SAFRIZAL RAHMAN  
19 – SIB 1G <https://www.youtube.com/watch?v=dQw4w9WgXcQ>

<https://github.com/safrizalrahman46/Jobsheet9-SEM-2>  
JOBSHEET IX STACK

* 1. Learning Objective

After finishing this practicum session, students will be able to:

* Define the Stack Data Structure
* Create and implement Stack Data Structure
* Implement Stack data Structure with arrays
  1. Lab Activities

In this practicum, we will implement **Stack** class

* + 1. Steps
       1. Take a look at this following class diagram for **Stack** class:

|  |
| --- |
| Stack |
| size: int top: int  data[]: int |
| Stack(size: int) IsEmpty(): boolean IsFull(): boolean push(): void  pop(): void peek(): void print(): void  clear(): void |

Based on class diagram above, we will create the **Stack** class in Java program.

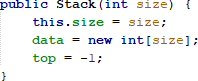
* + - 1. Create a new project named **Jobsheet7.** Create a new package with name **Practicum1.**

Then, create a new class named **Stack**.

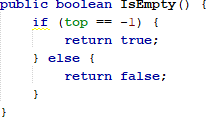
* + - 1. Create new attributes size, top, and data as follows:



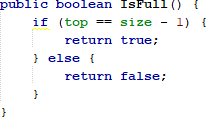
* + - 1. Add a constructor with parameter as written below:



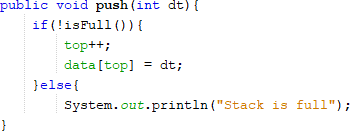
* + - 1. Create a method **isEmpty** with Boolean as its return type to check whether the stack is empty or not.



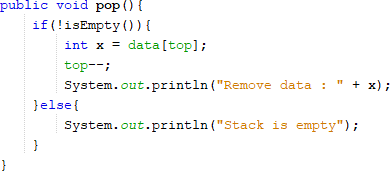
* + - 1. Create a method **isFull** with Boolean as its return type to check whether the stack is filled completely or not.



* + - 1. Create method **push** with void as its return type to add new stack element with parameter **dt**. This dt variable is in form of integer



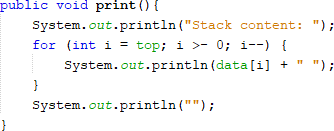
* + - 1. Create method **pop** with void as its return type to remove an element from the stack



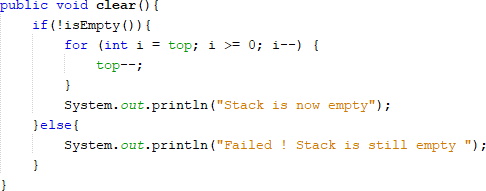
* + - 1. Create method **peek** with void as its return type to check the top element of the stack



* + - 1. Create method **print** with void as its return type to display the content of the stack



* + - 1. Create method **clear** with void as its data type to remove all elements and make the stack empty



* + - 1. Next up, we create a new class named **StackMain** inside the package **Practicum1.** Create a main function and make object instantiation with name is **stk**



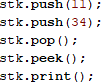
* + - 1. Fill the stack object by calling method **push**, the data is being inserted accordingly



* + - 1. Display the data that we’ve inserted in previous step by calling method **print**



* + - 1. Repeat the insertion process twice, then call pop **method** to remove an element. We can also check the top data with **peek** method. Finally, display all the data by calling method **print**



* + - 1. Compile and run the program, check the result

1. public class stack19 {
2. int size;
3. int top;
4. book19 data[];
6. public stack19(int size) {
7. this.size = size;
8. data = new book19[size];
9. top = -1;
10. }
11. public boolean IsEmpty(){
12. return top == -1;
13. }
14. public boolean IsFull() {
15. return top == size - 1;
16. }
17. public void push(book19 dt) {
18. if (!IsFull()) {
19. top++;
20. data[top] = dt;
21. } else {
22. System.out.println("Stack is full. Cannot push element.");
23. }
24. }
25. public void pop() {
26. if (!IsEmpty()) {
27. book19 x = data[top];
28. top--;
29. System.out.println("Removed data: " + x.title + " " + x.authorName + " " + x.publishedYear + " " + x.pagesAmount + " " + x.price);
30. } else {
31. System.out.println("Stack is empty. Cannot pop element.");
32. }
33. }
34. public void peek(){
35. if (!IsEmpty()) {
36. System.out.println("Top element: " + data[top].title + " " + data[top].authorName + " " + data[top].publishedYear + " " + data[top].pagesAmount + " " + data[top].price);
37. } else {
38. System.out.println("Stack is empty. No top element.");
39. }
40. }
41. public void print() {
42. System.out.println("Stack content:");
43. for (int i = top; i >= 0; i--) {
44. System.out.println(data[i].title + " " + data[i].authorName + " " + data[i].publishedYear + " " + data[i].pagesAmount + " " + data[i].price);
45. }
46. System.out.println();
47. }
48. public void clear(){
49. top = -1; // Reset top to indicate an empty stack
50. System.out.println("Stack is now empty");
51. }
52. }
53. // /\*\*
54. //  \* stack19
55. //  \*/
56. // public class stack19 {
57. //     int size;
58. //     int top;
59. //     int push;
60. //     book19 data[];
62. //     public stack19(int size) {
63. //         this.size = size;
64. //         data = new book19[size];
65. //         top = -1;
66. //     }
67. //     public boolean IsEmpty(){
68. //         if (top == -1){
69. //         return true;
70. //         }else {
71. //         return false;
72. //         }
73. //     }
74. //     public boolean IsFull() {
75. //         return top == size - 1;
76. //     }
78. //     // public boolean IsFull(){
79. //     //     if (top == size) {
80. //     //         return true;
81. //     //     }else{
82. //     //         return false;
83. //     //     }
84. //     // }
85. //     public void push(book19 dt) {
86. //         if (!IsFull()) {
87. //             top++;
88. //             data[top] = dt;
89. //         } else {
90. //             System.out.println("Stack is full. Cannot push element.");
91. //         }
92. //     }
93. //     // public void push (book19 dt ){
94. //     //     if (!IsFull()) {
95. //     //         top++;
96. //     //         data[top] = dt;
97. //     //     }
98. //     // }
99. //     public void pop() {
100. //         if (!IsEmpty()) {
101. //             book19 x = data[top];
102. //             top--;
103. //             System.out.println("Removed data: " + x);
104. //         } else {
105. //             System.out.println("Stack is empty. Cannot pop element.");
106. //         }
107. //     }
109. //     // public void pop(){
110. //     //     if (!IsEmpty()) {
111. //     //         book19 x = data[top];
112. //     //         top--;
113. //     //         System.out.println("Remove data : " + x);
114. //     //     }else{
115. //     //         System.out.println("Stack is empty");
116. //     //     }
117. //     // }
118. //     public void peek(){
119. //         System.out.println("Top element : " +data[top]);
120. //     }
121. //     public void print() {
122. //         System.out.println("Stack content:");
123. //         for (int i = top; i >= 0; i--) {
124. //             System.out.println(data[i] + " ");
125. //         }
126. //         System.out.println();
127. //     }
129. //     // public void print (){
130. //     //     System.out.println("Stack content: ");
131. //     //     for (int i = top; i >- 0; i--) {
132. //     //         System.out.println(data[i] + " ");
133. //     //     }
134. //     //     System.out.println("");
135. //     // }
136. //     public void clear(){
137. //         if (!IsEmpty()) {
138. //             for (int i =top; i >= 0; i--) {
139. //                 top--;
140. //             }
141. //             System.out.println("Stack is now empty");
142. //         }else{
143. //             System.out.println("Failed ! Stack is still empty ");
144. //         }
145. //     }
146. // }

public class stackMain19 {

    public static void main(String[] args) {

        stack19 stk = new stack19(5);

        // Creating book19 objects and pushing them into the stack

        book19 book1 = new book19("Title1", "Author1", 2020, 300, 20);

        book19 book2 = new book19("Title2", "Author2", 2019, 250, 15);

        book19 book3 = new book19("Title3", "Author3", 2018, 400, 25);

        stk.push(book1);

        stk.push(book2);

        stk.push(book3);

        stk.print();

        stk.pop();

        stk.peek();

        stk.print();

    }

        // stack19 stk = new stack19(5);

    // stk.push(15);

    // stk.push(27);

    // stk.push(13);

    // stk.print();

    // stk.push(11);

    // stk.push(34);

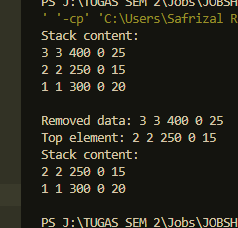
    // stk.pop();

    // stk.peek();

    // stk.print();

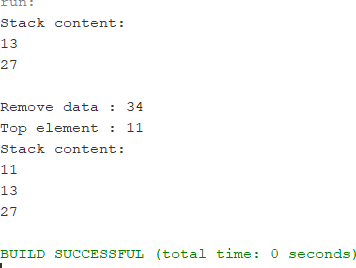
    // }

}



* + 1. Result

Check if the result match with following image:

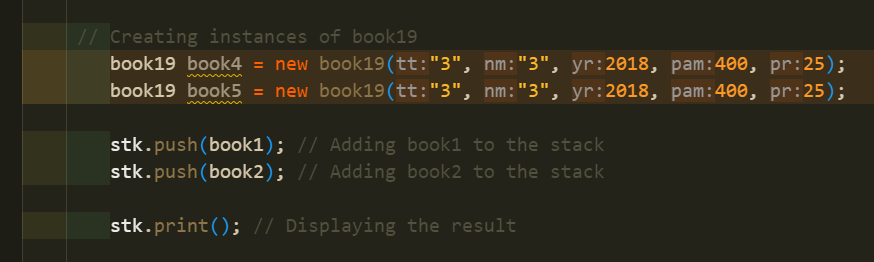


* + 1. Questions
       1. In class **StackMain,** what is the usage of number 5 in this following code?

**Within the course StackMain, the number 5 speaks to the size of the stack that's being made. Once you compose Stack stk = unused Stack(5);, you're making a unused Stack protest named stk with a capacity of 5 components.**



* + - 1. Add 2 more data in the stack with 18 and 40. Display the result!



* + - 1. In previous number, the data inserted in to the stack is only 18 and 40 is not inserted. Why is that?

**The reason why as it were the information "18" is embedded into the stack and "40" isn't embedded is since of the information sort bungle.**

**In your code piece, you've announced the stack to hold objects of type book19. In any case, after you attempt to thrust integrability onto the stack with stk.push(18) and stk.push(40), you experience a compilation mistake since 18 and 40 are integrability, not objects of sort book19.**

**To insert data into the stack successfully, you wish to make occasions of book19 and after that thrust them onto the stack, as appeared within the redressed code example in my past reaction. In the event that you need to embed the integrability "18" and "40", you wish to alter the information sort of the stack to hold integrability rather than book19.**

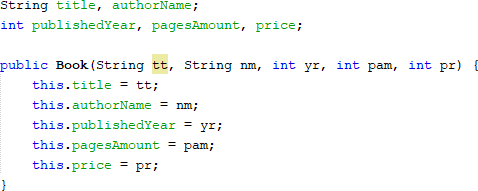
* 1. 2nd Lab Activities

In this practicum, we will create a program to illustrate a bunch of books that are stored in Stack. Since the book has some information on it, the stack implementation is done using array of object to represent each element.

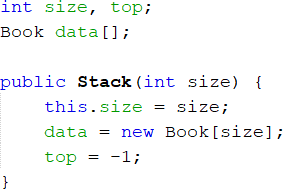
* + 1. Steps
       1. This class diagram is used for creating a program code written in Java programming language

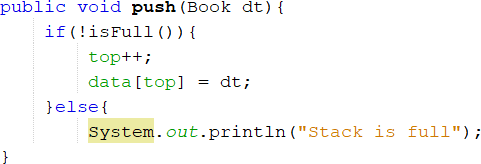
|  |
| --- |
| Book |
| title: String authorName: String publishedYear: int pagesAmount: int  price: int |
| Book(title: String, author: String, publishedYear: int,  pagesAmount: int, price: int) |

* + - 1. Create a new package named **Practicum2,** then create a new class named **Book.**
      2. Add attributes in that class, and add the constructor as well.

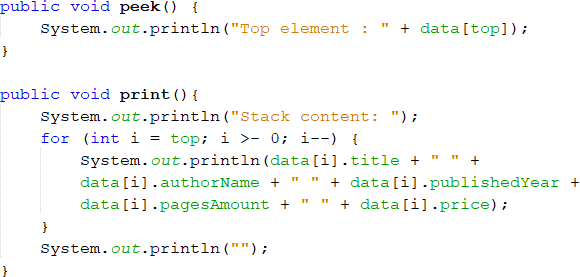
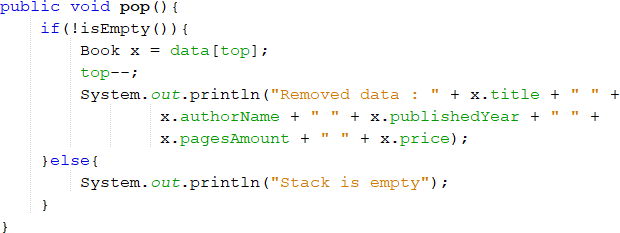


* + - 1. Copy the program code for **Stack** class in **Practicum1** to be used again in here. Since the data stored in Stack in **Practicum1** is integer array, and in **Practicum2** we use objects, we will need to modify some parts in that class.
      2. Modify the **Stack** class by changing the data type of **int data[]** to **Book data[].** This time we will need to save the data in stack in objects. In addition, we will need to change the **attributes**, **constructor**, **method push**, and **method pop**





* + - 1. We will need to change the **print, pop, and peek method** as well since the data that are going to be printed is not only a string, but an object consists of some information (title, authorName, etc.).



* + - 1. Next, we have to create a new class called **StackMain** in **Practicum2**. Create a main function and instantiate an object with named **st**
      2. Declare the **Scanner** object with name **sc**
      3. Insert these lines of codes to receive **Book** data input, alongside with its information to be stored in stack



* + - 1. Call print, pop, and peek method accordingly as follows:



* + - 1. Compile and run **StackMain**, and observe the result

/\*\*

 \* book19

 \*/

public class book19 {

    String title, authorName;

    int publishedYear, pagesAmount, price;

    public book19(String tt, String nm, int yr, int pam, int pr){

        this.title = tt;

        this.authorName= nm;

        this.publishedYear = pam;

        this.price = pr;

    }

    int size,top;

    book19 data[];

    public void stack19 (int size){

        this.size = size;

        data = new book19[size];

        top = -1 ;

    }

    public boolean IsEmpty(){

        if (top == -1){

        return true;

        }else {

        return false;

        }

    }

    public boolean IsFull(){

        if (top == size) {

            return true;

        }else{

            return false;

        }

    }

    public void push (book19 dt){

        if (!IsFull()) {

            top++;

            data[top] = dt ;

        }else {

            System.out.println("Stack is full");

        }

    }

    public void pop(){

        if (!IsEmpty()) {

            book19 x = data[top];

            top--;

            System.out.println("Removed data : " + x.title + " "+

             x.authorName + " " + x.publishedYear + " " +

             x.pagesAmount + " " + x.price );

        }else {

            System.out.println("Top element : " + data[top]);

        }

    }

    public void peek(){

        System.out.println("Top element : " +data[top]);

    }

    public void print() {

        System.out.println("Stack content:");

        for (int i = top; i >= 0; i--) {

            System.out.println("Title: " + data[i].title + " " +

            data[i].authorName + " " + data[i].publishedYear +

            data[i].pagesAmount + " " + data[i].price);

        }

        System.out.println("");

    }

}

import java.util.Scanner;

public class bookMain19 {

    public static void main(String[] args) {

        stack19 st = new stack19(8);

        Scanner sc19 = new Scanner(System.in);

        char choose;

        do {

            System.out.print("Title: ");

            String title = sc19.nextLine();

            System.out.print("Author name: ");

            String name = sc19.nextLine();

            System.out.print("Published year: ");

            int year = sc19.nextInt();

            System.out.print("Pages Amount: ");

            int pages = sc19.nextInt();

            System.out.print("Price: ");

            int price = sc19.nextInt();

            book19 bk = new book19(title, name, year, pages, price);

            System.out.println("Do you want to add new data to the stack (y/n)? ");

            choose = sc19.next().charAt(0);

            sc19.nextLine();

            st.push(bk); // Pushing a book19 object

        } while (choose == 'y');

        st.print();

        st.pop();

        st.peek();

        st.print();

    }

    // public static void main(String[] args) {

    // stack19 st = new stack19(8);

    // Scanner sc19 = new Scanner(System.in);

    // char choose ;

    // do {

    //     System.out.print("Title : ");

    //     String title = sc19.nextLine();

    //     System.out.print("Author name : ");

    //     String name = sc19.nextLine();

    //     System.out.print("Published year : ");

    //     int year = sc19.nextInt();

    //     System.out.print("Pages Amount : ");

    //     int pages = sc19.nextInt();

    //     System.out.print("Price : ");

    //     int price = sc19.nextInt();

    //     book19 bk = new book19(title, name, year, pages, price);

    //     System.out.println("Do you want add new data to stack (y/n)? ");

    //     choose = sc19.next().charAt(0);

    //     sc19.nextLine();

    //     st.push(bk);

    // } while (choose == 'y');

    // st.print();

    // st.pop();

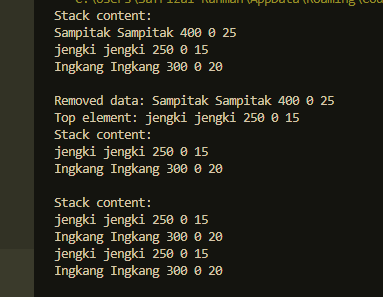
    // st.peek();

    // st.print();

    }

* + 1. Result

Check if the result match with following image:





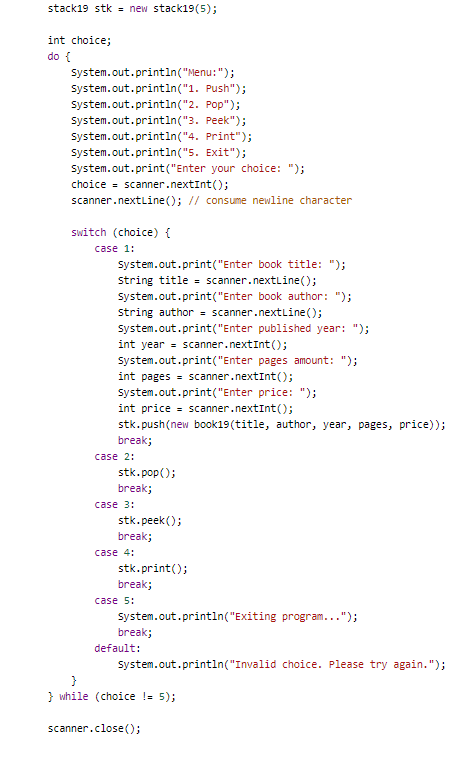
* + 1. Questions
       1. In class StackMain, when calling **push** method, the argument is **bk.** What information is included in the **bk** variable?

**The variable bk likely holds an question of sort book19. The precise data included in bk would depend on how it's instantiated. Ordinarily, a book19 protest might contain information such as the title, creator, distribution date, etc., depending on how the book19 course is characterized.**

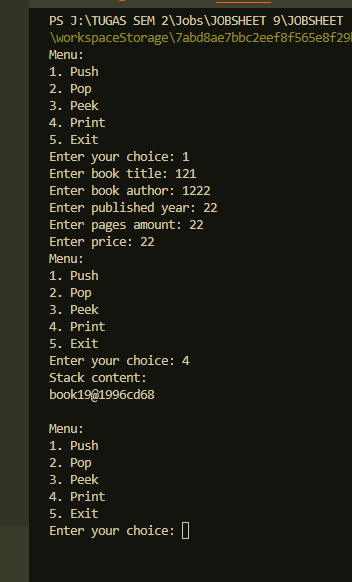
* + - 1. Which of the program that its usage is to define the capacity of the stack ?  
         **The program that characterizes the capacity of the stack is the one where the stack is initialized. In your code scrap, it appears the capacity is characterized when making a unused occurrence of the stack19 lesson,**
      2. stack19 stk = unused stack19(5);. Here, 5 indicates the capacity of the stack to be 5.

**The do-while circle within the StackMain class is likely utilized to supply a menu-driven interface for collaboration with the stack. It over and over shows a menu of alternatives (e.g., push, pop, look, print) and prompts the client to choose an operation until the client chooses to exit or end the circle.**

* + - 1. What is the function of do-while that is exist in **StackMain** class?



* + - 1. Modify the program in **StackMain,** so that the user may choose which operation (push, pop, peek, print) to do in stack from program menu!



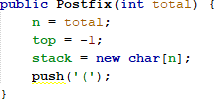
* 1. 3rd Lab Activities

In this practicum, we will create program to convert infix notation into postfix notation

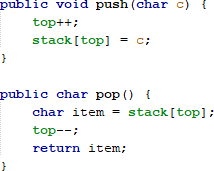
* + 1. Steps
       1. We will use class diagram to create **Postfix** class in Java program

|  |
| --- |
| Postfix |
| n: int top: int  stack: char[] |
| Postfix(total: int) push(c: char): void pop(): void  IsOperand(c: char): boolean IsOperator(c: char): boolean degree(c: char): int  convert(Q: String): string |

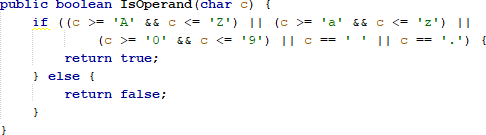
* + - 1. Create a package named **Practicum3.** Then, we create a new class named **Postfix.** Add attributes **n, top, and stack** based on class diagram above.
      2. Add a constructor with parameter as follows:



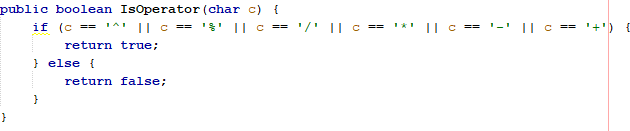
* + - 1. Create method **push** and **pop** with void as its return type



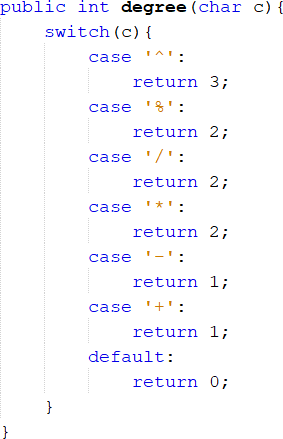
* + - 1. Create method **isOperand** as Boolean that will be used to check if the element is operand or not



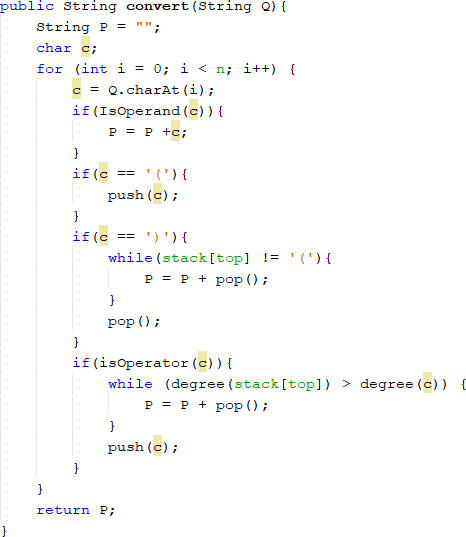
* + - 1. Create method **isOperator** as booelan that will be used to check if the element is operator or not



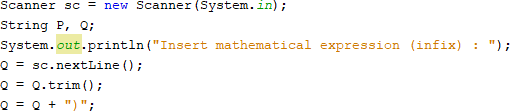
* + - 1. Create method **degree** as integer to define the degree of the operator



* + - 1. Create method **convert** to convert infix notation to postfix notation by checking the element one by one in data element.



* + - 1. Next, we will need create a class named **PostfixMain.** After creating the main function, we create a variable P and Q. P variable will be used to store the final result of converted postfix notation, while Q variable is used to store user input in the form mathematical expression written in infix notation. Instantiate the Scanner object with **sc** variable, then call build-in **trim** method to remove spaces within a string.



We need to add string **“)”** to ensure all symbol/ characters that are exist in the stack will be retrieved and moved in postfix.

* + - 1. Create a **total** variable to calculate how many characters in variable Q



* + - 1. Instantiate object **post** with **total** as the argument. Then, call **convert** method to change the infix notation in Q string to postfix notation P



* + - 1. Compile and run **StackMain**, and observe the result

package Practicum3;

import java.util.Scanner;

public class PostfixMain {

    public static void main(String[] args) {

        Scanner sUcaP19 = new Scanner(System.in);

        String P, Q;

        System.out.println("Insert mathematical expression (infix) : ");

        Q = sUcaP19.nextLine();

        Q = Q.trim();

        Q += ' ';

        int total  = Q.length();

        Postflix post = new Postflix(total);

        P = post.convert(Q);

        System.out.println("Postfix: " + P);

    }

}

package Practicum3;

public class Postflix {

    private int n;

    private int top;

    private char[] stack;

    // Constructor

    public Postflix(int total) {

        n = total;

        top = -1;

        stack = new char[n];

        push('(');

    }

    public void push(char c) {

        if (top == n - 1) {

            System.out.println("Stack Overflow");

            return;

        }

        top++; // Menambahkan top terlebih dahulu

        stack[top] = c; // Menambahkan elemen ke tumpukan

    }

    // public void push(char c) {

    //     if (top == n - 1) {

    //         top++;

    //         stack[top] = c;

    //         System.out.println("Stack Overflow");

    //         return;

    //     }

    // }

    public char pop() {

        if (top == -1) {

            System.out.println("Stack Underflow");

            return '\0'; // Mengembalikan nilai karakter kosong jika tumpukan kosong

        } else {

            char item = stack[top];

            top--;

            return item;

        }

    }

    // public char pop() {

    //     char item = stack[top];

    //     top--;

    //     return item;

    // }

    public boolean isOperand(char c) {

        if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z') ||

            (c >= '0' && c <= '9') || c == '\_' || c == '.') {

            return true;

        } else {

            return false;

        }

    }

    public boolean isOperator(char c) {

        if (c == '^' || c == '%' || c == '/' || c == '\*' || c == '-' || c == '+') {

            return true;

        } else {

            return false;

        }

    }

    public int degrees(char c) {

        switch (c) {

            case '^':

                return 3;

            case '%':

                return 2;

            case '/':

                return 2;

            case '\*':

                return 2;

            case '-':

                return 1;

            case '+':

                return 1;

            default:

                return 0;

        }

    }

    public String convert(String Q) {

        String P = "";

        char c;

        for (int i = 0; i < Q.length(); i++) {

            c = Q.charAt(i);

            if (isOperand(c)) {

                P = P + c;

            } else if (c == '(') {

                push(c);

            } else if (c == ')') {

                while (top != -1 && stack[top] != '(') {

                    P = P + pop();

                }

                if (top != -1 && stack[top] == '(') {

                    pop(); // Hapus '(' dari tumpukan

                } else {

                    System.out.println("Error: Kurung tidak seimbang");

                    return ""; // Mengembalikan string kosong karena kurung tidak seimbang

                }

            } else if (isOperator(c)) {

                while (top != -1 && degrees(stack[top]) >= degrees(c)) {

                    P = P + pop();

                }

                push(c);

            }

        }

        // Pastikan tumpukan kosong setelah membaca seluruh karakter

        while (top != -1 && stack[top] != '(') {

            P = P + pop();

        }

        return P;

    }

    // public String convert(String Q) {

    //     String P = "";

    //     char c;

    //     for (int i = 0; i < Q.length(); i++) {

    //         c = Q.charAt(i);

    //         if (isOperand(c)) {

    //             P = P + c;

    //         }

    //         if (c == '(') {

    //             push(c);

    //         }

    //         if (c == ')') {

    //             while(stack[top] != '(') {

    //                 P = P + pop();

    //             }

    //             pop(); // remove '(', not used anymore

    //         }

    //         if(isOperator(c)) {

    //             while(degrees(stack[top]) >= degrees(c)) {

    //                 P = P + pop();

    //             }

    //             push(c);

    //         }

    //     }

    //     return P;

    // }

    // public String convert(String Q) {

    //     String P = "";

    //     char c;

    //     for (int i = 0; i < Q.length(); i++) {

    //         c = Q.charAt(i);

    //         if (isOperand(c)) {

    //             P = P + c;

    //         }

    //         if (c == '(') {

    //             push(c);

    //         }

    //         if (c == ')') {

    //             while(stack[top] != '(') {

    //                 P = P + pop();

    //             }

    //             pop(); // remove '(', not used anymore

    //         }

    //         }

    //         if(isOperator(c)) {

    //             while(degrees(stack[top]) >= degrees(c)) {

    //                 P = P + pop();

    //             }

    //             push(c);

    //         }

    //     return P;

    // }

}

// public class Postflix {

//     private int n;

//     private int top;

//     private char[] stack;

//     // Constructor

//     public Postflix(int total) {

//         n = total;

//         top = -1;

//         stack = new char[n];

//         push('(');

//     }

//     public void push(char c) {

//         if (top == n - 1) {

//             top++;

//             stack[top] = c;

//             System.out.println("Stack Overflow");

//             return;

//         }

//     }

//     public char pop() {

//         char item = stack[top];

//         top--;

//         return item;

//     }

//     public boolean isOperand(char c) {

//         if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z') ||

//             (c >= '0' && c <= '9') || c == '\_' || c == '.' ) {

//             return true;

//         } else {

//             return false;

//         }

//     }

//     public boolean isOperator(char c) {

//         if (c == '^' || c == '%' || c == '/' || c == '\*' || c == '-' || c == '+') {

//             return true;

//         } else {

//             return false;

//         }

//         public int degrees(char c) {

//             switch(c) {

//                 case '^':

//                     return 3;

//                 case '%':

//                     return 2;

//                 case '/':

//                     return 2;

//                 case '\*':

//                     return 2;

//                 case '-':

//                     return 1;

//                 case '+':

//                     return 1;

//                 default:

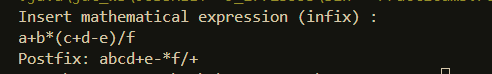
//                     return 0;

//             }

//         }

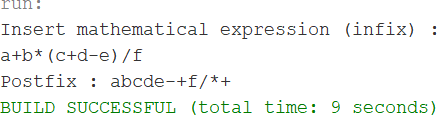
//     }

// }



* + 1. Result

Check if the result match with following image:



* + 1. Questions
       1. Please explain the flow of method in **Postfix** class

**Constructor (Postflix(int total)): Initializes the stack with a size of total and pushes an opening parenthesis '(' onto the stack.**

**push(char c): Adds an element c to the stack if there is space available. If the stack is full, it prints "Stack Overflow".**

**pop(): Removes and returns the top element of the stack if the stack is not empty. If the stack is empty, it prints "Stack Underflow".**

**isOperand(char c): Checks if the character c is an operand (letter, digit, underscore, or dot) and returns true if it is, false otherwise.**

**isOperator(char c): Checks if the character c is an operator (^, %, /, \*, -, or +) and returns true if it is, false otherwise.**

**degrees(char c): Returns the precedence level of the operator c.**

**convert(String Q): Converts the infix expression Q to postfix notation. It iterates through each character of the input expression, processing operands, operators, and parentheses according to the rules of converting infix to postfix. It utilizes the push, pop, isOperand, isOperator, and degrees methods to handle the conversion process.**

* + - 1. What is the function of this program code?

**This line of code retrieves the character at index i from the string Q and assigns it to the variable c. It allows the program to access each character of the input expression Q individually for processing.**



* + - 1. Execute the program again, how’s the result if we insert **3\*5^(8-6)%3** for the expression?

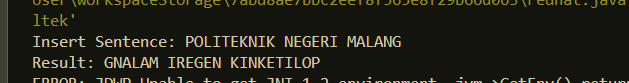
**When executing the program with this expression, it should convert the infix expression to postfix notation and print the resulting postfix expression. The % operator has a lower precedence than ^, \*, and /, so the resulting postfix expression should be 35586-^3%\*.**

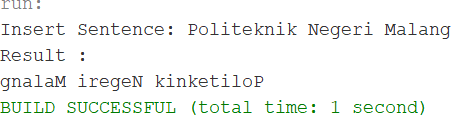
* + - 1. In 2nd number, why the braces are not displayed in conversion result? Please explain

**In the original implementation of the convert method, when encountering an opening parenthesis '(', it is pushed onto the stack. However, when encountering a closing parenthesis ')', it is immediately removed from the stack, and the conversion process continues without adding it to the postfix expression. This behavior causes the braces not to be displayed in the conversion result. To include the braces in the conversion result, modifications to the convert method are needed to properly handle opening and closing parentheses.**

* 1. Assignment
     1. Create a program with Stack implementation to insert a sentence and display the reversed version of the sentence as a result!

1. package Practicum3;
2. import java.util.Scanner;
3. import java.util.Stack;
4. public class poltek {
5. public static void main(String[] args) {
6. Scanner scanner = new Scanner(System.in);
7. System.out.print("Insert Sentence: ");
8. String sentence = scanner.nextLine();
10. String reversedSentence = reverseSentence(sentence);
12. System.out.println("Result: " + reversedSentence);
13. }
15. public static String reverseSentence(String sentence) {
16. Stack<Character> stack = new Stack<>();
18. // Push each character of the sentence onto the stack
19. for (int i = 0; i < sentence.length(); i++) {
20. stack.push(sentence.charAt(i));
21. }
23. // Pop each character from the stack to construct the reversed sentence
24. StringBuilder reversed = new StringBuilder();
25. while (!stack.isEmpty()) {
26. reversed.append(stack.pop());
27. }
29. return reversed.toString();
30. }
31. }





* + 1. Every Sunday, Dewi shops to a supermarket that is in her residential area. Everytime she finishes, she keeps the receipt of what she has bought in a wardrobe. After 2 months, She had 8 receipts. She plans to trade her 5 receipts in exchange for a voucher.

Create a program using stack implementation to store Dewi’s receipt. As well as the retrieving the receipts. The information that are included in a receipt are as follows:

* + - * Transaction ID
      * Date
      * Quantity of items
      * Total price

package Practicum3;

import java.util.Scanner;

import java.util.Stack;

public class DewiReceipts {

    public static void main(String[] args) {

        Stack<Receipt> receiptStack = new Stack<>();

        Scanner scanner = new Scanner(System.in);

        // Simulate adding receipts to the stack

        for (int i = 1; i <= 8; i++) {

            System.out.println("Enter details for Receipt " + i + ":");

            System.out.print("Transaction ID: ");

            String transactionId = scanner.nextLine();

            System.out.print("Date: ");

            String date = scanner.nextLine();

            System.out.print("Quantity of Items: ");

            int quantity = Integer.parseInt(scanner.nextLine());

            System.out.print("Total Price: ");

            double totalPrice = Double.parseDouble(scanner.nextLine());

            Receipt receipt = new Receipt(transactionId, date, quantity, totalPrice);

            receiptStack.push(receipt);

        }

        // Retrieve and display the receipts

        System.out.println("\nRetrieving Receipts:");

        while (!receiptStack.isEmpty()) {

            Receipt receipt = receiptStack.pop();

            System.out.println(receipt);

        }

    }

}

package Practicum3;

import java.util.Scanner;

import java.util.Stack;

public class Receipt {

    String transactionId;

    String date;

    int quantity;

    double totalPrice;

    public Receipt(String transactionId, String date, int quantity, double totalPrice) {

        this.transactionId = transactionId;

        this.date = date;

        this.quantity = quantity;

        this.totalPrice = totalPrice;

    }

    @Override

    public String toString() {

        return "Transaction ID: " + transactionId + ", Date: " + date + ", Quantity: " + quantity + ", Total Price: " + totalPrice;

    }

}

