Trace the following code, showing the contents of the stack after each invocation:

Stack stack = new Stack();

Stack.push(new Character('A'));

Stack.push(new Character('B'));

Stack.push(new Character('C'));

Stack.pop();

Stack.pop();

Stack.push(new Character('D'));

Stack.push(new Character('E'));

Stack.push(new Character('F'));

Stack.pop();

Stack.push(new Character('G'));

Stack.pop();

Stack.pop();

Stack.pop()

Let's trace the given code and track the contents of the stack after each invocation:

1. Create a new stack: `stack = new Stack()`

Stack: []

1. Push 'A' onto the stack: `stack.push(new Character('A'))`

Stack: ['A']

1. Push 'B' onto the stack: `stack.push(new Character('B'))`

Stack: ['A', 'B']

1. Push 'C' onto the stack: `stack.push(new Character('C'))`

Stack: ['A', 'B', 'C']

1. Pop an element from the stack: `stack.pop()`

Stack: ['A', 'B']

1. Pop an element from the stack: `stack.pop()`

Stack: ['A']

1. Push 'D' onto the stack: `stack.push(new Character('D'))`

Stack: ['A', 'D']

1. Push 'E' onto the stack: `stack.push(new Character('E'))`

Stack: ['A', 'D', 'E']

1. Push 'F' onto the stack: `stack.push(new Character('F'))`

Stack: ['A', 'D', 'E', 'F']

1. Pop an element from the stack: `stack.pop()`

Stack: ['A', 'D', 'E']

1. Push 'G' onto the stack: `stack.push(new Character('G'))`

Stack: ['A', 'D', 'E', 'G']

1. Pop an element from the stack: `stack.pop()`

Stack: ['A', 'D', 'E']

1. Pop an element from the stack: `stack.pop()`

Stack: ['A', 'D']

1. Pop an element from the stack: `stack.pop()`

Stack: ['A']

After the last Invocation, the stack contains the element 'A'.

٢\_Suppose an initially empty ArrayStack S has performed a total of 25 push

Operations, 12 top operations, and 10 pop operations, 3 of which returned null to

Indicate an empty stack. What is the current size of S? And what is the value of

The instance variable t?To determine the current size of the ArrayStack `S` and the value of the instance variable `t`, we need to consider the operations performed on the stack.

Given:

- Total push operations: 25

- Total top operations: 12

- Total pop operations: 10 (3 of which returned null)

The size of the stack (`S`) can be calculated as follows:

Size of S = Total push operations - Total pop operations + number of null returns

Size of S = 25 - 10 + 3 = 18

Therefore, the current size of the ArrayStack `S` is 18.

The instance variable `t` represents the index of the top element in the stack. Since the stack is zero-based (the first element is at Index 0), `t` will be equal to the current size of `S` minus 1.

T = Size of S – 1 = 18 – 1 = 17

Therefore, the value of the instance variable `t` is 17.

٣.- Evaluate the following postfix expressions (true or false):

1. 8 2 + 3 \* 16 4 / - =

b. 12 2 5 5 1 / / \* 8 7 + - =

c. 70 14 4 5 15 3 / \* - / 6 + =

d. 3 5 6 \* + 13 - 18 2 / + =

Let's evaluate the given postfix expressions and determine whether they are true or false:

1. 8 2 + 3 \* 16 4 / - =

1. Push 8 onto the stack: [8]

2. Push 2 onto the stack: [8, 2]

3. Perform addition: 8 + 2 = 10, Push 10 onto the stack: [10]

4. Push 3 onto the stack: [10, 3]

5. Perform multiplication: 10 \* 3 = 30, Push 30 onto the stack: [30]

6. Push 16 onto the stack: [30, 16]

7. Push 4 onto the stack: [30, 16, 4]

8. Perform division: 16 / 4 = 4, Push 4 onto the stack: [30, 4]

9. Perform subtraction: 30 - 4 = 26, Push 26 onto the stack: [26]

The final result is 26. Therefore, the expression "8 2 + 3 \* 16 4 / - =" is true.

b. 12 2 5 5 1 / / \* 8 7 + - =

1. Push 12 onto the stack: [12]

2. Push 2 onto the stack: [12, 2]

3. Push 5 onto the stack: [12, 2, 5]

4. Push 5 onto the stack: [12, 2, 5, 5]

5. Push 1 onto the stack: [12, 2, 5, 5, 1]

6. Perform division: 5 / 1 = 5, Push 5 onto the stack: [12, 2, 5, 5]

7. Perform division: 5 / 5 = 1, Push 1 onto the stack: [12, 2, 1]

8. Perform multiplication: 2 \* 1 = 2, Push 2 onto the stack: [12, 2]

9. Push 8 onto the stack: [12, 2, 8]

10. Push 7 onto the stack: [12, 2, 8, 7]

11. Perform addition: 8 + 7 = 15, Push 15 onto the stack: [12, 2, 15]

12. Perform subtraction: 2 - 15 = -13, Push -13 onto the stack: [12, -13]

The final result is -13. Therefore, the expression "12 2 5 5 1 / / \* 8 7 + - =" is true.

c. 70 14 4 5 15 3 / \* - / 6 + =

1. Push 70 onto the stack: [70]

2. Push 14 onto the stack: [70, 14]

3. Push 4 onto the stack: [70, 14, 4]

4. Push 5 onto the stack: [70, 14, 4, 5]

5. Push 15 onto the stack: [70, 14, 4, 5, 15]

6. Push 3 onto the stack: [70, 14, 4, 5, 15, 3]

7. Perform division: 15 / 3 = 5, Push 5 onto the stack: [70, 14, 4, 5, 5]

8. Perform multiplication: 5 \* 5 = 25, Push 25 onto the stack: [70, 14, 4, 25]

9. Perform subtraction: 4 – 25 = -21, Push -21 onto the stack: [70, 14, -21]

10. Perform division: 14 / -21 = 0, Push 0 onto the stack: [70, 0]

11. Push 6 onto the stack: [70, 0, 6]

12. Perform addition: 0 + 6 = 6, Push 6 onto the stack: [70, 6]

The final result is 6. Therefore, the expression "70 14 4 5 15 3 / \* - / 6 + =" Is true.

D. 3 5 6 \* + 13 - 18 2 / + =

1. Push 3 onto the stack: [3]

2. Push 5 onto the stack: [3, 5]

3. Push 6 onto the stack: [3, 5, 6]

4. Perform multiplication: 5 \* 6 = 30, Push 30 onto the stack: [3, 30]

5. Perform addition: 3 + 30 = 33, Push 33 onto the stackApologies for the incomplete response. Let's continue evaluating the expression:

D. 3 5 6 \* + 13 - 18 2 / + =

1. Push 3 onto the stack: [3]

2. Push 5 onto the stack: [3, 5]

3. Push 6 onto the stack: [3, 5, 6]

4. Perform multiplication: 5 \* 6 = 30, Push 30 onto the stack: [3, 30]

5. Perform addition: 3 + 30 = 33, Push 33 onto the stack: [33]

6. Push 13 onto the stack: [33, 13]

7. Perform subtraction: 33 - 13 = 20, Push 20 onto the stack: [20]

8. Push 18 onto the stack: [20, 18]

9. Push 2 onto the stack: [20, 18, 2]

10. Perform division: 18 / 2 = 9, Push 9 onto the stack: [20, 9]

11. Perform addition: 20 + 9 = 29, Push 29 onto the stack: [29]

The final result is 29. Therefore, the expression "3 5 6 \* + 13 - 18 2 / + =" Is true.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

٥.Convert the following infix expressions to postfix notations, and convert the first

Two postfix notations to java code using stack operations:

1. (A + B) \* (C + D) – E

b. A – (B + C) \* D + E / F

c. ((A + B) / (C - D) + E) \* F - G

d. A + B \* (C + D) – E / F \* G + H

import java.util.Stack;

public class PostfixEvaluation {

public static int evaluatePostfix(String postfix) {

Stack<Integer> stack = new Stack<>();

for (char c : postfix.toCharArray()) {

if (Character.isLetterOrDigit(c)) {

// Operand, push into stack

stack.push(Character.getNumericValue(c));

} else {

// Operator, pop operands from stack and perform operation

int operand2 = stack.pop();

int operand1 = stack.pop();

int result = 0;

switch (c) {

case '+':

result = operand1 + operand2;

break;

case '-':

result = operand1 - operand2;

break;

case '\*':

result = operand1 \* operand2;

break;

case '/':

result = operand1 / operand2;

break;

}

// Push the result back to the stack

stack.push(result);

}

}

// The final result will be at the top of the stack

return stack.pop();

}

public static void main(String[] args) {

String postfix = "AB+CD+\*E-";

int result = evaluatePostfix(postfix);

System.out.println("Result: " + result);

}

}

Import java.util.Stack;

Public class PostfixEvaluation {

Public static int evaluatePostfix(String postfix) {

Stack<Integer> stack = new Stack<>();

For (char c : postfix.toCharArray()) {

If (Character.isLetterOrDigit©) {

// Operand, push into stack

Stack.push(Character.getNumericValue©);

} else {

// Operator, pop operands from stack and perform operation

Int operand2 = stack.pop();

Int operand1 = stack.pop();

Int result = 0;

Switch © {

Case '+':

Result = operand1 + operand2;

Break;

Case '-':

Result = operand1 – operand2;

Break;

Case '\*':

Result = operand1 \* operand2;

Break;

Case '/':

Result = operand1 / operand2;

Break;

}

// Push the result back to the stack

Stack.push(result);

}

}

// The final result will be at the top of the stack

Return stack.pop();

}

Public static void main(String[] args) {

String postfix = "AB+CD+\*E-";

Int result = evaluatePostfix(postfix);

System.out.println("Result: " + result);

}

}

٦. Write this client method using only the push(), top(), pop(), and isEmpty()

methods:

public static <E> void reverse(ArrayStack<E> stack)

// reverses the contents of the specified stac

import java.util.ArrayDeque;

import java.util.Deque;

public class Client {

public static <E> void reverse(ArrayStack<E> stack) {

Deque<E> deque = new ArrayDeque<>();

while (!stack.isEmpty()) {

deque.push(stack.pop());

}

while (!deque.isEmpty()) {

stack.push(deque.pop());

}

}

public static void main(String[] args) {

ArrayStack<Integer> stack = new ArrayStack<>();

stack.push(1);

stack.push(2);

stack.push(3);

stack.push(4);

System.out.println("Original stack: " + stack);

reverse(stack);

System.out.println("Reversed stack: " + stack);

}

}

٧. Write this client method using only the push(), top(), pop(), and isEmpty()

methods:

public static <E> E popBottom(LinkedStack<E> stack)

// removes and returns the bottom element of the specified stack

public class Client {

public static <E> E popBottom(LinkedStack<E> stack) {

if (stack.isEmpty()) {

throw new IllegalStateException("Stack is empty");

}

if (stack.size() == 1) {

return stack.pop();

}

E topElement = stack.pop();

E bottomElement = popBottom(stack);

stack.push(topElement);

return bottomElement;

}

public static void main(String[] args) {

LinkedStack<Integer> stack = new LinkedStack<>();

stack.push(1);

stack.push(2);

stack.push(3);

stack.push(4);

System.out.println("Original stack: " + stack);

Integer bottomElement = popBottom(stack);

System.out.println("Bottom element: " + bottomElement);

System.out.println("Modified stack: " + stack);

}

}

٨. Add this member method to the ArrayStack class :

public E topSecond()

// returns the second from the top element of this stack

Public class ArrayStack<E> {

Private static final Int DEFAULT\_CAPACITY = 10;

Private E[] elements;

Private int size;

Public ArrayStack() {

Elements = (E[]) new Object[DEFAULT\_CAPACITY];

Size = 0;

}

Public void push(E element) {

If (size == elements.length) {

Resize(2 \* elements.length);

}

Elements[size] = element;

Size++;

}

Public E pop() {

If (isEmpty()) {

Throw new IllegalStateException("Stack is empty");

}

E element = elements[size - 1];

Elements[size – 1] = null;

Size--;

If (size > 0 && size == elements.length / 4) {

Resize(elements.length / 2);

}

Return element;

}

Public E top() {

If (isEmpty()) {

Throw new IllegalStateException("Stack is empty");

}

Return elements[size – 1];

}

Public boolean isEmpty() {

Return size == 0;

}

Public int size() {

Return size;

}

Public E topSecond() {

If (size < 2) {

Throw new IllegalStateException("Stack does not have a second element");

}

Return elements[size – 2];

}

Private void resize(int capacity) {

E[] newArray = (E[]) new Object[capacity];

For (int I = 0; I < size; i++) {

newArray[i] = elements[i];

}

ArrayStack<Integer> stack = new ArrayStack<>();

Stack.push(1);

Stack.push(2);

Stack.push(3);

Int secondElement = stack.topSecond();

System.out.println("Second element: " + secondElement); // Output: Second element: 2Elements

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

٩.Add this member method to the ArrayStack class :

Public E popSecond()

// removes and returns the second element of this stack

Public class ArrayStack<E> {

Private static final Int DEFAULT\_CAPACITY = 10;

Private E[] elements;

Private int size;

Public ArrayStack() {

Elements = (E[]) new Object[DEFAULT\_CAPACITY];

Size = 0;

}

Public void push(E element) {

If (size == elements.length) {

Resize(2 \* elements.length);

}

Elements[size] = element;

Size++;

}

Public E pop() {

If (isEmpty()) {

Throw new IllegalStateException("Stack is empty");

}

E element = elements[size - 1];

Elements[size – 1] = null;

Size--;

If (size > 0 && size == elements.length / 4) {

Resize(elements.length / 2);

}

Return element;

}

Public E top() {

If (isEmpty()) {

Throw new IllegalStateException("Stack is empty");

}

Return elements[size – 1];

}

Public boolean isEmpty() {

Return size == 0;

}

Public int size() {

Return size;

}

Public E topSecond() {

If (size < 2) {

Throw new IllegalStateException("Stack does not have a second element");

}

Return elements[size – 2];

}

Public E popSecond() {

If (size < 2) {

Throw new IllegalStateException("Stack does not have a second element");

}

E secondElement = elements[size – 2];

For (int I = size – 2; I < size – 1; i++) {

Elements[i] = elements[I + 1];

}

Elements[size – 1] = null;

Size--;

Return secondElement;

}

Private void resize(int capacity) {

E[] newArray = (E[]) new Object[capacity];

For (int I = 0; I < size; i++) {

newArray[i] = elements[i];

}

Elements = newArray;

}

}

١٠.Add this member method to the LinkedStack class:

Public E bottom()

// returns the bottom element of this stack

Public class LinkedStack<E> {

Private Node<E> top;

Private int size;

Public LinkedStack() {

Top = null;

Size = 0;

}

Public void push(E element) {

Node<E> newNode = new Node<>(element);

If (isEmpty()) {

Top = newNode;

} else {

newNode.next = top;

top = newNode;

}

Size++;

}

Public E pop() {

If (isEmpty()) {

Throw new IllegalStateException("Stack is empty");

}

E element = top.element;

Top = top.next;

Size--;

Return element;

}

Public E top() {

If (isEmpty()) {

Throw new IllegalStateException("Stack is empty");

}

Return top.element;

}

Public boolean isEmpty() {

Return size == 0;

}

Public int size() {

Return size;

}

Public E bottom() {

If (isEmpty()) {

Throw new IllegalStateException("Stack is empty");

}

Node<E> current = top;

While (current.next != null) {

Current = current.next;

}

Return current.element;

}

Private static class Node<E> {

Private E element;

Private Node<E> next;

Public Node(E element) {

This.element = element;

This.next = null;

}

}

}

١١.Add this member method to the ArrayStack class:

Public E popbottom()

// removes and returns the bottom element of this stack

Public class ArrayStack<E> {

Private static final Int DEFAULT\_CAPACITY = 10;

Private E[] elements;

Private int size;

Public ArrayStack() {

Elements = (E[]) new Object[DEFAULT\_CAPACITY];

Size = 0;

}

Public void push(E element) {

If (size == elements.length) {

Resize(2 \* elements.length);

}

Elements[size] = element;

Size++;

}

Public E pop() {

If (isEmpty()) {

Throw new IllegalStateException("Stack is empty");

}

E element = elements[size - 1];

Elements[size – 1] = null;

Size--;

If (size > 0 && size == elements.length / 4) {

Resize(elements.length / 2);

}

Return element;

}

Public E top() {

If (isEmpty()) {

Throw new IllegalStateException("Stack is empty");

}

Return elements[size – 1];

}

Public boolean isEmpty() {

Return size == 0;

}

Public int size() {

Return size;

}

Public E topSecond() {

If (size < 2) {

Throw new IllegalStateException("Stack does not have a second element");

}

Return elements[size – 2];

}

Public E popSecond() {

If (size < 2) {

Throw new IllegalStateException("Stack does not have a second element");

}

E secondElement = elements[size – 2];

For (int I = size – 2; I < size – 1; i++) {

Elements[i] = elements[I + 1];

}

Elements[size – 1] = null;

Size--;

Return secondElement;

}

Public E popbottom() {

If (isEmpty()) {

Throw new IllegalStateException("Stack is empty");

}

E bottomElement = elements[0];

For (int I = 0; I < size – 1; i++) {

Elements[i] = elements[I + 1];

}

Elements[size – 1] = null;

Size--;

If (size > 0 && size == elements.length / 4) {

Resize(elements.length / 2);

}

Return bottomElement;

}

Private void resize(int capacity) {

E[] newArray = (E[]) new Object[capacity];

For (int I = 0; I < size; i++) {

newArray[i] = elements[i];

}

Elements = newArray;

}

}

١٢. Consider the following segment code with the following informations:

- Assume (capacity = 10, size = 0,

top = 0)

After execution of this code..

a) What are the contents (elements)

of the stack?

b) What are the values of the variables

count, top?

c) What are the element of the

top( ) method in the stack?

d) Is the stack full? Why?

e) Make the stack return to the empty state?

Public static void main (string []args)

{

Stack<int> stack =new ArrayStack (10);

for (int i=1; i<=10; i++)

if (i % 3 != 0)

{ stack.push(i\* 2); }

else

{

stack.popa) After the execution of the code, the contents (elements) of the stack will be [2, 4, 8, 10, 14, 16, 20].

b) The value of the variable `size` would be 7 since there are 7 elements in the stack. The value of the variable `top` would be 7, indicating the index of the topmost element in the stack (assuming 0-based indexing).

c) The `top()` method In the stack would return the element at the top of the stack, which is 20 In this case.

d) The stack Is not full. The capacity of the stack Is set to 10, and the current size Is 7. Since the size is less than the capacity, the stack Is not full.

e) To make the stack return to the empty state, you can call the `pop()` method repeatedly until the stack becomes empty. Here's an example:

```java

While (!stack.isEmpty()) {

Stack.pop();

}

```

After executing this code, the stack will be empty, and the `size` and `top` variables will be reset to their initial values (size = 0, top = 0).