk
   a. Konversi dari nilai desimal ke nilai biner, oktal dan heksadesimal.

      Contoh :

      ```
      Masukkan nilai desimal = 25
      Hasil nilai biner = 11001
      Hasil nilai oktal = 31
      Hasil nilai heksadesimal = 19
      ```

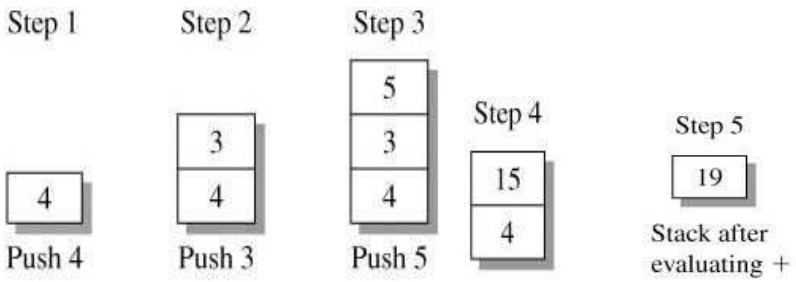
   b. Membalik kalimat dan menentukan sebuah kalimat termasuk palindrom atau bukan

      ```
      Masukkan kalimat : algoritma dan struktur data
      Hasil = atad rutkurts nad amtirogla
      Bukan palindrom

      Masukkan kalimat : sugus
      Hasil = sugus
      Palindrom
      ```

2. Dari soal latihan 1 untuk soal yang sama tetapi menggunakan stack dengan array.
3. Dari soal latihan 1 untuk soal yang sama tetapi menggunakan stack dengan arraylist.
4. Buatlah program untuk menghitung hasil dari ekspresi postfix dengan mengimplementasikan class PostfixEval

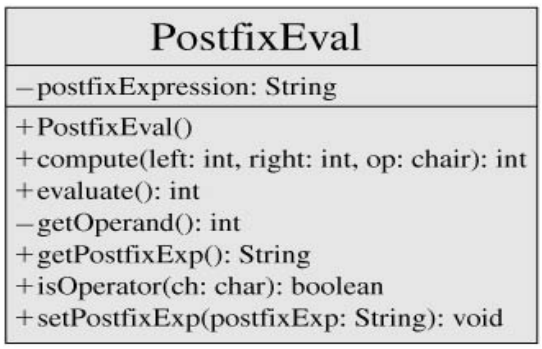
   Contoh : hitunglah nilai dari ekspresi postfix berikut "4 3 5 * +"



Gambar 2. Menghitung nilai dari notasi postfix

Tabel 5. Class PostfixEval

Class PostfixEval	
Atribut	
`postfixExpression`	untuk menerima inputan ekspresi Postfix
Method	
`PostfixEval()`	
`compute(left:int, right:int, op:chair)`	Untuk menghitung operand left dan right dengan operator op
`evaluate()`	untuk mengambil pertama operand kanan dan kemudian operand kiri. Fungsi ini melempar ArithmeticException jika stack kosong.
`getOperand()`	Untuk mendapatkan operand
`getPostfixExp()`	Untuk mendapatkan ekspresi Postfix
`isOperator()`	Untuk menentukan apakah karakter termasuk operator valid ('+','-','*','/','%','^').



Gambar 3. UML dari Class PostfixEval

F. LAPORAN RESMI

Kerjakan hasil percobaan(D) dan latihan(E) di atas dan tambahkan analisa.