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

Stack stack = new Stack();

Empty

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

A

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

B,A

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

C,B,A

stack.pop(); stack.pop();

B,A

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

A,D

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

F,E,D,A

stack.pop();

E,D,A

stack.push(new Character('G')); stack.pop();  
G,E,D,A

stack.pop(); stack.pop();

E,D,A

D,A

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**?

Arrystack size and lnstance variable

Current size of s:5(25 pushes -10 successful pops)

Value of instance variable t:4(index of the top element

1. Evaluate the following postfix expressions (true or false):
   1. 8 2 + 3 \* 16 4 / - =
   2. 12 2 5 5 1 / / \* 8 7 + - =
   3. 70 14 4 5 15 3 / \* - / 6 + =
   4. 3 5 6 \* + 13 - 18 2 / + =
2. True
3. False
4. True
5. True
6. 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
   1. A - (B + C) \* D + E / F
   2. ((A + B) / (C - D) + E) \* F - G
   3. A + B \* (C + D) - E / F \* G + H
7. import java.util.Stack;
8. public class PostfixEvaluation {
9. public static int evaluatePostfix(String postfix) {
10. Stack<Integer> stack = new Stack<>();
11. for (int i = 0; i < postfix.length(); i++) {
12. char c = postfix.charAt(i);
13. if (Character.isDigit(c)) {
14. stack.push(c - '0');
15. } else {
16. int operand2 = stack.pop();
17. int operand1 = stack.pop();
18. switch (c) {
19. case '+':
20. stack.push(operand1 + operand2);
21. break;
22. case '-':
23. stack.push(operand1 - operand2);
24. break;
25. case '\*':
26. stack.push(operand1 \* operand2);
27. break;
28. case '/':
29. stack.push(operand1 / operand2);
30. break;
31. }
32. }
33. }
34. return stack.pop();
35. }
36. public static void main(String[] args) {
37. String postfix = "AB+CD+\*E-";
38. int result = evaluatePostfix(postfix);
39. System.out.println("Result: " + result);
40. }
41. 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**,
42. template <typename T>
43. void linkedStack<T>::printListReverse() const {
44. stack<T> s;
45. nodeType<T>\* current = this->stackTop;
46. while (current != nullptr) {
47. s.push(current->info);
48. current = current->link;
49. }
50. while (!s.empty()) {
51. cout << s.top() << " ";
52. s.pop();
53. }
54. }
55. 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 stack

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

Stack<E> tempStack = new Stack<>();

while (!stack.isEmpty()) {

tempStack.push(stack.pop());

}

while (!tempStack.isEmpty()) {

stack.push(tempStack.pop());

}

}

1. Write this client method using only the push(), top(), pop(), and isEmpty() methods:
2. **public static <E> E popBottom(LinkedStack<E> stack) // removes and returns the bottom element of the specified stack**

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

1. Stack<E> tempStack = new Stack<>();
2. while (!stack.isEmpty()) {
3. tempStack.push(stack.pop());
4. }
5. E bottomElement = tempStack.pop();
6. while (!tempStack.isEmpty()) {
7. stack.push(tempStack.pop());
8. }
9. return bottomElement;
10. 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 (this.size() < 2) {

throw new NoSuchElementException("Stack has less than two elements");

}

E top = this.pop();

E second = this.top();

this.push(top);

return second;

}

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 (this.size() < 2) {

throw new NoSuchElementException("Stack has less than two elements");

}

E top = this.pop();

E second = this.pop();

this.push(top);

return second;

}

10- Add this member method to the LinkedStack class: **public E bottom()**

# // returns the bottom element of this stack

public E bottom() {

if (this.isEmpty()) {

throw new NoSuchElementException("Stack is empty");

}

nodeType<E> current = this.top;

while (current.link != null) {

current = current.link;

}

return current.info;

}

11- Add this member method to the ArrayStack class: **public E popbottom()**

# // removes and returns the bottom element of this stack

public E popbottom() {

// check if the stack is empty

if (isEmpty()) {

throw new EmptyStackException();

}

// get the bottom element

E bottom = data[0];

// shift the elements to the left

for (int i = 0; i < t; i++) {

}

// decrement the top index

t--;

// return the bottom element

return bottom;

}

# 

12- **Consider the following segment code with the following informations:**

Public static void main (string []args)

- Assume (capacity = 10, size = 0, {

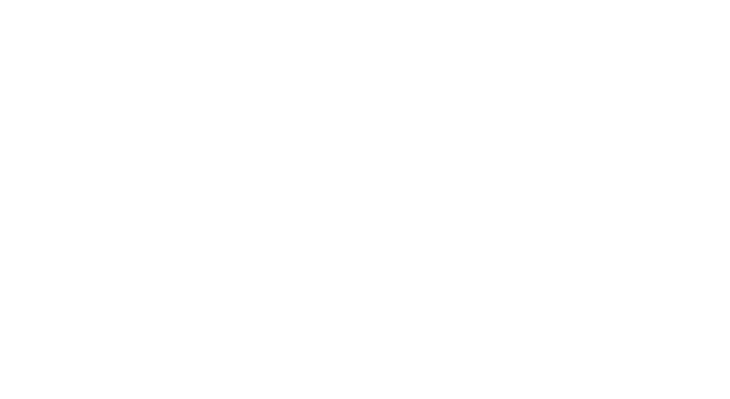
Stack<int> stack =new ArrayStack (10);

top = 0)

After execution of this code.. for (int i=1; i<=10; i++)

if (i % 3 != 0)

1. What are the contents (elements) { stack.push(i\* 2); } of the stack? else



{

1. What are the values of the variables stack.pop(); }

*}* count, top?

1. What are the element of the **top( )** method in the stack?
2. Is the stack full? Why?
3. Make the stack return to the empty state?

* Here are the answers to your questions:
* contents of the stack:
* **This because the loop pushes only the even**
* **numbers that are not divisible by 3** **(2, 4, 8, 10, 14, 16, 20) onto** the stack
* values of the variables count,top
* count: There is no variable named "count" in the given code.
* top: The value of the variable "top" will be 7, indicating the index of the next available position in the stack.
* c) Element at the top of the stack:
* The element at the top of the stack, returned by the top() method, will be 20, which is the last
* element that was pushed onto the stack
* d) Is the stack full? Why?
* No, the stack is not full. Its capacity is 10, and it currently contains only 7 elements.

# e) Make the stack return to the empty state:

1. while (!stack.isEmpty()) {
2. stack.pop();
3. }