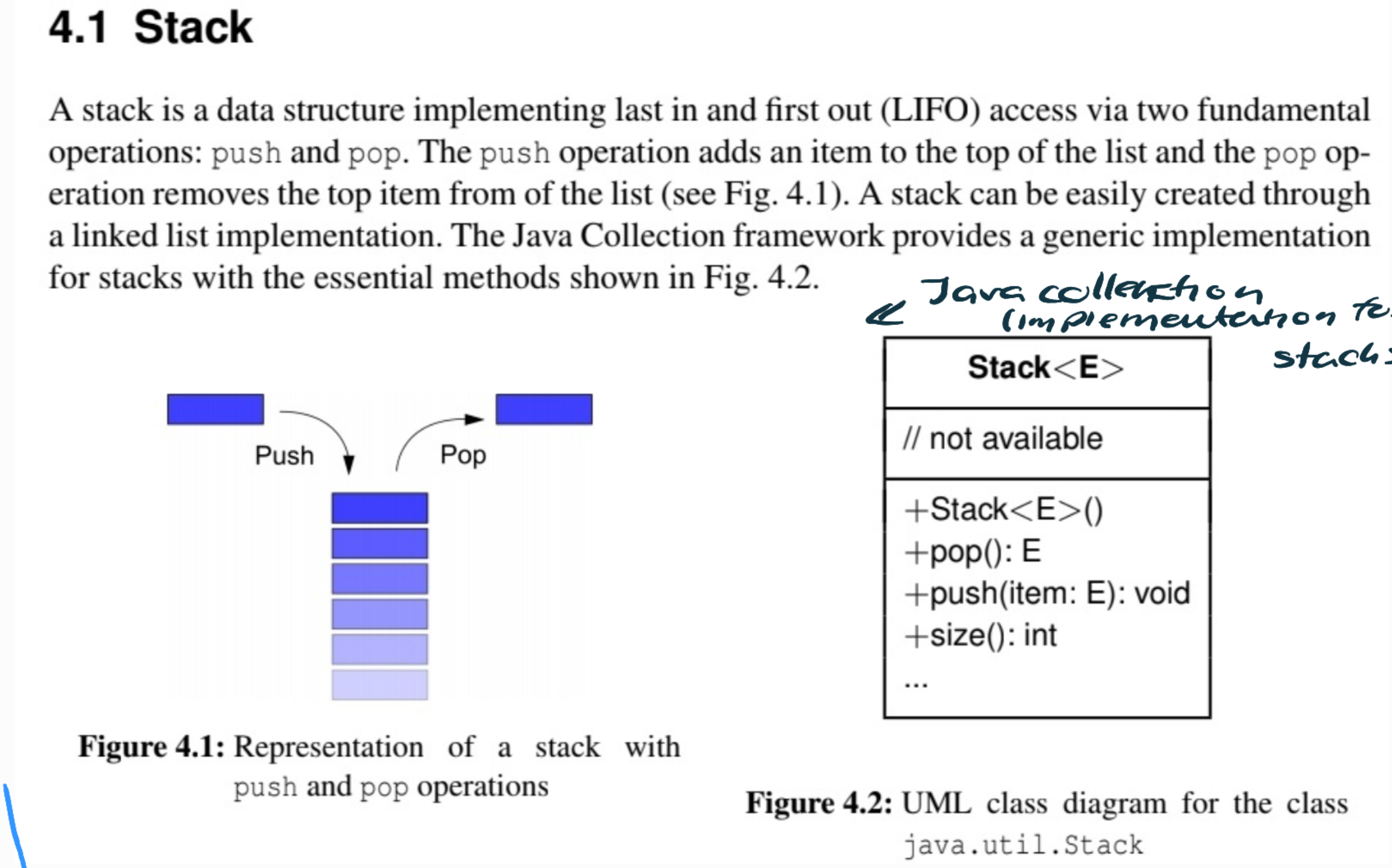


4.1 Stack

A stack is a data structure implementing last in and first out (LIFO) access via two fundamental operations: push and pop. The push operation adds an item to the top of the list and the pop operation removes the top item from of the list (see Fig. 4.1). A stack can be easily created through a linked list implementation. The Java Collection framework provides a generic implementation for stacks with the essential methods shown in Fig. 4.2.

Figure 4.1: Representation of a stack with push and pop operations

Java collection (implementation for stacks)

```
Stack<E>
// not available
+Stack<E>()
+pop(): E
+push(item: E): void
+size(): int
...
```

Figure 4.2: UML class diagram for the class java.util.Stack

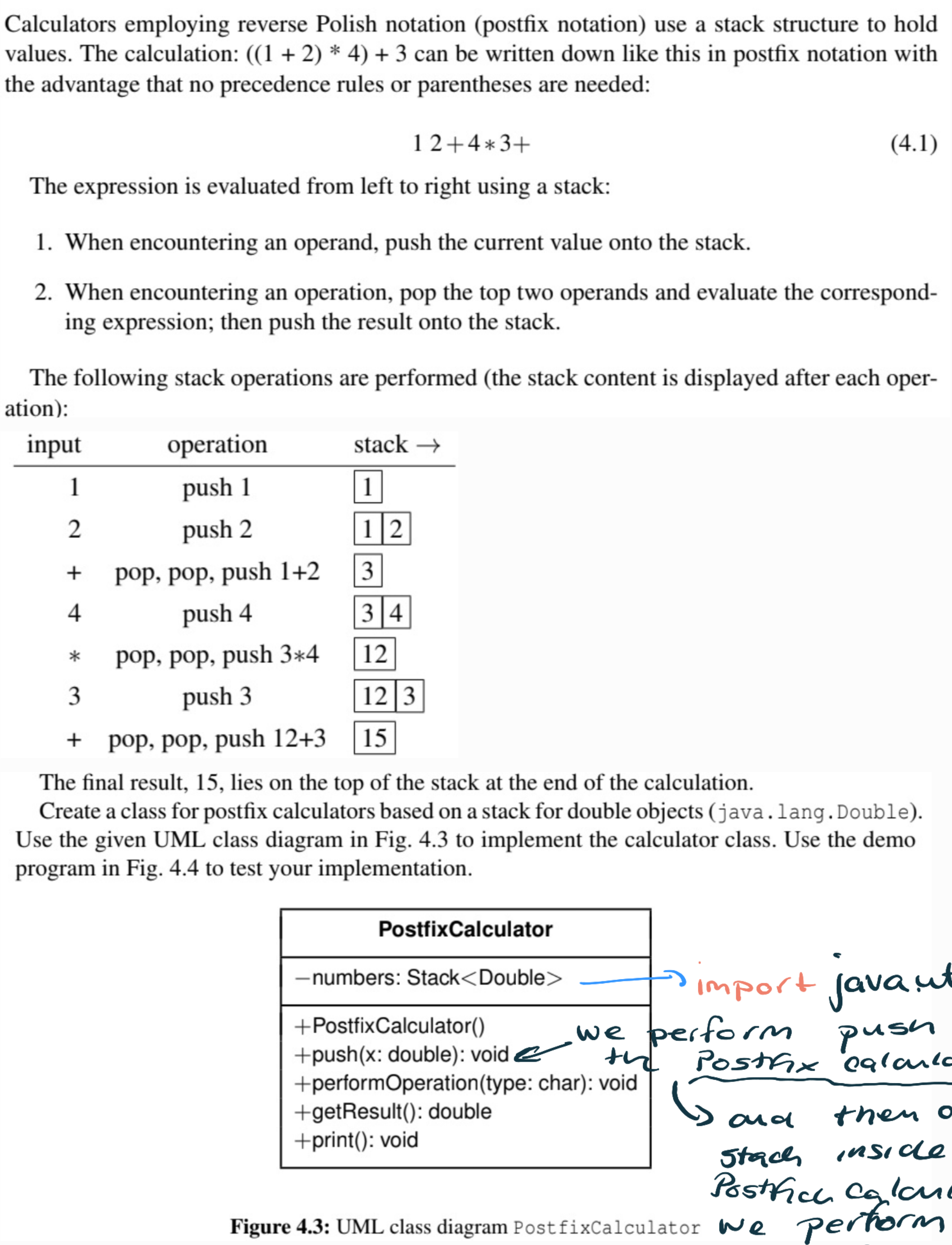
Stack - data structure implementing LAST IN and FIRST OUT (LIFO) access via two fundamental operations PUSH and POP.

push operation - adds an item at the top of the list

pop operation - removes the top item from the list

Stacks can be easily created using a linked list implementation.

Assignment 1



Calculators employing reverse Polish notation (postfix notation) use a stack structure to hold values. The calculation: ((1 + 2) * 4) + 3 can be written down like this in postfix notation with the advantage that no precedence rules or parentheses are needed:

$$1\ 2 + 4 * 3 + \tag{4.1}$$

The expression is evaluated from left to right using a stack:

1. When encountering an operand, push the current value onto the stack.
2. When encountering an operation, pop the top two operands and evaluate the corresponding expression; then push the result onto the stack.

The following stack operations are performed (the stack content is displayed after each operation):

input	operation	stack →
1	push 1	1
2	push 2	1 | 2
+	pop, pop, push 1+2	3
4	push 4	3 | 4
*	pop, pop, push 3*4	12
3	push 3	12 | 3
+	pop, pop, push 12+3	15

The final result, 15, lies on the top of the stack at the end of the calculation.

Create a class for postfix calculators based on a stack for double objects (`java.lang.Double`). Use the given UML class diagram in Fig. 4.3 to implement the calculator class. Use the demo program in Fig. 4.4 to test your implementation.

```
PostfixCalculator
-----------------------------------------
−numbers: Stack<Double>
-----------------------------------------
+PostfixCalculator()
+push(x: double): void
+performOperation(type: char): void
+getResult(): double
+print(): void
```

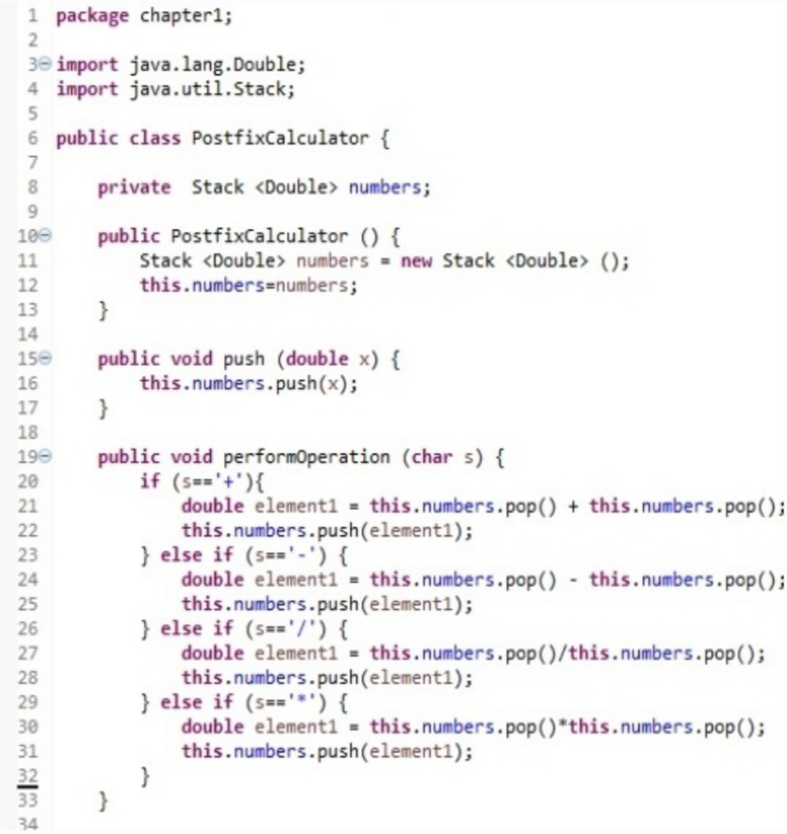
Figure 4.3: UML class diagram `PostfixCalculator`

import java.util.Stack;

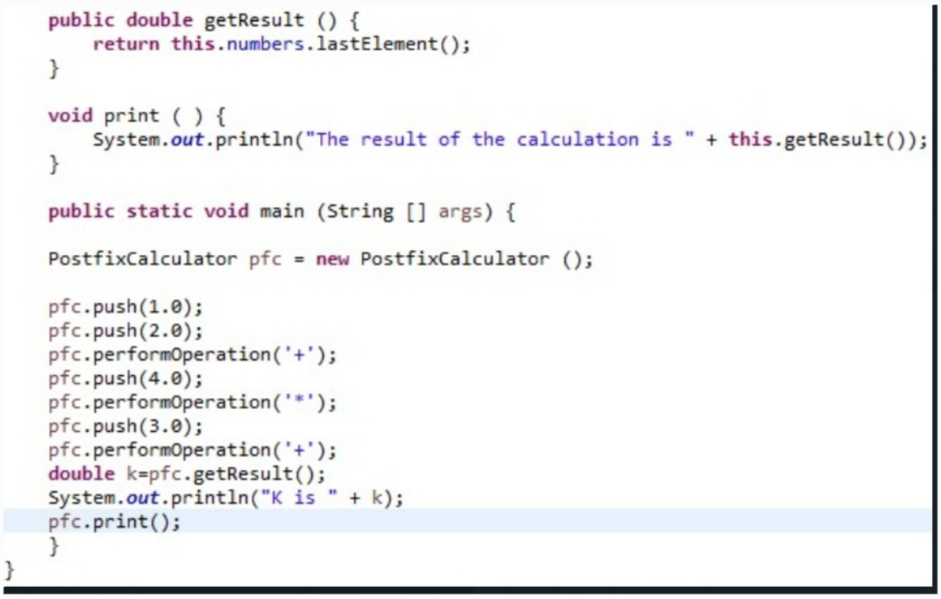
we perform push on the Postfix calculator

and then on stack inside the Postfick calculator we perform push from the java.util.stack

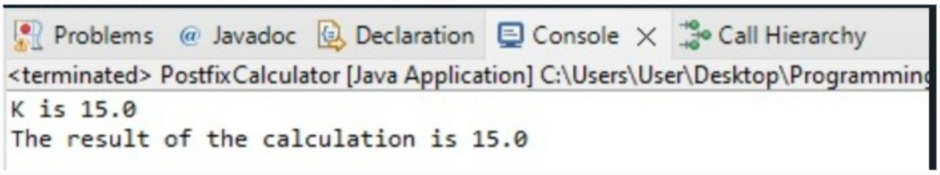
My Java code



```java
package chapter1;

import java.lang.Double;
import java.util.Stack;

public class PostfixCalculator {

    private  Stack <Double> numbers;

    public PostfixCalculator () {
        Stack <Double> numbers = new Stack <Double> ();
        this.numbers=numbers;
    }

    public void push (double x) {
        this.numbers.push(x);
    }

    public void performOperation (char s) {
        if (s=='+'){
            double element1 = this.numbers.pop() + this.numbers.pop();
            this.numbers.push(element1);
        } else if (s=='-') {
            double element1 = this.numbers.pop() - this.numbers.pop();
            this.numbers.push(element1);
        } else if (s=='/') {
            double element1 = this.numbers.pop()/this.numbers.pop();
            this.numbers.push(element1);
        } else if (s=='*') {
            double element1 = this.numbers.pop()*this.numbers.pop();
            this.numbers.push(element1);
        }
    }

```



```java
    public double getResult () {
        return this.numbers.lastElement();
    }

    void print ( ) {
        System.out.println("The result of the calculation is " + this.getResult());
    }

    public static void main (String [] args) {

    PostfixCalculator pfc = new PostfixCalculator ();

    pfc.push(1.0);
    pfc.push(2.0);
    pfc.performOperation('+');
    pfc.push(4.0);
    pfc.performOperation('*');
    pfc.push(3.0);
    pfc.performOperation('+');
    double k=pfc.getResult();
    System.out.println("K is " + k);
    pfc.print();
    }
}
```



```
<terminated> PostfixCalculator [Java Application] C:\Users\User\Desktop\Programmin
K is 15.0
The result of the calculation is 15.0
```