Stack

1)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();

1. After `stack.push(new Character('A'))`:
   Stack: [A]
2. After `stack.push(new Character('B'))`:
   Stack: [A, B]
3. After `stack.push(new Character('C'))`:
   Stack: [A, B, C]
4. After `stack.pop()`:
   Stack: [A, B]
5. After another `stack.pop()`:
   Stack: [A]
6. After `stack.push(new Character('D'))`:
   Stack: [A, D]
7. After `stack.push(new Character('E'))`:
   Stack: [A, D, E]
8. After `stack.push(new Character('F'))`:
   Stack: [A, D, E, F]
9. After `stack.pop()`:
   Stack: [A, D, E]
10. After `stack.push(new Character('G'))`:
    Stack: [A, D, E, G]
11. After another `stack.pop()`:
    Stack: [A, D, E]
12. After another `stack.pop()`:
    Stack: [A, D]
13. After the final `stack.pop()`:
    Stack: [A]

2- 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?

Current size of S = 25 - 10
Current size of S = 15

Since there have been 15 elements pushed onto the stack and 7 elements popped, the index of the top element would be 15 - 7 - 1 (subtracting 1 since the index is zero-based).
Value of t = 15 - 7 - 1
Value of t = 7

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

a. 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 / + =

a. 8 2 + 3 * 16 4 / - =
الخطوات:
8 + 2 = 10
3 * 10 = 30
16 / 4 = 4
30 - 4 = 26
إذاً، قيمة التعبير هي 26.
الجواب: صحيح (True)
b. 12 2 5 5 1 / / * 8 7 + - =
الخطوات:
5 / 1 = 5
5 / 5 = 1
2 * 1 = 2
12 / 2 = 6
8 + 7 = 15
6 - 15 = -9

9- إذًا، قيمة التعبير هي.
الجواب: صحيح (True)
c. 70 14 4 5 15 3 / * - / 6 + =
الخطوات:
15 / 3 = 5
5 * 5 = 25
4 - 25 = -21
14 / -21 = 0 (تقريبًا)
70 / 0 = ∞ (لا نستطيع تقسيم عدد على الصفر)
إذًا، قيمة التعبير غير معرفة.
الجواب: خاطئ (False)
d. 3 5 6 * + 13 - 18 2 / + =
الخطوات:
5 * 6 = 30
3 + 30 = 33
33 - 13 = 20
18 / 2 = 9
20 + 9 = 29
إذًا، قيمة التعبير هي 29.
الجواب: صحيح (True)

4)Convert the following infix expressions to postfix notations, and convert the first

:two postfix notations to java code using stack operations

a. (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

```
Stack<Character> stack = new Stack<>();
StringBuilder postfix = new StringBuilder();
String expression = "(A + B) * (C + D) - E";

for (char c : expression.toCharArray()) {
    if (Character.isLetterOrDigit(c)) {
        postfix.append(c);
    } else if (c == '(') {
        stack.push(c);
    } else if (c == ')') {
        while (!stack.isEmpty() && stack.peek() != '(') {
            postfix.append(stack.pop());
        }
        stack.pop();
    } else {
        while (!stack.isEmpty() && precedence(c) <= precedence(stack.peek())) {
            postfix.append(stack.pop());
        }
        stack.push(c);
    }
}

while (!stack.isEmpty()) {
    postfix.append(stack.pop());
}

String postfixExpression = postfix.toString();
System.out.println(postfixExpression);
```
Note: The `precedence()` method is used to determine the precedence of operators.

b. A - (B + C) * D + E / F
Postfix notation: ABC+D*-EF/+
Java code using stack operations:
````java
Stack<Character> stack = new Stack<>();
StringBuilder postfix = new StringBuilder();
String expression = "A - (B + C) * D + E / F";

for (char c : expression.toCharArray()) {
    if (Character.isLetterOrDigit(c)) {
        postfix.append(c);
    } else if (c == '(') {
        stack.push(c);
    } else if (c == ')') {
        while (!stack.isEmpty() && stack.peek() != '(') {
            postfix.append(stack.pop());
        }
```

```
        stack.pop();
    } else {
        while (!stack.isEmpty() && precedence(c) <= precedence(stack.peek())) {
            postfix.append(stack.pop());
        }
        stack.push(c);
    }
}

while (!stack.isEmpty()) {
    postfix.append(stack.pop());
}

String postfixExpression = postfix.toString();
System.out.println(postfixExpression);
```

c. ((A + B) / (C - D) + E) * F - G
Postfix notation: AB+CD-/E+F*G-
Java code using stack operations:
````java
Stack<Character> stack = new Stack<>();
StringBuilder postfix = new StringBuilder();
String expression = "((A + B) / (C - D) + E) * F - G";

for (char c : expression.toCharArray()) {
    if (Character.isLetterOrDigit(c)) {
        postfix.append(c);
    } else if (c == '(') {
        stack.push(c);
    } else if (c == ')') {
        while (!stack.isEmpty() && stack.peek() != '(') {
            postfix.append(stack.pop());
        }
        stack.pop();
    } else {
        while (!stack.isEmpty() && precedence(c) <= precedence(stack.peek())) {
            postfix.append(stack.pop());
        }
        stack.push(c);
    }
}

while (!stack.isEmpty()) {
    postfix.append(stack.pop());
}

String postfixExpression = postfix.toString();
System.out.println(postfixExpression);
```

d. A + B * (C + D) - E / F * G + H
Postfix notation: ABCD+*+EF/G*-H+
Java code using stack operations:
````java
````

```
Stack<Character> stack = new Stack<>();
StringBuilder postfix = new StringBuilder();
String expression = "A + B * (C + D) - E / F * G + H";

for (char c : expression.toCharArray()) {
    if (Character.isLetterOrDigit(c)) {
        postfix.append(c);
    } else if (c == '(') {
        stack.push(c);
    } else if (c == ')') {
        while (!stack.isEmpty() && stack.peek() != '(') {
            postfix.append(stack.pop());
        }
        stack.pop();
    } else {
        while (!stack.isEmpty() && precedence(c) <= precedence(stack.peek())) {
            postfix.append(stack.pop());
        }
        stack.push(c);
    }
}

while (!stack.isEmpty()) {
    postfix.append(stack.pop());
}

String postfixExpression = postfix.toString();
System.out.println(postfixExpression);
```

5- Write the definition of the function template printListReverse that uses a stack to print a linked list in reverse order. Assume that this function is a member of the class linkedStack,

```
template <class T>
class linkedStack {
private:
    struct Node {
        T data;
        Node* next;
    };

    Node* top;

public:
    // Other member functions of linkedStack

    void printListReverse() {
        std::stack<T> stack;
        Node* temp = top;

        // Push elements of linked list onto the stack
        while (temp != nullptr) {
            stack.push(temp->data);
            temp = temp->next;
        }
```

```
        // Print elements in reverse order
        while (!stack.empty()) {
            std::cout << stack.top() << " ";
            stack.pop();
        }
    }
};
```

6- Write this client method using only the push(), top(), pop(), and isEmpty() methods: public static void reverse(ArrayStack stack) // reverses the contents of the specified stack

```
public static <E> void reverseStack(ArrayStack<E> stack) {
    if (stack.isEmpty()) {
        return; // إذا كان المكدس فارغًا، لا يوجد أي شيء لعكسه
    }

    E element = stack.pop(); // قم بإزالة العنصر الأعلى من المكدس
    reverseStack(stack); // قم بتنفيذ الدالة بشكل متكرر لعكس باقي المكدس

    insertAtBottom(stack, element); // قم بإدراج العنصر الحالي في أسفل المكدس
}

private static <E> void insertAtBottom(ArrayStack<E> stack, E element) {
    if (stack.isEmpty()) {
        stack.push(element); // إذا كان المكدس فارغًا، قم بإدراج العنصر فيه
    } else {
        E top = stack.pop(); // قم بإزالة العنصر الأعلى من المكدس
        insertAtBottom(stack, element); // قم بتنفيذ الدالة بشكل متكرر لإدراج العن
صر في أسفل المكدس
        stack.push(top); // قم بإعادة إدراج العنصر الأعلى في المكدس
    }
}
```

7- Write this client method using only the push(), top(), pop(), and isEmpty() methods: public static E popBottom(LinkedStack stack) // removes and returns the bottom element of the specified stack

```
public static <E> E popBottom(LinkedStack<E> stack) {
    if (stack.isEmpty()) {
        throw new EmptyStackException(); // إذا كان المكدس فارغًا، يتم رمي استثناء
    }

    LinkedStack<E> tempStack = new LinkedStack<>(); // مكدس مؤقت لتخزين العناصر ال
مؤقتة

    while (!stack.isEmpty()) {
        tempStack.push(stack.pop()); // نقل العناصر من المكدس الأصلي إلى المكدس ال
مؤقت
    }

    E bottomElement = tempStack.pop(); // إزالة واسترداد العنصر الأسفل من المكدس ا
لمؤقت
```

```
    while (!tempStack.isEmpty()) {
        stack.push(tempStack.pop()); // نقل العناصر المؤقتة إلى المكدس الأصلي
    }

    return bottomElement; // إرجاع العنصر الأسفل
}
```

8- Add this member method to the ArrayStack class : public E topSecond() // returns the second from the top element of this stack

```
public E topSecond() {
    if (size < 2) {
        throw new EmptyStackException(); //  إذا كان حجم المكدس أقل من 2، يتم رمي استثناء
    }

    return elements[size - 2]; // إرجاع العنصر الثاني من الأعلى
}
```

9- Add this member method to the ArrayStack class : public E popSecond() // removes and returns the second element of this stack

```
public E popSecond() {
    if (size < 2) {
        throw new EmptyStackException(); //  إذا كان حجم المكدس أقل من 2، يتم رمي استثناء
    }

    E secondElement = elements[size - 2]; // العنصر الثاني من الأعلى

    // نقل العناصر للأمام لإزالة العنصر الثاني
    for (int i = size - 2; i < size - 1; i++) {
        elements[i] = elements[i + 1];
    }

    size--; // تحديث حجم المكدس

    return secondElement; // إرجاع العنصر الثاني
}
```

10- Add this member method to the LinkedStack class: public E bottom() // returns the bottom element of this stack

```
public E bottom() {
    if (isEmpty()) {
        throw new EmptyStackException(); // إذا كان المكدس فارغًا، يتم رمي استثناء
    }

    Node<E> currentNode = top;
```

```
    while (currentNode.next != null) {
        currentNode = currentNode.next; // التنقل إلى العنصر التالي في الرابطة
    }

    return currentNode.data; // إرجاع العنصر السفلي
}
```

11- Add this member method to the ArrayStack class: public E popbottom() // removes and returns the bottom element of this stack

لا يمكن تنفيذها

12- Consider the following segment code with the following informations: - Assume (capacity = 10, size = 0, top = 0) After execution of this code.. (string []args) { Stack stack =new ArrayStack (10); for (int i=1; i<=10; i++) if (i % 3 != 0) { stack.push(i* 2); } else { stack.pop(); }

a) What are the contents (elements) of the stack?

[2, 4, 8, 10, 14, 16, 20]

b) What are the values of the variables count, top?

المتغير countلم يتم ذكره بشكل واضح في قطعة الكود، ولكن من الظاهر أنه يمثل عدد العناصر في المكدس. في هذه الحالة، قيمة countستكون 7.

c) What are the element of the top( ) method in the stack?

هو العنصر رقم 20

d) Is the stack full? Why?

لا، المكدس ليس ممتلئًا. السعة القصوى للمكدس هي 10، وحجم المكدس بعد تنفيذ قطعة الكود هو **7**

e) Make the stack return to the empty state?

```
stack.clear();
```