JOBSHEET VII

STACK

7.1 Practicum Objectives

After completing this practical material, students will be able to:

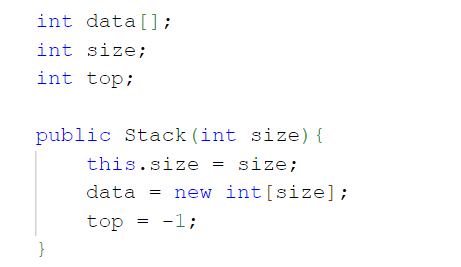
1. Get to know the Stack data structure
2. Create and declare a Stack data structure
3. Implement the Stack algorithm using arrays
4. Practical 1

**Trial time: 30 minutes**

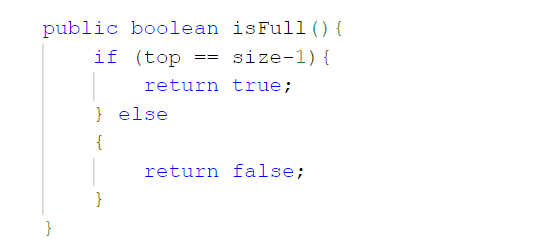
In this experiment, we will create a program that implements the Stack data structure and basic operations on the Stack data structure using arrays.

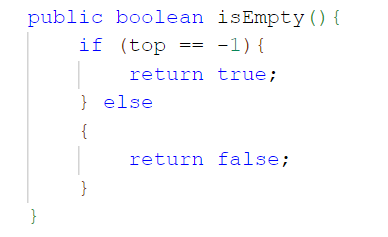
* + 1. Experimental Steps

1. Create a folder with the name Practicum07. Create a Stack.java file.
2. Write code to create attributes and constructors for the Stack class as follows:

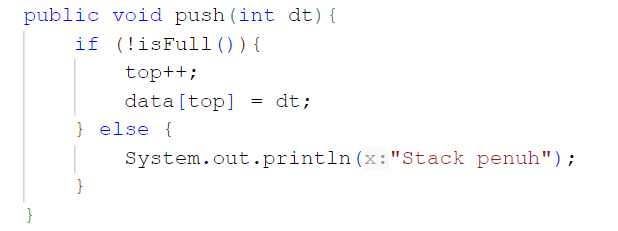


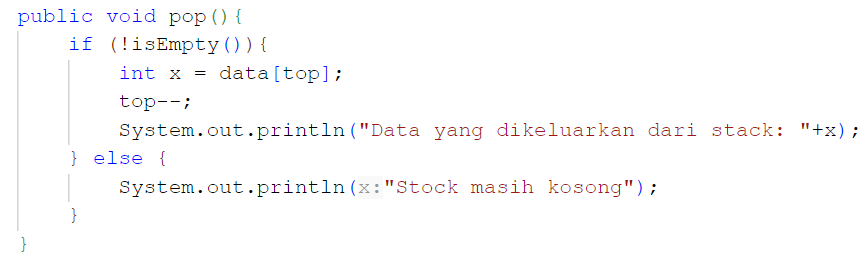
1. Then add the isFull() and isEmpty() methods to the Stack class as follows:



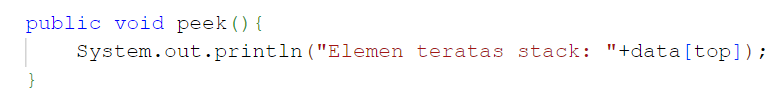


1. Add the push(int data) and pop() methods as follows:

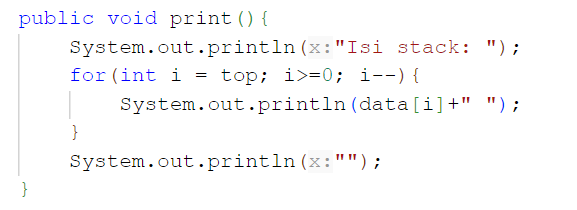


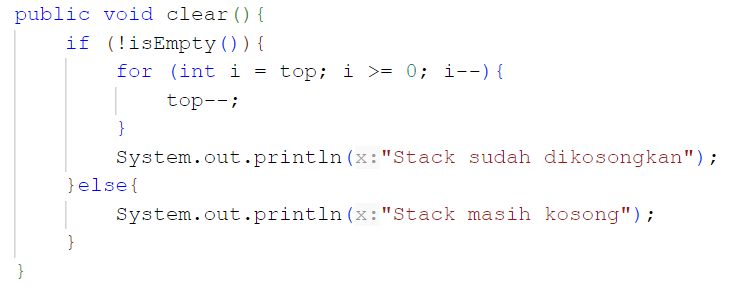


1. Add the peek() method as follows:

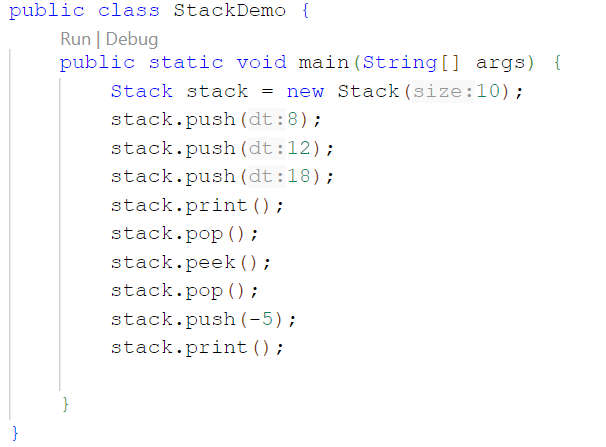


1. Add print() and clear() methods as follows:

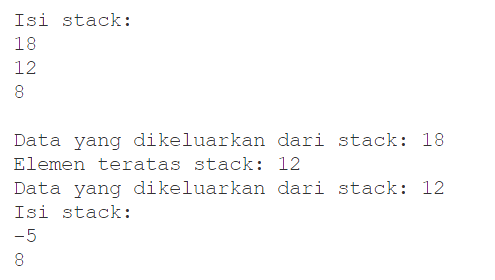


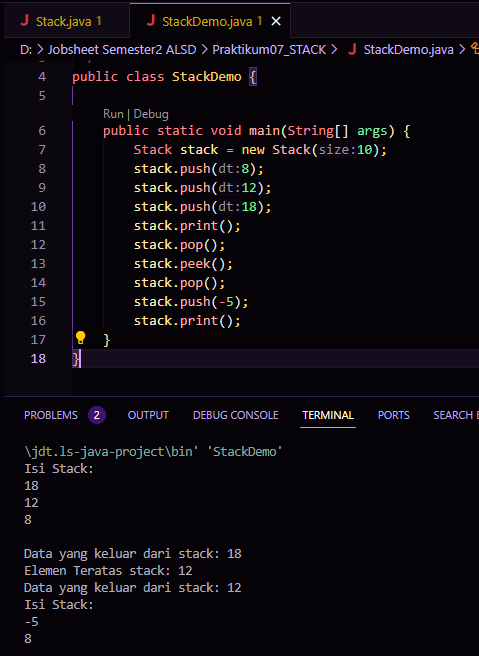


1. Create a **StackDemo.java file** to implement the StackDemo class which contains the main function to create Stack objects and operate methods in the Stack class.



1. Compile and run the StackDemo class.
   * 1. Verification of Experiment Results





* + 1. Question

1. In the pop() method, why is it necessary to call the isEmpty() method? What happens if there is no isEmpty() call?

**In the pop() method, the isEmpty() call is used to check whether the stack is empty before trying to output data. If there is no call to isEmpty(), there is a risk of index errors or unwanted operations if it tries to access data from an otherwise empty stack.**

1. Explain the difference between the peek() method and the pop() method in the Stack class.

* peek(): View the top element of the stack without removing it from the stack.
* pop(): Removes and returns the top element from the stack.

1. Practicum 2

**Experiment time: 45 minutes**

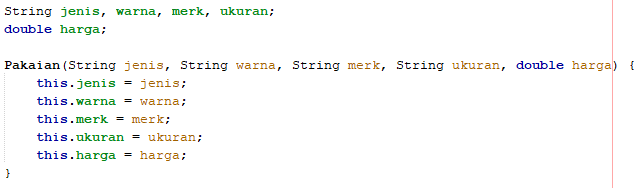
In this experiment, we will create a program that illustrates a pile of clothes stored in a stack. Because a piece of clothing has some information, the implementation of Stack is done by using an array of objects to represent each element.

1. Experimental Steps
2. Pay attention to the following Clothing Class Diagram:

|  |
| --- |
| Clothes |
| type: String  Color: String  Brand: String  Size: String  price: double |
| Clothing(type: String, color: String, brand: String , size: String , price: double) |

Based on the class diagram, a Java clothing class program will be created.

1. Create a new class with the name **Clothing** .
2. Add Clothing attributes as in the Clothing Class Diagram, then add the constructor as shown in the following image.

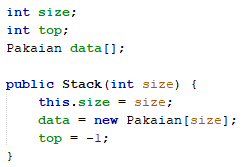


1. After creating the Clothing class, next you need to create a **Stack class** which contains attributes and methods according to the following Class Stack diagram:

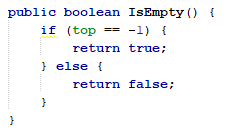
|  |
| --- |
| Stacks |
| size: int  top: int  data[]: Clothing |
| Stack(size: int)  IsEmpty(): boolean  IsFull(): boolean  push(): void  pop(): void  peek(): void  print(): void  clear(): void |

***Description*** *: The data type in the variable* ***data*** *corresponds to the data that will be stored in the Stack. In this practicum, the data to be stored is an array of Clothing objects, so the data type used is* ***Clothing***

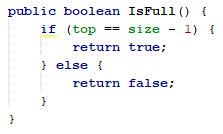
1. Create a new class with the name Stack. Then add attributes and constructors as shown in the following image.



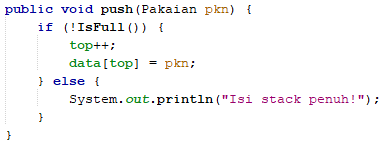
1. Create an **IsEmpty method** of type boolean which is used to check whether the stack is empty.



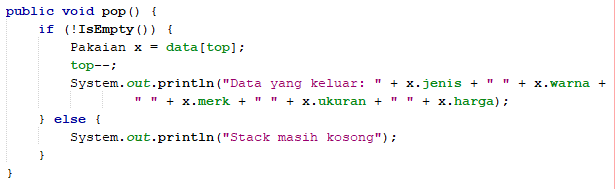
1. Create an **IsFull method** of type boolean which is used to check whether the stack is completely filled.



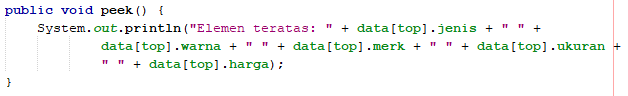
1. Create a **push method** of type void to add the contents of a stack element with a **pkn parameter** in the form of a **Clothing object**



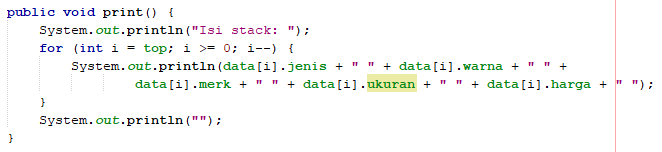
1. Create a **Pop method** of type void to display the contents of the stack element. Because one stack element consists of several pieces of information (type, color, brand, size, and price), when printing the data it is also necessary to display all of this information.



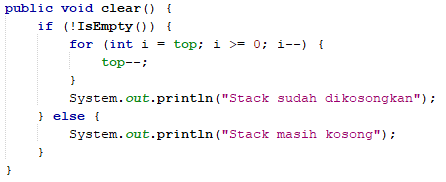
1. Create a **peek method** of type void to check the stack element at the top position.



1. Create a **print method** of type void to display all elements on the stack.



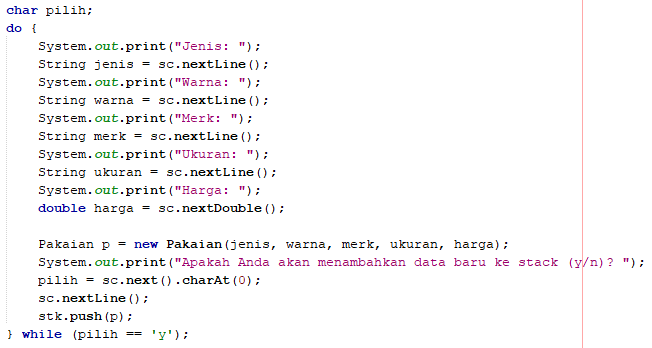
1. Create a **clear method** of type void to delete the entire contents of the stack.



1. Next, create a new class with the name **StackMain** . Create a main function, then instantiate an object from **the Stack class** with the name **stk** and the parameter value is 5.

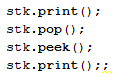


1. Declare the Scanner with the name **sc**
2. Add the following code to receive Clothing data input, then all the information is put into the stack

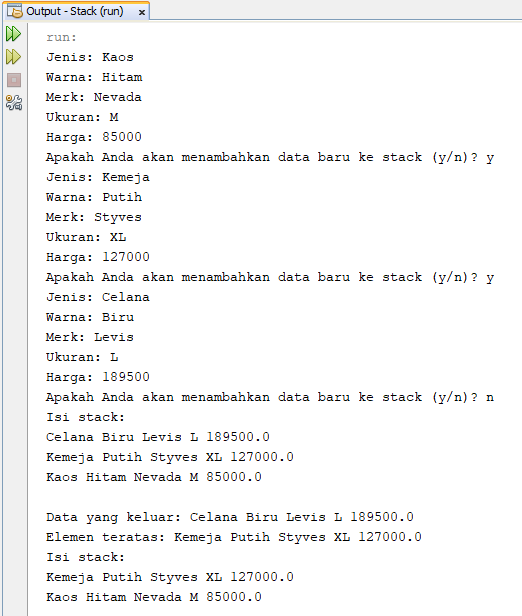


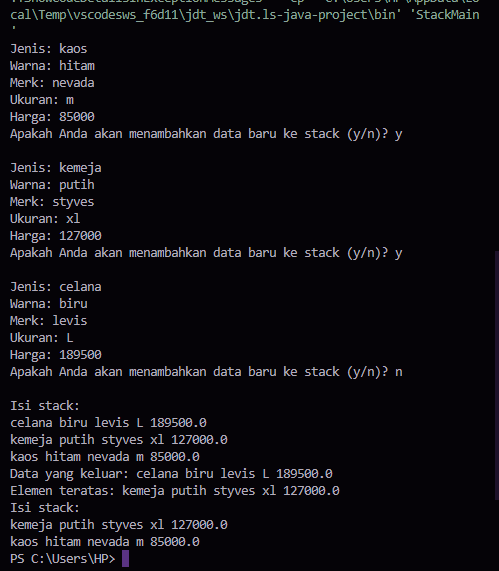
***Note*** *: sc.nextLine() syntax before st.push(p) syntax is used to ignore new line characters*

1. Call the print method, pop method, and peek method in the following order.



1. Compile and run the **StackMain class** , then observe the results.
2. Verification of Experiment Results





1. Question
2. How much clothing data can be accommodated in the stack? Show a snippet of program code to support your answer!

**Stack11 stk = new Stack11(5);**

1. **StackMain** class , when calling the push function, the parameter sent is **p** . What data is stored in the variable **p** ?



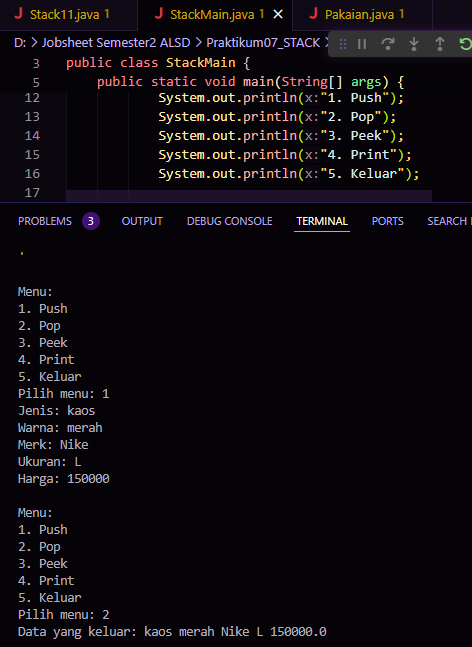
**In the StackMain class, when calling the push function, the parameter p passed is the Clothes object. The Clothing object contains information about a garment, namely type, color, brand, size and price.**

1. **do-while** function in the **StackMain class** ?

**The StackMain class uses do-while to enter clothing data and repeats the process of adding that data to the stack.**

**This loop continues as long as the user types ``y'' to add new data to the stack. When the user enters input 'n', the loop stops and the program continues to the next section after the 'do-while' block.**

1. Modify the program code in the **StackMain class** so that users can select operations on the stack (push, pop, peek, or print) through the program menu options by utilizing the IF-ELSE or SWITCH-CASE conditions!



1. Practicum 3

**Experiment time: 30 minutes**

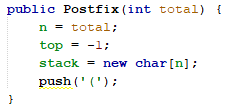
In this experiment, we will create a program to convert infix notation to postfix notation.

1. Experimental Steps
2. Look at the following Class Diagram:

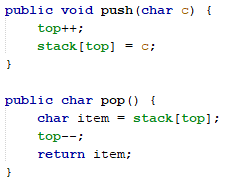
|  |
| --- |
| Postfix |
| n: int  top: int  stack: char[] |
| Postfix(total: int)  push(c: char): void  pop(): void  IsOperand(c: char): boolean  IsOperator(c: char): boolean  degree(c: char): int  convert(Q: String): string |

Based on the class diagram, a Postfix class program in Java will be created.

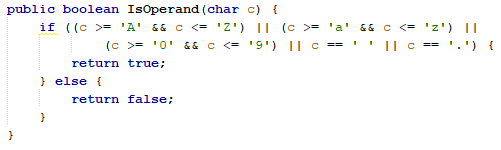
1. Create a new class with the name **Postfix** . Add the **n** , **top** , and **stack attributes** according to the Postfix class diagram.
2. Also add a constructor with parameters as shown in the following image.



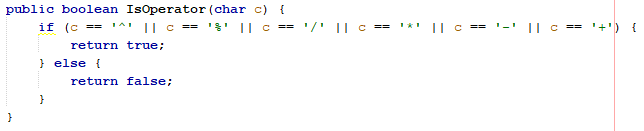
1. Make the **push** and **pop methods** of type void.



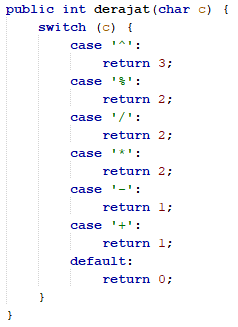
1. Create an **IsOperand method** with type boolean which is used to check whether the data element is an operand.



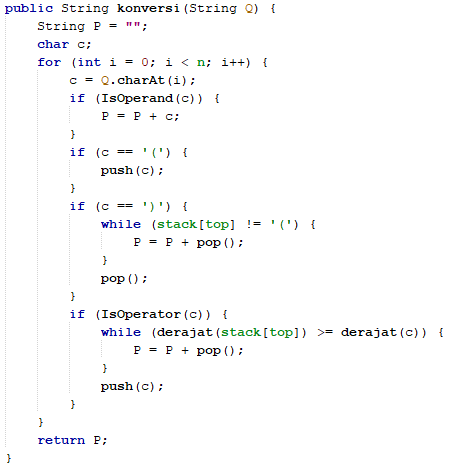
1. Create an **IsOperator method** with type boolean which is used to check whether the data element is an operator.



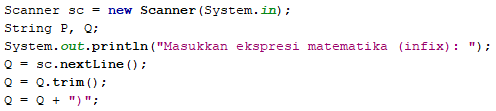
1. Create a **degree method** that has an integer return value to determine the degree operator.



1. Create a conversion method to convert infix notation to postfix notation by checking the data elements in **String Q one by one** as input parameters.



1. Next, create a new class with the name **PostfixMain** . Create a main class, then create variables P and Q. Variable P is used to store the final result of postfix notation after conversion, while variable Q is used to store input from the user in the form of mathematical expressions with infix notation. Declare the Scanner variable with the name sc, then call the *built-in function* **trim** is used to remove spaces in front or behind the text from the equation text entered by the user.



The addition of the string **")"** is used to ensure that all symbols/characters that are still on the stack after all the equations have been read, will be removed and moved to postfix.

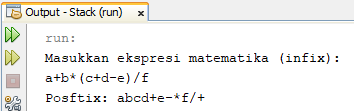
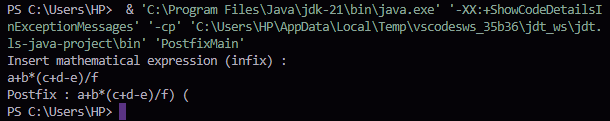
1. Create a total variable to count the number of characters in variable Q.



1. Instantiate an object with the name **post** and the parameter value is total. Then call the **conversion method** to convert infix Q notation to postfix P notation.



1. Compile and run the **PostfixMain class** and observe the results.
2. Verification of Experiment Results



1. Question
2. **Postfix** class , explain the workflow **of the degree method** !

**Steps: Initialization: This method starts by initializing an empty string P to store the postfix expression.**

**Additionally, a character c is declared that repeats each character in the infix expression.**

**Process each character: This method loops through each character of the Q infix expression.**

**For each character c: If c is not an operand (operator or parenthesis), then it is immediately added to the postfix expression P.**

**if c is the operand.**

**This method checks whether it is an opening bracket '('.**

**If it exists, the method adds it to the stack to mark the start of the subexpression.**

**If c is a closing parenthesis ")", then the method takes the operator off the stack and adds it to the postfix expression P until a matching opening parenthesis "(" is found.**

**Then the opening bracket is removed.**

**Preference Operator If the child c is less than or equal to, this method compares its priority with the preference operator at the top of the stack, removes the operator from the stack, and adds it to the postfix expression P.**

**This process continues until the stack is empty or the operator at the top of the stack has a priority lower than c.**

**Finally, the c operator is added to the stack.**

**Solution: After processing all the characters in the infix expression Q, this method checks if there are any operators left on the stack.**

**If it exists, the method prints it and adds it to P's postfix expression.**

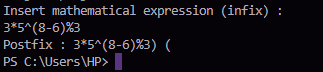
**Returns: This method returns the postfix expression P which is the result of converting the infix expression Q to postfix notation.**

1. What is the function of the following program code?



**Code c = Q.charAt(i); functions to take the character at the i-th index of the string Q and store it in the variable c.**

1. Run the program again, enter the expression **5\*4^(1+2)%3** . Show the results!



1. In question number 3, why are brackets not displayed in the conversion results? Explain!

**d in the conversion of mathematical expressions from infix to postfix, parentheses are not included in the postfix expression because the order of operations is determined by the position of the operands and operators, without the need for parentheses to determine precedence.**

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

