

Nama : Muhammad Nazril Nur Rahman

NIM : 2341760174

JOBSHEET IX

STACK

1.1. Learning Objective

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

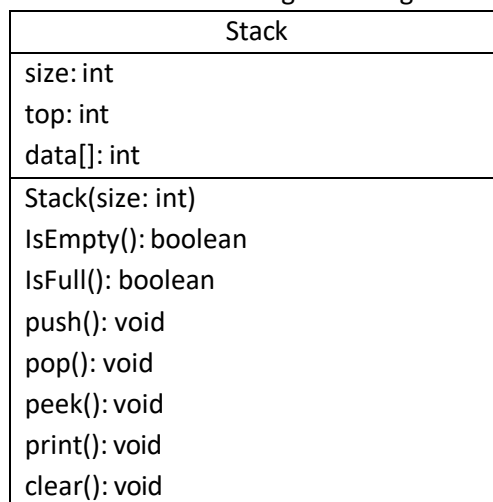
- Define the Stack Data Structure
- Create and implement Stack Data Structure
- Implement Stack data Structure with arrays

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:



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

1.2.2. 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)

```

```

package pratikum1;

public class Stack {
    int data[];
    int size;
    int top;

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

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

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

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

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

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

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

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

```

```

package pratikum1;

public class StackMain {
    Run | Debug
    public static void main(String[] args) {
        Stack stk = new Stack(size:5);
        stk.push(dt:15);
        stk.push(dt:27);
        stk.push(dt:13);
        stk.print();
        stk.push(dt:11);
        stk.push(dt:34);
        stk.pop();
        stk.peek();
        stk.print();
    }
}

```

1.StackMain

Stack content::

13
27

Remove data : 34

Top element : 11

Stack content::

11
13
27

1.2.3. Questions

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

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

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.

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

```
//ADD two more data
stk.push(dt:18);
stk.push(dt:40);
```

```
Stack content::
13
27
Stack is full
Stack is full
Remove data : 34
Top element : 11
Stack content::
11
13
27
```

3. In previous number, the data inserted in to the stack is only 18 and 40 is not inserted.

Why is that?

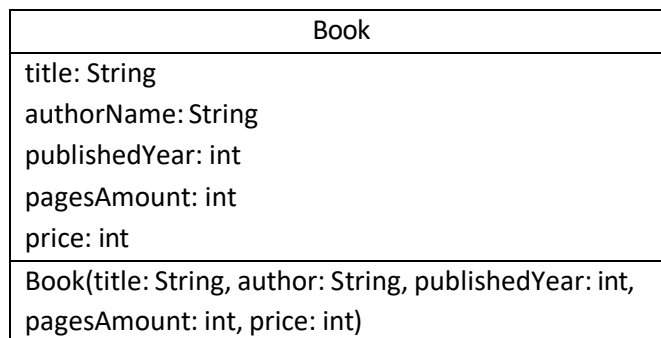
The reason why as it were the information "18" is embedded into the stack and "40" isn't embedded is since of the information sort bungle.

1.3. 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.

1.3.1. Steps

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



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

1.3.2. 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)
,
```


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

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

```
public class StackMain {
    Run | Debug
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Stack stk = new Stack(size:5);

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

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

        scanner.close();
    }
}
```

1.4. 3rd Lab Activities

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

1.4.1. 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 boolean 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);
```

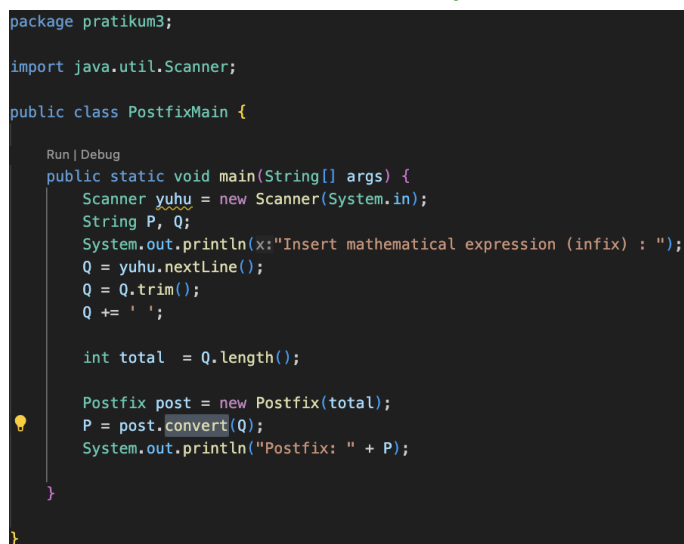
12. Compile and run **StackMain**, and observe the result

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)
```



```
package pratikum3;

import java.util.Scanner;

public class PostfixMain {

    Run | Debug
    public static void main(String[] args) {
        Scanner yuhu = new Scanner(System.in);
        String P, Q;
        System.out.println(x:"Insert mathematical expression (infix) : ");
        Q = yuhu.nextLine();
        Q = Q.trim();
        Q += ' )';

        int total = Q.length();

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

```

public class Postfix {
    int n;
    int top;
    char[] stack;

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

    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 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;
        }
    }
}

```

Insert mathematical expression (infix) :
a+b*(c+d-e)/f
Postfix: abcd+e-*f/+

12.1.2. Questions

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

Constructor (Postfix(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?

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

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.

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!

run:

Insert Sentence: Politeknik Negeri Malang

Result :

gnalaM iregeN kinketiloP

BUILD SUCCESSFUL (total time: 1 second)

```
package Assignment;

import java.util.Scanner;
import java.util.Stack;

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

        String reversedSentence = reverseSentence(sentence);

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

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

        for (int i = 0; i < sentence.length(); i++) {
            stack.push(sentence.charAt(i));
        }

        StringBuilder reversed = new StringBuilder();
        while (!stack.isEmpty()) {
            reversed.append(stack.pop());
        }

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

```
Insert Sentence: Politeknik Negeri Malang
Result: gnalaM iregeN kinketiloP
nazril@Muhammads-MacBook-Air JOBSHEET7 %
```


2. 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 Assignment;

import java.util.Scanner;
import java.util.Stack;

public class DewiReceipt {
    Run | Debug
    public static void main(String[] args) {
        Stack<Receipt> receiptStack = new Stack<>();
        Scanner scanner = new Scanner(System.in);

        for (int i = 1; i <= 8; i++) {
            System.out.println("Enter details for Receipt " + i + ":");
            System.out.print(s:"Transaction ID: ");
            String transactionId = scanner.nextLine();
            System.out.print(s:"Date: ");
            String date = scanner.nextLine();
            System.out.print(s:"Quantity of Items: ");
            int quantity = Integer.parseInt(scanner.nextLine());
            System.out.print(s:"Total Price: ");
            double totalPrice = Double.parseDouble(scanner.nextLine());

            Receipt receipt = new Receipt(transactionId, date, quantity, totalPrice);
            receiptStack.push(receipt);
        }

        System.out.println(x:"\nRetrieving Receipts:");
        while (!receiptStack.isEmpty()) {
            Receipt receipt = receiptStack.pop();
            System.out.println(receipt);
        }
    }
}
```

```
package Assignment;

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;
    }
}
```