1. Similarities Between an Array and a Stack

An array and a stack are similar in that both are data structures used to store collections of data items. In both cases, the elements are stored in an organized way and can be accessed programmatically. A stack is often implemented using an array, meaning the stack relies on an underlying array to store its values. Additionally, both structures have a defined size (either fixed or managed) and use indexing internally to track stored elements.

However, while an array allows direct access to any element, a stack restricts access to maintain order — this distinction highlights their structural similarity but functional difference.

2. Use of a Stack in Compiler Software

A programmer might use a stack in compiler software to track nested structures, such as parentheses, brackets, or function calls. For example, when a compiler checks whether parentheses are properly matched, it pushes opening symbols onto a stack and pops them when a matching closing symbol is encountered. Stacks are also essential for managing function calls, where the most recent function call must be completed before returning to the previous one. This aligns naturally with the stack's LIFO behavior.

3. Stack Code Output Analysis

```
Stack s = new Stack(10);
s.push(5);
s.push(8);
int x = s.pop();   // x = 8
s.push(x);
s.push(12);
s.push(13);
int y = s.pop();   // y = 13
System.out.println(x + " " + y);

y = s.pop();       // y = 12
x = s.top();       // x = 8
System.out.println(x + " " + y);
```

Output

```
8 13
8 12
```

Explanation:
Stacks remove the most recently added item first. Each pop() removes the top value, while top() simply observes it without removal.

4. The “Hot Plate” Problem and Stack Analogy

The hot plate problem is analogous to a stack because plates are added and removed from the top only. When a fresh plate is placed on the pile, it becomes the first one used by the next customer, even if older plates are beneath it. This mirrors a stack’s Last-In, First-Out (LIFO) behavior, where the most recently added item is accessed first.

5. Queue Code Output Analysis

```
Queue q = new Queue(10);
q.enqueue(5);
q.enqueue(8);
int x = q.dequeue();  // x = 5
q.enqueue(x);
q.enqueue(12);
q.enqueue(13);
int y = q.dequeue();  // y = 8
System.out.println(x + " " + y);

y = q.dequeue();      // y = 5
x = q.front();        // x = 12
System.out.println(x + " " + y);
```

Output

```
5 8
12 5
```

Explanation:
Queues remove items in the order they were added, following FIFO behavior.

6. Difference Between FIFO and LIFO

FIFO (First-In, First-Out): The first item added is the first one removed (used by queues).

LIFO (Last-In, First-Out): The last item added is the first one removed (used by stacks).

This difference affects how data is accessed and is chosen based on the problem being solved.

7. Real-World Queue Examples (Not from the Chapter)

Customer support tickets being handled in the order they are received.

Print jobs sent to a printer, where the earliest request prints first.

8. True / False with Explanation

a) A stack data structure has a front and a rear

False. A stack has only a top.

b) A stack can be emptied

True.

c) In a stack, top refers to the first item pushed

False. The top refers to the most recent item pushed.

d) The isEmpty operation returns an int value

False. It returns a boolean.

e) A queue can hold more than one data item

True.

f) In a queue, all removals are made at the rear

False. Removals occur at the front.

g) The enqueue operation adds an item to the front

False. Enqueue adds to the rear.

h) The first item in a linked list is called the head

True.

i) A node refers to an item in a stack

False. A node is a component of a linked list.

j) The number of items can be determined with the length operation

True.

k) In a linked list, the tail points to null

True.