SAFRIZAL

RAHMAN

19 – SIB 1G

https://www.youtube.com/watch?v=dQw4w9WgXcQ

https://github.com/safrizalrahman46/Jobsheet9-SEM-2

JOBSHEET IX

STACK

1.1. 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

1.2. Lab Activities

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

1.2.1. 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.

2. Create a new project named **Jobsheet7.** Create a new package with name **Practicum1.**

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

3. Create new attributes size, top, and data as follows:

```
int size;
int top;
int data[];
```

4. Add a constructor with parameter as written below:

```
public Stack(int size) {
    this.size = size;
    data = new int[size];
    top = -1;
}
```

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

```
public boolean IsEmpty() {
    if (top == -1) {
        return true;
    } else {
        return false;
    }
}
```

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

```
public boolean IsFull() {
    if (top == size - 1) {
        return true;
    } else {
        return false;
    }
}
```

7. 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

```
public void push(int dt){
    if(!isFull()){
        top++;
        data[top] = dt;
    }else{
        System.out.println("Stack is full");
}
```

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

```
public void pop(){
    if(!isEmpty()){
        int x = data[top];
        top--;
        System.out.println("Remove data : " + x);
    }else{
        System.out.println("Stack is empty");
    }
}
```

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

```
public void peek() {
    System.out.println("Top element : " + data[top]);
}
```

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

```
public void print(){
    System.out.println("Stack content: ");
    for (int i = top; i >= 0; i--) {
        System.out.println(data[i] + " ");
    }
    System.out.println("");
}
```

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

```
public void clear(){
    if(!isEmpty()){
        for (int i = top; i >= 0; i--) {
            top--;
        }
        System.out.println("Stack is now empty");
    }else{
        System.out.println("Failed ! Stack is still empty ");
    }
}
```

12. 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**

```
Stack stk = new Stack(5);
```

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

```
stk.push(15);
stk.push(27);
stk.push(13);
```

14. Display the data that we've inserted in previous step by calling method **print**

```
stk.print();
```

15. 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**

```
stk.push(11);
stk.push(34);
stk.pop();
stk.peek();
stk.print();
```

16. Compile and run the program, check the result

```
2  public class stack19 {
3
4      int size;
5      int top;
6      book19 data[];
7
8      public stack19(int size) {
9          this.size = size;
10         data = new book19[size];
11         top = -1;
12     }
13
14     public boolean IsEmpty(){
15         return top == -1;
16     }
```

```
17
18      public boolean IsFull() {
19          return top == size - 1;
20      }
21
22      public void push(book19 dt) {
23          if (!IsFull()) {
24              top++;
25              data[top] = dt;
26          } else {
27              System.out.println("Stack is full. Cannot push
    element.");
28          }
29      }
30
31      public void pop() {
32          if (!IsEmpty()) {
33              book19 x = data[top];
34              top--;
35              System.out.println("Removed data: " + x.title +
    " " + x.authorName + " " + x.publishedYear + " " +
    x.pagesAmount + " " + x.price);
36          } else {
37              System.out.println("Stack is empty. Cannot pop
    element.");
38          }
39      }
40
41      public void peek(){
42          if (!IsEmpty()) {
43              System.out.println("Top element: " +
    data[top].title + " " + data[top].authorName + " " +
    data[top].publishedYear + " " + data[top].pagesAmount + " "
    + data[top].price);
44          } else {
45              System.out.println("Stack is empty. No top
    element.");
```

```
        }
    }

    public void print() {
        System.out.println("Stack content:");
        for (int i = top; i >= 0; i--) {
            System.out.println(data[i].title + " " +
  data[i].authorName + " " + data[i].publishedYear + " " +
  data[i].pagesAmount + " " + data[i].price);
        }
        System.out.println();
    }

    public void clear(){
        top = -1; // Reset top to indicate an empty stack
        System.out.println("Stack is now empty");
    }
}


// /**
//  * stack19
//  */
// public class stack19 {

//     int size;
//     int top;
//     int push;
//     book19 data[];

//     public stack19(int size) {
//         this.size = size;
//         data = new book19[size];
//         top = -1;
//     }

//     public boolean IsEmpty(){
```

```
80//         if (top == -1){
81//         return true;
82//         }else {
83//         return false;
84//         }
85//     }
86
87//     public boolean IsFull() {
88//         return top == size - 1;
89//     }
90
91//     // public boolean IsFull(){
92//     //     if (top == size) {
93//     //         return true;
94//     //     }else{
95//     //         return false;
96//     //     }
97//     // }
98
99//     public void push(book19 dt) {
100 //         if (!IsFull()) {
101 //             top++;
102 //             data[top] = dt;
103 //         } else {
104 //             System.out.println("Stack is full. Cannot
  push element.");
105 //         }
106 //     }
107 //     // public void push (book19 dt ){
108 //     //     if (!IsFull()) {
109 //     //         top++;
110 //     //         data[top] = dt;
111 //     //     }
112 //     // }
113
114 //     public void pop() {
115 //         if (!IsEmpty()) {
```

```
116 //             book19 x = data[top];
117 //             top--;
118 //             System.out.println("Removed data: " + x);
119 //         } else {
120 //             System.out.println("Stack is empty. Cannot
  pop element.");
121 //         }
122 //     }
123 
124 //     // public void pop(){
125 //     //     if (!IsEmpty()) {
126 //     //         book19 x = data[top];
127 //     //         top--;
128 //     //         System.out.println("Remove data : " +
  x);
129 //     //     }else{
130 //     //         System.out.println("Stack is empty");
131 //     //     }
132 //     // }
133 
134 //     public void peek(){
135 //         System.out.println("Top element : "
  +data[top]);
136 //     }
137 
138 //     public void print() {
139 //         System.out.println("Stack content:");
140 //         for (int i = top; i >= 0; i--) {
141 //             System.out.println(data[i] + " ");
142 //         }
143 //         System.out.println();
144 //     }
145 
146 //     // public void print (){
147 //     //     System.out.println("Stack content: ");
148 //     //     for (int i = top; i >- 0; i--) {
149 //     //         System.out.println(data[i] + " ");
```

```
150 //      //      }
151 //      //      System.out.println("");
152 //      // }
153
154 //      public void clear(){
155 //          if (!IsEmpty()) {
156 //              for (int i =top; i >= 0; i--) {
157 //                  top--;
158 //              }
159 //              System.out.println("Stack is now empty");
160 //          }else{
161 //              System.out.println("Failed ! Stack is
  still empty ");
162 //          }
163 //      }
164 // }
165
```

```
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();
    // }

}
```

```
PS J:\TUGAS SEM 2\Jobs\JOBSH
' '-cp' 'C:\Users\Safrizal R
Stack content:
3 3 400 0 25
2 2 250 0 15
1 1 300 0 20

Removed data: 3 3 400 0 25
Top element: 2 2 250 0 15
Stack content:
2 2 250 0 15
1 1 300 0 20

PS J:\TUGAS SEM 2\Jobs\JOBSH
```

165.1.1. Result

Check if the result match with following image:

```
run:
Stack content:
13
27

Remove data : 34
Top element : 11
Stack content:
11
13
27

BUILD SUCCESSFUL (total time: 0 seconds)
```

165.1.2. 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.

```
Stack stk = new Stack(5);
```

2. Add 2 more data in the stack with 18 and 40. Display the result!

```
// Creating instances of book19
book19 book4 = new book19(tt:"3", nm:"3", yr:2018, pam:400, pr:
book19 book5 = new book19(tt:"3", nm:"3", yr:2018, pam:400, pr:

stk.push(book1); // Adding book1 to the stack
stk.push(book2); // Adding book2 to the stack

stk.print(); // Displaying the result
```

3. 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.

165.2. 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.

165.2.1. 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)

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

```
String title, authorName;
int publishedYear, pagesAmount, price;

public Book(String tt, String nm, int yr, int pam, int pr) {
    this.title = tt;
    this.authorName = nm;
    this.publishedYear = yr;
    this.pagesAmount = pam;
    this.price = pr;
}
```

4. 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.

5. 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**

```
int size, top;
Book data[];

public Stack(int size) {
    this.size = size;
    data = new Book[size];
    top = -1;
}

public void push(Book dt){
    if(!isFull()){
        top++;
        data[top] = dt;
    }else{
        System.out.println("Stack is full");
}
```

6. 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.).

```
public void pop(){
    if(!isEmpty()){
        Book x = data[top];
        top--;
        System.out.println("Removed data : " + x.title + " " +
                x.authorName + " " + x.publishedYear + " " +
                x.pagesAmount + " " + x.price);
    }else{
        System.out.println("Stack is empty");
    }
}
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(data[i].title + " " +
        data[i].authorName + " " + data[i].publishedYear +
        data[i].pagesAmount + " " + data[i].price);
    }
    System.out.println("");
}
```

7. Next, we have to create a new class called **StackMain** in **Practicum2**. Create a main function and instantiate an object with named **st**

8. Declare the **Scanner** object with name **sc**

9. Insert these lines of codes to receive **Book** data input, alongside with its information to be stored in stack

```
Stack st = new Stack(8);
Scanner sc = new Scanner(System.in);

char choose;
do {
    System.out.print("Title : ");
    String title = sc.nextLine();

    System.out.print("Author Name : ");
    String name = sc.nextLine();

    System.out.print("Published year : ");
    int year = sc.nextInt();

    System.out.print("Pages Amount: ");
    int pages = sc.nextInt();

    System.out.print("Price: ");
    int price = sc.nextInt();

    Book bk = new Book(title, name, year, pages, price);
    System.out.print("Do you want to add new data to stack (y/n)? ");
    choose = sc.next().charAt(0);
    sc.nextLine();
    st.push(bk);

} while (choose == 'y');
```

10. Call print, pop, and peek method accordingly as follows:

```
st.print();
st.pop();
st.peek();
st.print();;
```

11. 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 ;
```

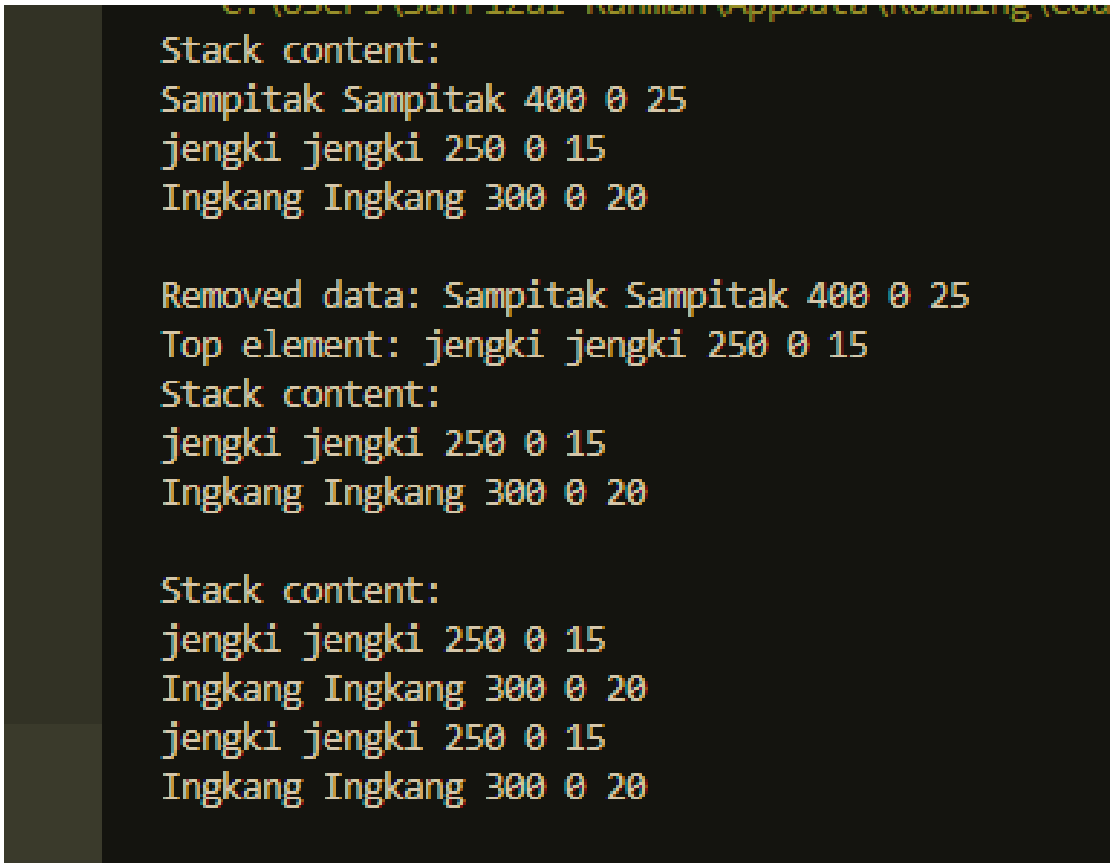
```
        }else {
            System.out.println("Stack is full");
        }
    }

    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();
    }
```

165.2.2. Result

Check if the result match with following image:



```
run:
Title : Programming
Author Name : Burhantoro
Published year : 2016
Pages Amount: 126
Price: 58000
Do you want to add new data to stack (y/n)? y
Title : Statistics
Author Name : Yasir
Published year : 2014
Pages Amount: 98
Price: 44000
Do you want to add new data to stack (y/n)? y
Title : Economics
Author Name : Diana
Published year : 2019
Pages Amount: 86
Price: 47500
Do you want to add new data to stack (y/n)? n
Stack content:
Economics Diana 201986 47500
Statistics Yasir 201498 44000

Removed data : Economics Diana 2019 86 47500
Top element : Stack.Book@55f96302
Stack content:
Statistics Yasir 201498 44000

BUILD SUCCESSFUL (total time: 1 minute 5 seconds)
```

165.2.3. Questions

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

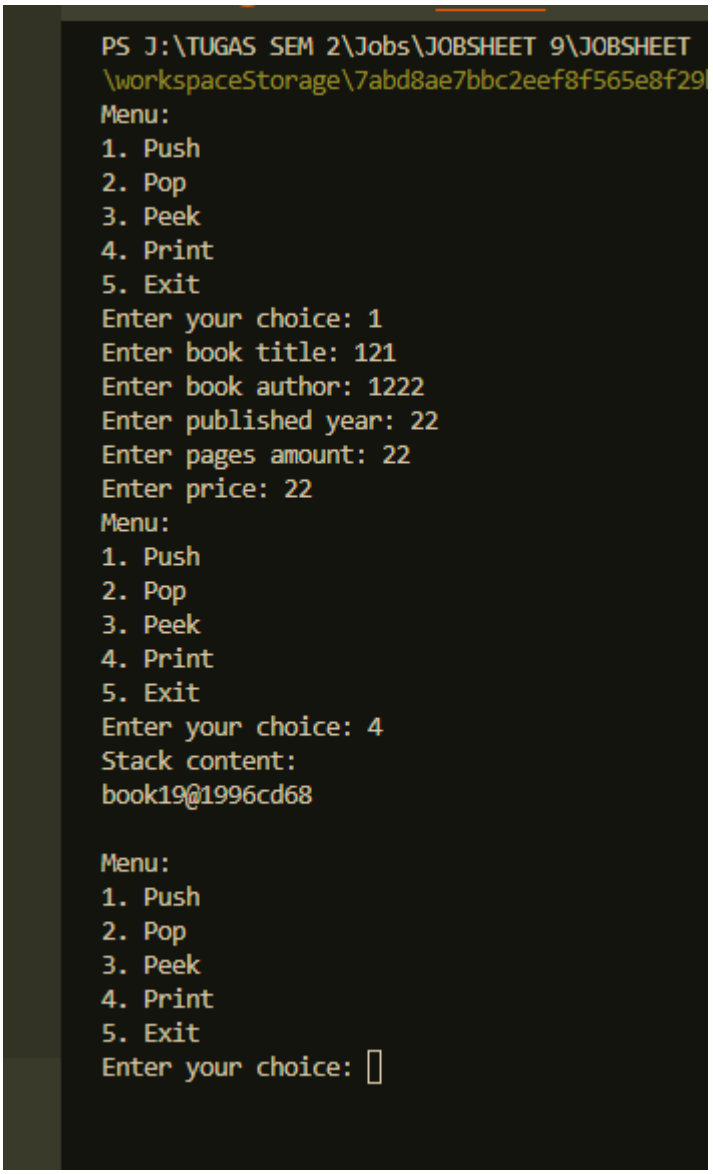
The variable bk likely holds an question of sort book19. The precise data included in bk would depend on how it's instantiated. Ordinarily, a book19 protest might contain information such as the title, creator, distribution date, etc., depending on how the book19 course is characterized.

2. 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,
3. 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.

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

```
stack19 stk = new stack19(5);

int choice;
do {
    System.out.println("Menu:");
    System.out.println("1. Push");
    System.out.println("2. Pop");
    System.out.println("3. Peek");
    System.out.println("4. Print");
    System.out.println("5. Exit");
    System.out.print("Enter your choice: ");
    choice = scanner.nextInt();
    scanner.nextLine(); // consume newline character

    switch (choice) {
        case 1:
            System.out.print("Enter book title: ");
            String title = scanner.nextLine();
            System.out.print("Enter book author: ");
            String author = scanner.nextLine();
            System.out.print("Enter published year: ");
            int year = scanner.nextInt();
            System.out.print("Enter pages amount: ");
            int pages = scanner.nextInt();
            System.out.print("Enter price: ");
            int price = scanner.nextInt();
            stk.push(new book19(title, author, year, pages, price));
            break;
        case 2:
            stk.pop();
            break;
        case 3:
            stk.peek();
            break;
        case 4:
            stk.print();
            break;
        case 5:
            System.out.println("Exiting program...");
            break;
        default:
            System.out.println("Invalid choice. Please try again.");
    }
} while (choice != 5);

scanner.close();
```

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



165.3. 3rd Lab Activities

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

165.3.1. 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

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

```
public Postfix(int total) {
    n = total;
    top = -1;
    stack = new char[n];
    push('(');
}
```

4. Create method **push** and **pop** with void as its return type

```
public void push(char c) {
    top++;
    stack[top] = c;
}

public char pop() {
    char item = stack[top];
    top--;
    return item;
}
```

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

```
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;
    }
}
```

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

```
public boolean IsOperator(char c) {
    if (c == '^' || c == '%' || c == '/' || c == '*' || c == '-' || c == '+') {
        return true;
    } else {
        return false;
    }
}
```

7. Create method **degree** as integer to define the degree of the operator

```
public int degree(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;
    }
}
```

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

```
public String convert(String Q){
    String P = "";
    char c;
    for (int i = 0; i < n; i++) {
        c = Q.charAt(i);
        if(IsOperand(c)){
            P = P +c;
        }
        if(c == '('){
            push(c);
        }
        if(c == ')'){
            while(stack[top] != '('){
                P = P + pop();
            }
            pop();
        }
        if(isOperator(c)){
            while (degree(stack[top]) > degree(c)) {
                P = P + pop();
            }
            push(c);
        }
    }
    return P;
}
```

9. 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.

```
Scanner sc = new Scanner(System.in);
String P, Q;
System.out.println("Insert mathematical expression (infix) : ");
Q = sc.nextLine();
Q = Q.trim();
Q = Q + ")";
```

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

10. Create a **total** variable to calculate how many characters in variable Q

```
int total = Q.length();
```

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

```
Postfix post = new Postfix(total);
P = post.convert(Q);
System.out.println("Postfix : " + P);
```

12. 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 {
```

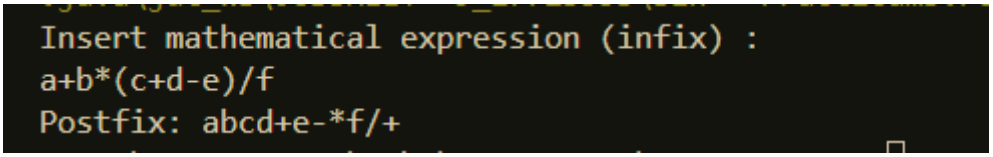
```
            return false;
        }
    }

    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;
//             }
//         }

//     }

// }
```



12.1.1. Result

Check if the result match with following image:

```
run:
Insert mathematical expression (infix) :
a+b*(c+d-e)/f
Postfix : abcde-+f/*+
BUILD SUCCESSFUL (total time: 9 seconds)
```

12.1.2. 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.

2. 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.

   ```
   c = Q.charAt(i);
   ```

3. 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%*.

4. 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.

12.2. Assignment

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

```
13  package Poltek;
14
15  public class CustomStack<T> {
16      private Node<T> top;
17      private int size;
18
19      private static class Node<T> {
20          T data;
21          Node<T> next;
22
23          Node(T data) {
24              this.data = data;
25          }
26      }
27
28      public CustomStack() {
29          top = null;
30          size = 0;
31      }
32
33      public void push(T data) {
34          Node<T> newNode = new Node<>(data);
35          newNode.next = top;
36          top = newNode;
37          size++;
38      }
39
40      public T pop() {
41          if (isEmpty()) {
42              throw new IllegalStateException("Stack is
    empty");
43          }
44          T data = top.data;
45          top = top.next;
46          size--;
```

```
47            return data;
48        }
49
50        public T peek() {
51            if (isEmpty()) {
52                throw new IllegalStateException("Stack is
    empty");
53            }
54            return top.data;
55        }
56
57        public boolean isEmpty() {
58            return size == 0;
59        }
60
61        public int size() {
62            return size;
63        }
64    }
65
```

```
package Poltek;
import java.util.Scanner;
public class poltek {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Insert Sentence: ");
        String sentence = scanner.nextLine();

        String reversedSentence = reverseSentence(sentence);

        System.out.println("Result: " + reversedSentence);
    }

    public static String reverseSentence(String sentence) {
        CustomStack<Character> stack = new CustomStack<>();
```

```
        // Push each character of the sentence onto the stack
        for (int i = 0; i < sentence.length(); i++) {
            stack.push(sentence.charAt(i));
        }

        // Pop each character from the stack to construct the
reversed sentence
        StringBuilder reversed = new StringBuilder();
        while (!stack.isEmpty()) {
            reversed.append(stack.pop());
        }

        return reversed.toString();
    }
}
```

```
ltek'
Insert Sentence: POLITEKNIK NEGERI MALANG
Result: GNALAM IREGEN KINKETILOP
```

```
run:
Insert Sentence: Politeknik Negeri Malang
Result :
gnalaM iregeN kinketiloP
BUILD SUCCESSFUL (total time: 1 second)
```

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

```
66  package Dewi;
67
68  import java.util.Scanner; // Import Scanner class if not
    already imported
```

```
69
70  public class dewi {
71      public static void main(String[] args) {
72
73          CustomStack<Integer> intStack = new
    CustomStack<>();
74          CustomStack<String> stringStack = new
    CustomStack<>(); // Corrected typo
75          CustomStack<Receipt> receiptStack = new
    CustomStack<>();
76          Scanner scanner = new Scanner(System.in);
77
78          // Simulate adding receipts to the stack
79          for (int i = 1; i <= 8; i++) {
80              System.out.println("Enter details for
    Receipt " + i + ":");
81              System.out.print("Transaction ID: ");
82              String transactionId = scanner.nextLine();
83              System.out.print("Date: ");
84              String date = scanner.nextLine();
85              System.out.print("Quantity of Items: ");
86              int quantity =
    Integer.parseInt(scanner.nextLine());
87              System.out.print("Total Price: ");
88              double totalPrice =
    Double.parseDouble(scanner.nextLine());
89
90              Receipt receipt = new Receipt(transactionId,
    date, quantity, totalPrice);
91              receiptStack.push(receipt);
92          }
93
94          // Retrieve and display the receipts
95          System.out.println("\nRetrieving Receipts:");
96          while (!receiptStack.isEmpty()) {
97              Receipt receipt = receiptStack.pop();
98              System.out.println(receipt);
```

```
99          }
100     }
101 }
102
103 package Dewi;
104
105 public class CustomStack<T> {
106     private Node<T> top;
107     private int size;
108
109     private static class Node<T> {
110         T data;
111         Node<T> next;
112
113         Node(T data) {
114             this.data = data;
115         }
116     }
117
118     public CustomStack() {
119         top = null;
120         size = 0;
121     }
122
123     public void push(T data) {
124         Node<T> newNode = new Node<>(data);
125         newNode.next = top;
126         top = newNode;
127         size++;
128     }
129
130     public T pop() {
131         if (isEmpty()) {
132             throw new IllegalStateException("Stack is
    empty");
133         }
134         T data = top.data;
```

```
135             top = top.next;
136             size--;
137             return data;
138         }
139 
140         public T peek() {
141             if (isEmpty()) {
142                 throw new IllegalStateException("Stack is
    empty");
143             }
144             return top.data;
145         }
146 
147         public boolean isEmpty() {
148             return size == 0;
149         }
150 
151         public int size() {
152             return size;
153         }
154 }
155 
156 package Dewi;
157 
158 public class Receipt {
159         private String transactionId;
160         private String date;
161         private int quantity;
162         private double totalPrice;
163 
164         public Receipt(String transactionId, String date,
    int quantity, double totalPrice) {
165             this.transactionId = transactionId;
166             this.date = date;
167             this.quantity = quantity;
168             this.totalPrice = totalPrice;
169         }
```

```
170
171      // Getters and setters
172  }
173
```

TERMINAL

```
Date: 3
Quantity of Items: 23
Total Price: 2
Enter details for Receipt 6:
Transaction ID: 3
Date: 23
Quantity of Items: 2
Total Price: 3
Enter details for Receipt 7:
Transaction ID: 23
Date: 23
Quantity of Items: 32
Total Price: 332
Enter details for Receipt 8:
Transaction ID: 3
Date: 23
Quantity of Items: 23
Total Price: 2

Retrieving Receipts:
Dewi.Receipt@8807e25
Dewi.Receipt@2a3046da
Dewi.Receipt@2a098129
Dewi.Receipt@198e2867
Dewi.Receipt@12f40c25
Dewi.Receipt@3ada9e37
Dewi.Receipt@5cbc508c
Dewi.Receipt@3419866c
```

now
44 secs

```
Quantity of Items: 22
Total Price: 13244
Enter details for Receipt 6:
Transaction ID: 4
Date:
Quantity of Items: 55
Total Price: 1242
Enter details for Receipt 7:
Transaction ID: 6
Date: 634
Quantity of Items: 6
Total Price: 3
Enter details for Receipt 8:
Transaction ID: 6
Date: 67
Quantity of Items: 73
Total Price: 4

Retrieving Receipts:
Transaction ID: 6, Date: 67, Quantity: 73, Total Price: 4.0
Transaction ID: 6, Date: 634, Quantity: 6, Total Price: 3.0
Transaction ID: 4, Date: , Quantity: 55, Total Price: 1242.0
Transaction ID: 31, Date: 3, Quantity: 22, Total Price: 13244.0
Transaction ID: 3, Date: 1, Quantity: 23, Total Price: 2.0
Transaction ID: 3, Date: 23, Quantity: 1, Total Price: 3.0
Transaction ID: 12, Date: 21, Quantity: 2, Total Price: 21.0
Transaction ID: 12, Date: 2024, Quantity: 14, Total Price: 23.0
```