**JOB SHEET VIII Stack**



**From:**

AL AZHAR RIZQI RIFA’I FIRDAUS

**Class:**

1 I

**Absence:**

01

**Student Number Identity:**

2241720263

**Department:**

Information Technology

**Study Program:**

Informatics Engineering

* 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.

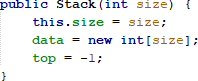
* + - 1. Create a new project named **Jobsheet7.** Create a new package with name **Practicum1.**

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

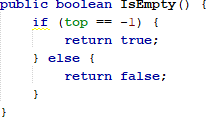
* + - 1. Create new attributes size, top, and data as follows:

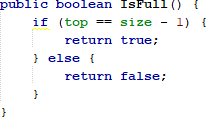


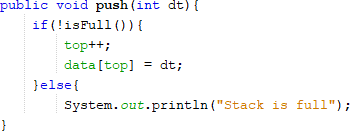
* + - 1. Add a constructor with parameter as written below:

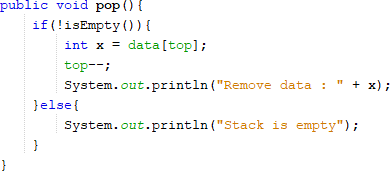


* + - 1. Create a method **isEmpty** with Boolean as its return type to check whether the stack is empty or not.

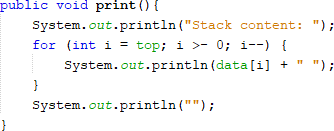


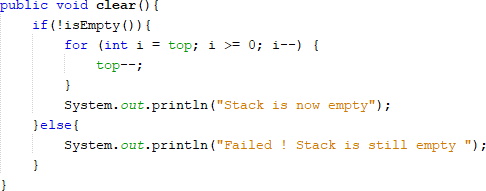
* + - 1. Create a method **isFull** with Boolean as its return type to check whether the stack is filled completely or not.
      2. Create method **push** with void as its return type to add new stack element with parameter **dt**. This dt variable is in form of integer

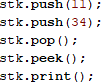


* + - 1. Create method **pop** with void as its return type to remove an element from the stack
      2. Create method **peek** with void as its return type to check the top element of the stack

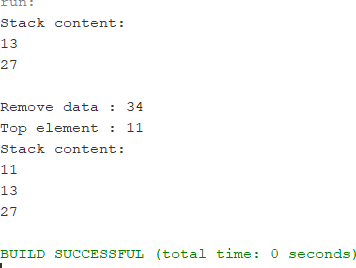


* + - 1. Create method **print** with void as its return type to display the content of the stack
      2. Create method **clear** with void as its data type to remove all elements and make the stack empty



* + - 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**
      2. Fill the stack object by calling method **push**, the data is being inserted accordingly
      3. Display the data that we’ve inserted in previous step by calling method **print**
      4. 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**
      5. Compile and run the program, check the result
    1. Result

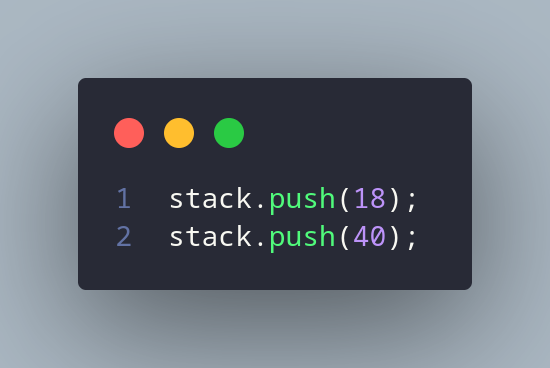
Check if the result match with following image:



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

- The number 5 represents the capacity of the stk object that is being created using the Stack class.

* + - 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?

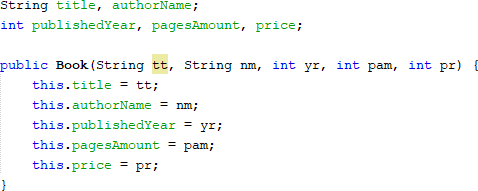
- Because the capacity of the stack is set to only 5. When we push 18 using .push() method the IsFull() method willl be return true it means the stack already full.

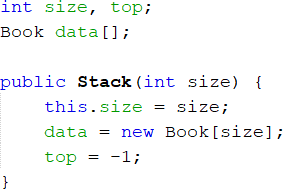
* 1. 2nd Lab Activities

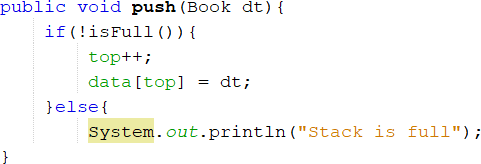
In this practicum, we will create a program to illustrate a bunch of books that are stored in Stack. Since the book has some information on it, the stack implementation is done using array of object to represent each element.

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

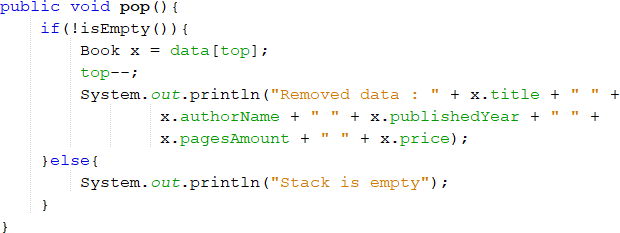
|  |
| --- |
| Book |
| title: String authorName: String publishedYear: int pagesAmount: int  price: int |
| Book(title: String, author: String, publishedYear: int,  pagesAmount: int, price: int) |

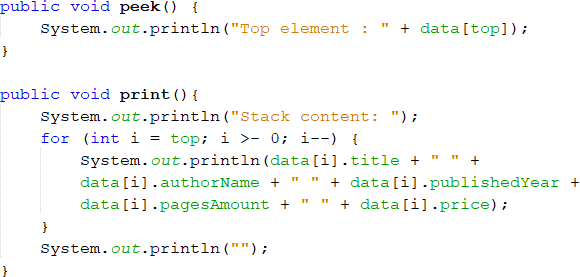
* + - 1. Create a new package named **Practicum2,** then create a new class named **Book.**
      2. Add attributes in that class, and add the constructor as well.
      3. Copy the program code for **Stack** class in **Practicum1** to be used again in here. Since the data stored in Stack in **Practicum1** is integer array, and in **Practicum2** we use objects, we will need to modify some parts in that class.
      4. 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:
      2. Compile and run **StackMain**, and observe the result
    1. Result

Check if the result match with following image:

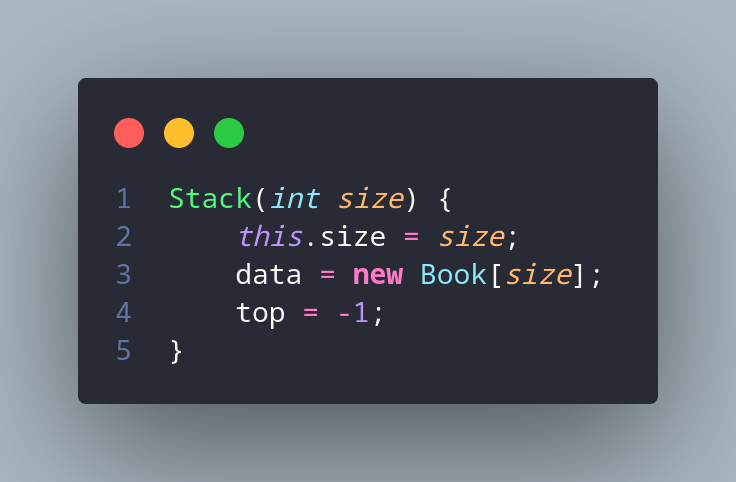


* + 1. Questions
       1. In class StackMain, when calling **push** method, the argument is **bk.** What information is included in the **bk** variable?

- bk variable is an instance of the Book class, created with the following parameters passed to its constructor title, name, year, pages, and price.

* + - 1. Which of the program that its usage is to define the capacity of the stack ?

- in the Stack.class



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

- To do itteration that contains input book data then every single itteration data will be stored at bk variable, and then will be push with .push() method.

* + - 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!

Code :



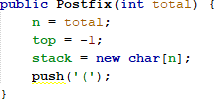
* 1. 3rd Lab Activities

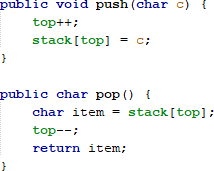
In this practicum, we will create program to convert infix notation into postfix notation

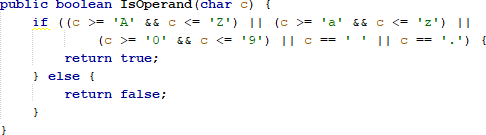
* + 1. Steps
       1. We will use class diagram to create **Postfix** class in Java program

|  |
| --- |
| 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 |

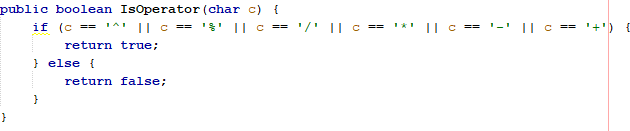
* + - 1. Create a package named **Practicum3.** Then, we create a new class named **Postfix.** Add attributes **n, top, and stack** based on class diagram above.
      2. Add a constructor with parameter as follows:



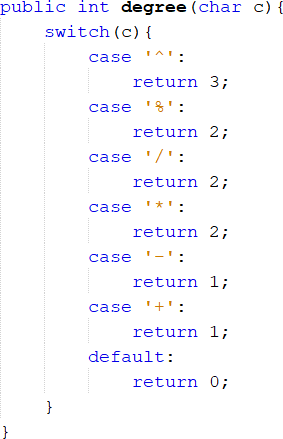
* + - 1. Create method **push** and **pop** with void as its return type
      2. Create method **isOperand** as Boolean that will be used to check if the element is operand or not



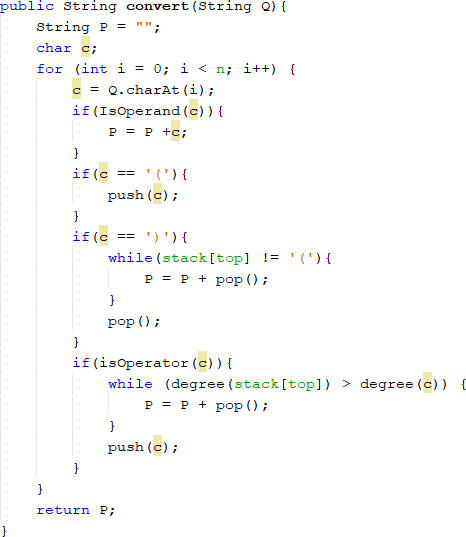
* + - 1. Create method **isOperator** as booelan that will be used to check if the element is operator or not



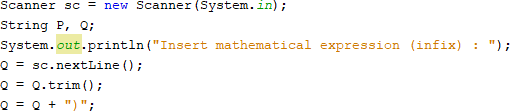
* + - 1. Create method **degree** as integer to define the degree of the operator



* + - 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.

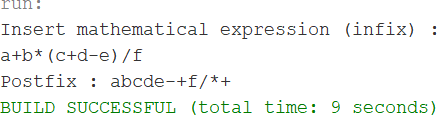
* + - 1. Create a **total** variable to calculate how many characters in variable Q



* + - 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
    1. Result

Check if the result match with following image:

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

Constructor Postfix(int total):

Initializes the n instance variable to the total parameter value.

Initializes the top instance variable to -1, indicating the stack is initially empty.

Creates a new char array stack with size n.

Calls the push() method to push a left parenthesis character '(' onto the stack.

Method push(char c):

Increments the top instance variable by 1 to prepare for the new character to be added.

Adds the character c to the stack array at the top index.

Method pop():

Gets the character at the top index of the stack array and assigns it to the item variable.

Decrements the top instance variable by 1 to remove the character from the stack.

Returns the item variable containing the removed character.

Method isOperand(char c):

Returns true if the character c is an operand (i.e., a letter, digit, space or period) and false otherwise.

Method isOperator(char c):

Returns true if the character c is an operator (i.e., one of '^', '%', '/', '\*', '-', or '+') and false otherwise.

Method degree(char c):

Returns the degree of precedence of the operator c. The operator with the highest degree of precedence has the highest integer value returned.

Method convert(String Q):

Initializes an empty string p to store the postfix expression.

Iterates through each character c in the infix expression Q.

If c is an operand, it is added to the p string.

If c is a left parenthesis, it is pushed onto the stack.

If c is a right parenthesis, all characters in the stack are popped off and added to the p string until a left parenthesis is found and removed.

If c is an operator, all operators with a higher precedence than c are popped off the stack and added to the p string before c is pushed onto the stack.

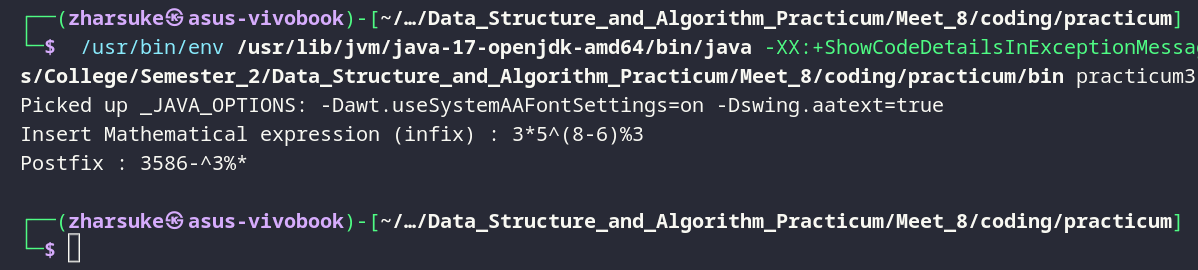
After all characters in Q have been processed, any remaining operators in the stack are popped off and added to the p string.

The postfix expression p is returned.

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

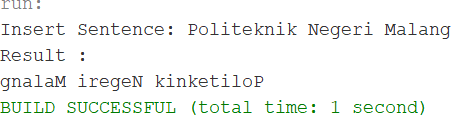
retrieves the character at index i from the string Q and assigns it to the variable c.

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

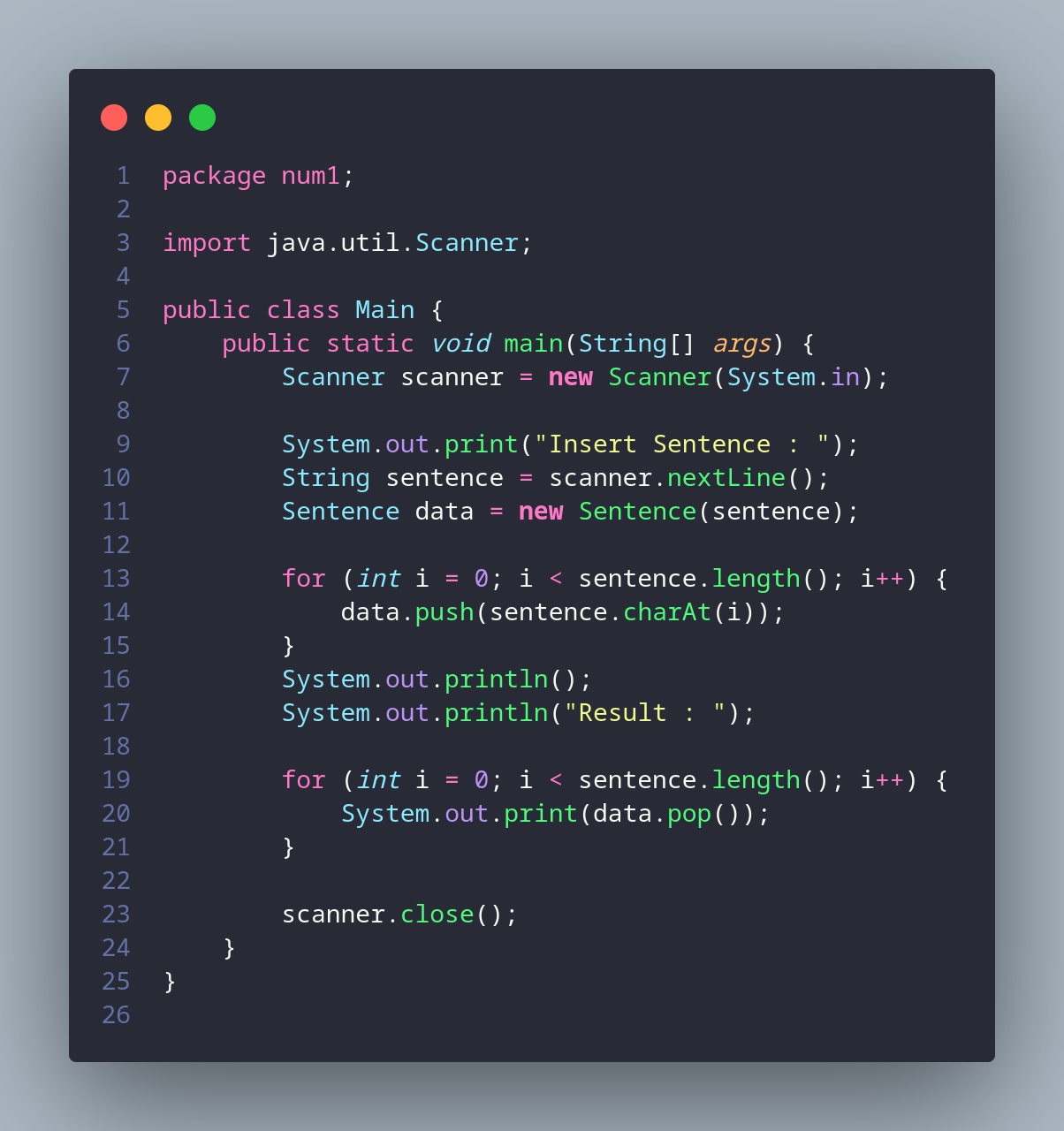


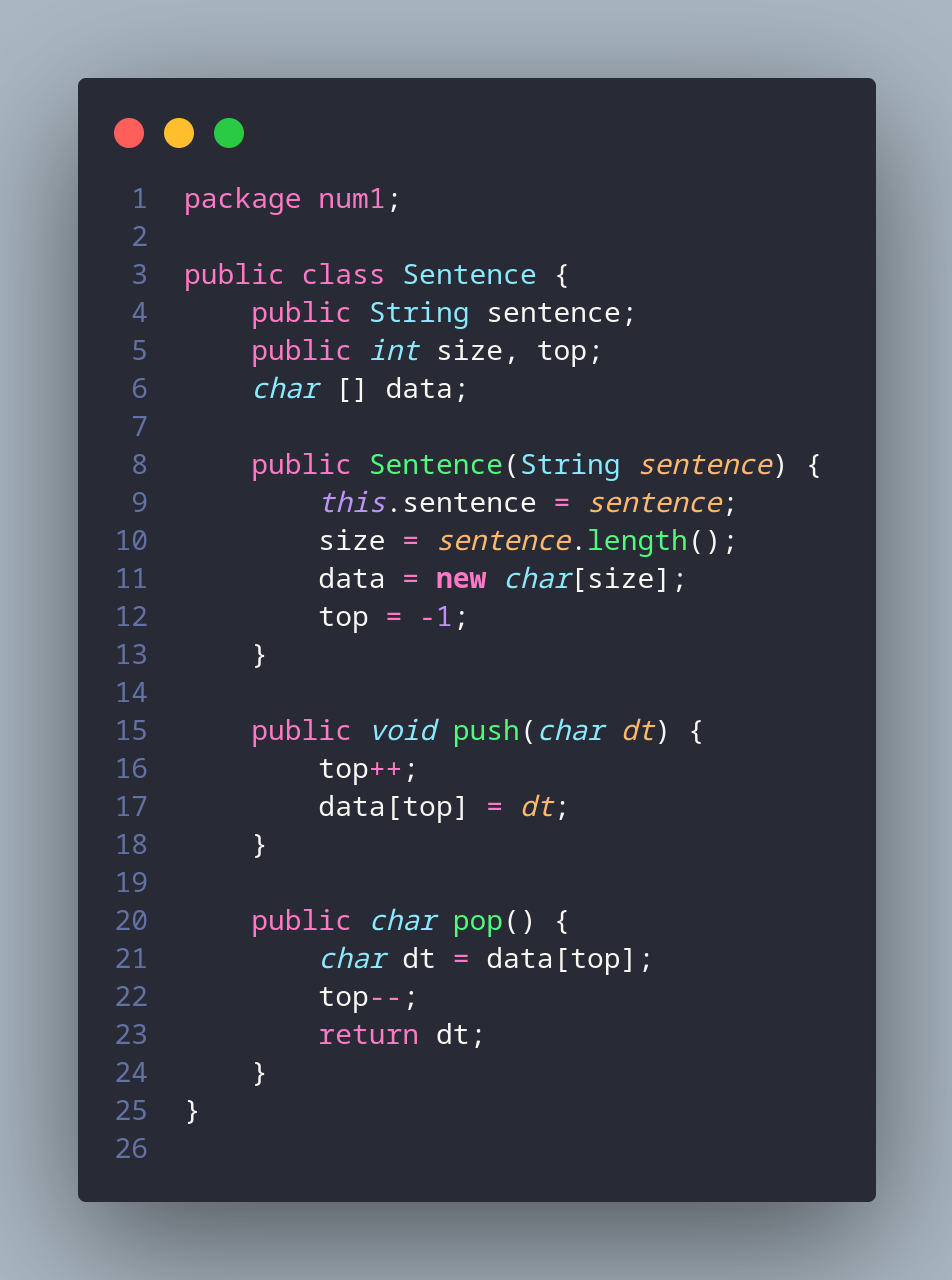
* + - 1. In 2nd number, why the braces are not displayed in conversion result? Please explain

- The reason why the braces are not visible in the converted output is because the Postfix class's convert method solely utilizes the parentheses to establish the order of operations and does not include them in the resultant postfix expression. Whenever the method encounters an opening parenthesis, it places it onto the stack. On the other hand, when it encounters a closing parenthesis, it pops the operators from the stack and adds them to the postfix expression until it locates the opening parenthesis, which is then removed from the stack. Hence, the parentheses serve as a mechanism to ensure the accurate order of operations, and they are not a constituent of the final postfix expression.

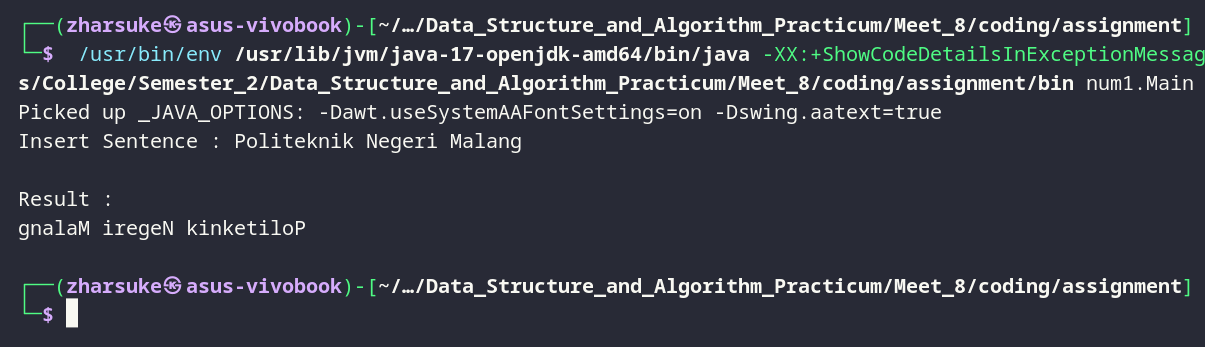
* 1. Assignment
     1. Create a program with Stack implementation to insert a sentence and display the reversed version of the sentence as a result!

Code :





Result :



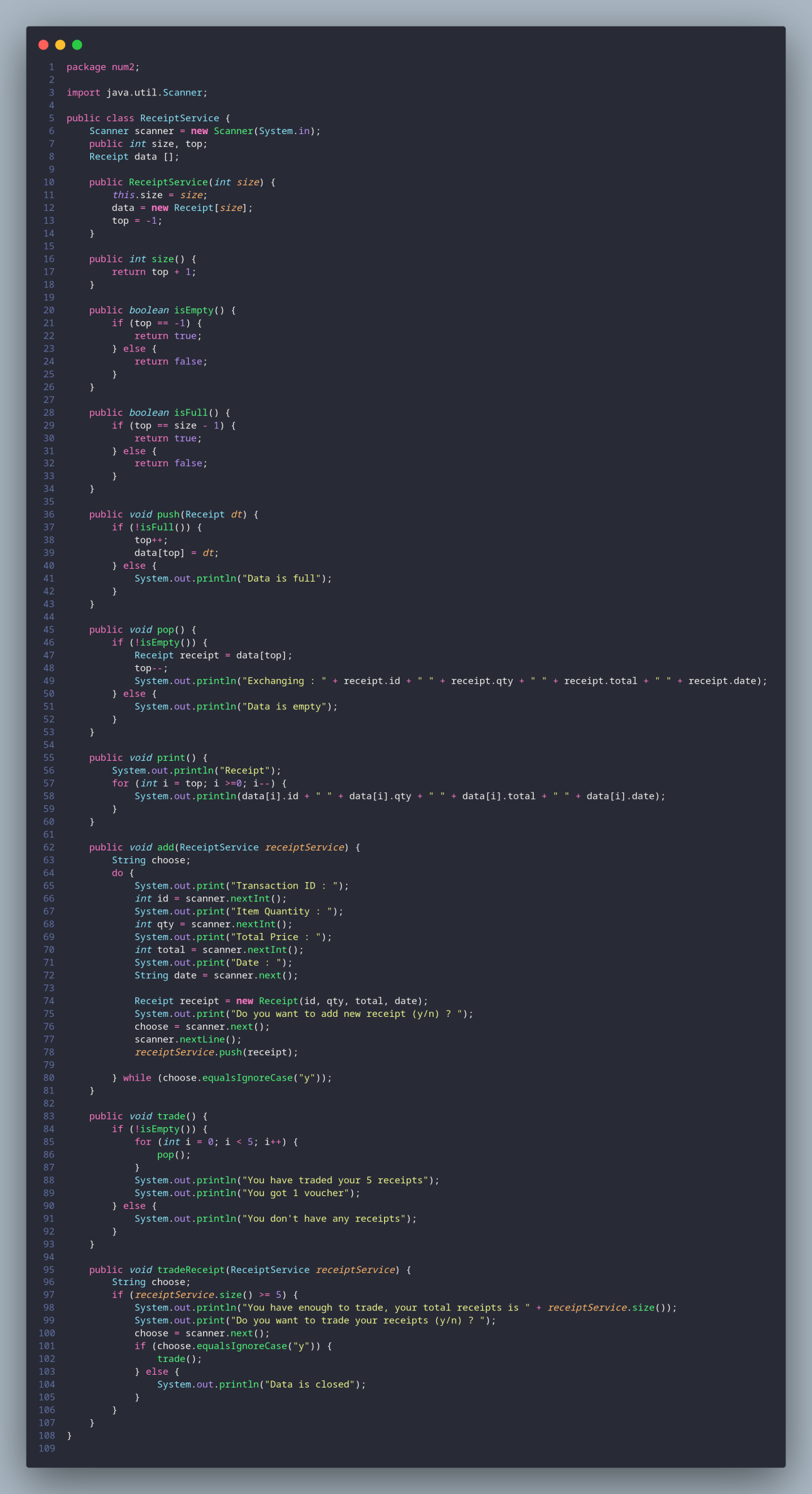
* + 1. Every Sunday, Dewi shops to a supermarket that is in her residential area. Everytime she finishes, she keeps the receipt of what she has bought in a wardrobe. After 2 months, She had 8 receipts. She plans to trade her 5 receipts in exchange for a voucher.

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

Code :







Result :

