Q:-1 Given below is the Node class to perform basic list operations and a Stack class with a no arg constructor.
Select from the options the appropriate pop() operation that can be included in the Stack class. Also 'first' is the top-of-the-stack.
advertisement

```
class Node
{
	protected Node next;
	protected Object ele;
	Node()
	{
		this(null,null);
	}
	Node(Object e,Node n)
	{
		ele=e;
		next=n;
	}
	public void setNext(Node n)
	{
		next=n;
	}
	public void setEle(Object e)
	{
		ele=e;
	}
	public Node getNext()
	{
		return next;
	}
	public Object getEle()
	{
		return ele;
	}
}

class Stack
{
	Node first;
	int size=0;
	Stack()
	{
		first=null;
	}
}
```

a)

```
public Object pop()
{
	if(size == 0)
	System.out.println("underflow");
	else
	{
		Object o = first.getEle();
		first = first.getNext();
		size--;
		return o;
	}
}
b)
public Object pop()
{
	if(size == 0)
	System.out.println("underflow");
	else
	{
		Object o = first.getEle();
		first = first.getNext().getNext();
		size--;
		return o;
	}
}
c)
public Object pop()
{
	if(size == 0)
	System.out.println("underflow");
	else
	{
		first = first.getNext();
		Object o = first.getEle();
		size--;
		return o;
	}
}
d)
public Object pop()
{
	if(size == 0)
	System.out.println("underflow");
	else
	{
		first = first.getNext().getNext();
```

```
            Object o = first.getEle();
            size--;
            return o;
      }
}
```

View Answer

Answer: a

Explanation: pop() should return the Object pointed to by the node ‘first’. The sequence of operations is, first, get the element stored at node ‘first’ using getEle(), and second, make the node point to the next node using getNext().

Q:-2. What does the following function do?

```
public Object some_func()throws emptyStackException
{
      if(isEmpty())
            throw new emptyStackException("underflow");
      return first.getEle();
}
```

a) pop

b) delete the top-of-the-stack element

c) retrieve the top-of-the-stack element

d) push operation

Answer: c

Explanation: This code is only retrieving the top element, note that it is not equivalent to pop operation as you are not setting the ‘next’ pointer point to the next node in sequence.

Q:-3. What is the functionality of the following piece of code?

```
public void display()
{
      if(size == 0)
            System.out.println("underflow");
      else
      {
            Node current = first;
            while(current != null)
            {
                  System.out.println(current.getEle());
                  current = current.getNext();
            }
      }
}
```

a) reverse the list
b) display the list
c) display the list excluding top-of-the-stack-element
d) reverse the list excluding top-of-the-stack-element
Answer: b
Explanation: An alias of the node 'first' is created which traverses through the list and displays the elements.

Q:-4. Given below is the Node class to perform basic list operations and a Stack class with a no arg constructor. Select from the options the appropriate push() operation that can be included in the Stack class. Also 'first' is the top-of-the-stack.

```
class Node
{
	protected Node next;
	protected Object ele;
	Node()
	{
		this(null,null);
	}
	Node(Object e,Node n)
	{
		ele=e;
		next=n;
	}
	public void setNext(Node n)
	{
		next=n;
	}
	public void setEle(Object e)
	{
		ele=e;
	}
	public Node getNext()
	{
		return next;
	}
	public Object getEle()
	{
		return ele;
	}
}
class Stack
{
	Node first;
	int size=0;
```

```
	Stack()
	{
		first=null;
	}
}
a)
public void push(Object item)
{
	Node temp = new Node(item,first);
	first = temp;
	size++;
}
b)
public void push(Object item)
{
	Node temp = new Node(item,first);
	first = temp.getNext();
	size++;
}
c)
public void push(Object item)
{
	Node temp = new Node();
	first = temp.getNext();
	first.setItem(item);
	size++;
}
d)
public void push(Object item)
{
	Node temp = new Node();
	first = temp.getNext.getNext();
	first.setItem(item);
	size++;
}
```

Answer: a

Explanation: To push an element into the stack, first create a new node with the next pointer point to the current top-of-the-stack node, then make this node as top-of-the-stack by assigning it to 'first'.

Q:-5. Assume that the operators +,-, x are left associative and ^ is right associative. The order of precedence (from highest to lowest) is ^, x, +, -. The postfix expression for the infix expression a + b x c – d ^ e ^ f is?

a) a b c x + d e f ^ ^ –

b) a b c x + d e ^ f ^ –
c) a b + c x d – e ^ f ^
d) – + a x b c ^ ^ d e f

Answer: a
Explanation: Given Infix Expression is a + b x c – d ^ e ^ f. And ^ is right associative. Thus, the final postfix expression is a b c x + d e f ^ ^ –

Q:-6 Find the output of following code.

```
import java.util.Stack;
public class Test {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.pop();
        stack.push(40);
        while (!stack.isEmpty()) {
            System.out.print(stack.pop() + " ");
        }
    }
}
```

A) 40 20 10
B) 10 20 40
C) 30 20 10
D) 10 20 30
Correct Answer: A) 40 20 10
Explanation: After pushing 10, 20, 30 → pop → push 40 → pop all = 40 20 10

Q:-7 Find the output of following code.

```
import java.util.Stack;
public class Test {
    public static void main(String[] args) {
        Stack<Character> stack = new Stack<>();
        String s = "XYZ";
        for (char c : s.toCharArray()) {
            stack.push(c);
        }
        StringBuilder sb = new StringBuilder();
        while (!stack.isEmpty()) {
            sb.append(stack.pop());
        }
        System.out.println(sb.toString());
    }
}
```

A) XYZ
B) ZYX
C) YZX
D) EmptyStackException
Correct Answer: B) ZYX
Explanation: Stack reverses the order of pushed characters.

Q:-8 Find the output of following code.

```
import java.util.Stack;
public class Test {
    public static void main(String[] args) {
        Stack<Character> stack = new Stack<>();
        String s = "123";
        for (char c : s.toCharArray()) {
            stack.push(c);
        }
        System.out.println(stack.search('1'));
    }
}
```

A) 1
B) 2
C) 3
D) -1
Correct Answer: C) 3
Explanation: Stack = [1, 2, 3]; '1' is 3rd from top.

Q:-9 Find the output of following code.

```
import java.util.Stack;
public class Test {
    public static void main(String[] args) {
        Stack<String> stack = new Stack<>();
        stack.push("Hello");
        stack.push("World");
        System.out.println(stack.size() + " " + stack.peek());
    }
}
```

A) 2 Hello
B) 2 World
C) 1 World
D) 1 Hello
Correct Answer: B) 2 World
Explanation: Two strings pushed. Size is 2, top is "World".

Q:-10 Find the output of following code.

```
import java.util.Stack;
public class Test {
```

```
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(10);
        stack.push(20);
        stack.push(30);
        System.out.println(stack.search(20));
    }
}
```

A) 1
B) 2
C) 3
D) -1
Correct Answer: B) 2
Explanation: Stack = [10, 20, 30]; 20 is 2nd from top.

Q:-11 Find the output of following code.

```
import java.util.Stack;
public class Test {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        try {
            stack.pop();
        } catch (Exception e) {
            System.out.println("Error");
        }
    }
}
```

A) 0
B) Error
C) null
D) Compilation error
Explanation: Popping from an empty stack throws an exception, caught and "Error" is printed.

Q:-12 Find the output of following code.

```
import java.util.Stack;
public class Test {
    public static void main(String[] args) {
        Stack<Character> stack = new Stack<>();
        String s = "ABC";
        for (char c : s.toCharArray()) {
            stack.push(c);
        }
        System.out.println(stack.pop());
    }
}
```

A) A
B) B
C) C
D) EmptyStackException
Answer: C) C.
Explanation: Stack = [A, B, C]; last pushed (C) is popped.
Higher priority elements will be deleted first whereas lower priority elements will be deleted next. Queue data structure always follows FIFO principle.

Q:-13 Find the output of following code.

```
import java.util.LinkedList;
import java.util.Queue;
public class Main {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        queue.offer(10);
        queue.offer(20);
        queue.offer(30);
        System.out.println(queue);
    }
}
```

A) [30, 20, 10]
B) [10, 20, 30]
C) [20, 10, 30]
D) [10, 30, 20]

Answer: B) [10, 20, 30]
Explanation: Elements are added in order and printed as a list.
Q:-14 Find the output of following code.

```
import java.util.LinkedList;
import java.util.Queue;
public class Main {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();
        queue.offer("Apple");
        queue.offer("Banana");
        System.out.println(queue.peek());
    }
}
```

A) Banana
B) Apple
C) null
D) Exception

Answer: B) Apple
Explanation: peek() returns the head of the queue, which is "Apple".

Q:-15. Find the output of following code.

```
import java.util.LinkedList;
import java.util.Queue;
public class Main {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        queue.offer(1);
        queue.offer(2);
        queue.poll();
        System.out.println(queue.peek());
    }
}
```

A) 1
B) 2
C) null
D) Exception

Answer: B) 2
Explanation: 1 is removed, so peek() returns 2

Q:-16 Find the output of following code.

```
import java.util.LinkedList;
import java.util.Queue;
public class Main {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        System.out.println(queue.poll());
    }
}
```

A) 0
B) null
C) Exception
D) Empty

Answer: B) null
Explanation: poll() returns null if the queue is empty.

Q:-17 Find the output of following code.

```
import java.util.LinkedList;
import java.util.Queue;
public class Main {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        queue.offer(5);
        queue.offer(10);
        queue.poll();
        queue.offer(15);
```

```
        System.out.println(queue.size());
    }
}
```

A) 1
B) 2
C) 3
D) 0

Answer: B) 2
Explanation: After operations, queue contains [10, 15].

Q:-18. Find the output of following code.

```
import java.util.ArrayDeque;
import java.util.Queue;
public class Main {
    public static void main(String[] args) {
        Queue<Integer> queue = new ArrayDeque<>();
        queue.offer(100);
        queue.offer(200);
        queue.offer(300);
        queue.poll();
        System.out.println(queue);
    }
}
```

A) [100, 200, 300]
B) [200, 300]
C) [300, 200]
D) [100, 300]

Answer: B) [200, 300]
Explanation: 100 is removed, the rest remain.

Q:-19 Find the output of following code.

```
import java.util.LinkedList;
import java.util.Queue;
public class Main {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        try {
            queue.remove();
        } catch (Exception e) {
            System.out.println("Caught");
        }
    }
}
```

A) null
B) Caught
C) 0

D) No output
Answer: B) Caught
Explanation: remove() throws an exception when the queue is empty, which is caught.

Q:-20 Find the output of following code.

```
import java.util.LinkedList;
import java.util.Queue;
public class Main {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();
        queue.offer("X");
        queue.offer("Y");
        queue.poll();
        queue.offer("Z");
        System.out.println(queue);
    }
}
```

A) [X, Y, Z]
B) [Y, Z]
C) [Z, Y]
D) [X, Z]
Answer: B) [Y, Z]
Explanation: "X" is removed, then "Z" added → ["Y", "Z"]

Q:-21 Find the output of following code.

```
import java.util.LinkedList;
import java.util.Queue;
public class Main {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        queue.offer(1);
        queue.offer(2);
        queue.offer(3);
        for (Integer num : queue) {
            System.out.print(num + " ");
        }
    }
}
```

A) 3 2 1
B) 1 2 3
C) 2 1 3
D) 1 3 2
Answer: B) 1 2 3
Explanation: Queue preserves insertion order.
Q:-22 Find the output of following code.

```
import java.util.LinkedList;
import java.util.Queue;
public class Main {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        queue.offer(10);
        queue.offer(20);
        queue.clear();
        System.out.println(queue.size());
    }
}
```

A) 2
B) 1
C) 0
D) Exception

Answer: C) 0
Explanation: clear() empties the queue.

Q:-23 Find the output of following code.

```
import java.util.LinkedList;
import java.util.Queue;
public class Main {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();
        queue.offer(null);
        queue.offer("Test");
        System.out.println(queue.peek());
    }
}
```

A) Test
B) null
C) Exception
D) No output

Answer: B) null
Explanation: null is allowed in LinkedList queues. peek() returns the head.

Q:-24 Find the output of following code.

```
import java.util.ArrayDeque;
import java.util.Queue;
public class Main {
    public static void main(String[] args) {
        Queue<Integer> queue = new ArrayDeque<>();
        System.out.println(queue.peek());
    }
}
```

A) 0
B) null
C) Exception
D) Empty

Answer: B) null
Explanation: peek() returns null if queue is empty, no exception thrown.

Q25:-. The prefix form of A-B/ (C * D ^ E) is?
a) -A/B*C^DE
b) -A/BC*^DE
c) -ABCD*^DE
d) -/*^ACBDE
Answer: a
Explanation: Infix Expression is A-B/(C*D^E)
This can be written as: A-(B/(C*(D^E)))
Thus prefix expression is -A/B*C^DE.