Queues and Stacks

Queue - FIFO (Line a costa lines)

This follows the First-In-First-Out (FIFO) approach. It is an ordered collection of elements where the addition of elements happens at one end called the "rear," and the removal of elements occurs at the other end called the "front."

The queue interface extends the collection interface, there are multiple implementations of the queue interface. The most common is the linked list implementation. LinkedList class provides the underlying implementation for the queue operations.

```
import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();

        queue.offer("Apple");
        queue.offer("Banana");
        queue.offer("Orange");

        System.out.println(queue);  // Output: [Apple, Banana, Orange]

        String firstElement = queue.poll();
        System.out.println("Removed Element: " + firstElement);  //
Output: Removed Element: Apple

        System.out.println(queue);  // Output: [Banana, Orange]

        String element = queue.peek();
        System.out.println("Element at front: " + element);  // Output:
Element at front: Banana

        boolean contains = queue.contains("Banana");
        System.out.println("Queue contains Banana? " + contains);  //
Output: Queue contains Banana? true
    }
}
```

Here are some key characteristics of queues in Java:

1. Adding elements: Elements are added to the rear of the queue using the `offer(element)` or `add(element)` methods. Both methods add the specified element to the queue.
2. Removing elements: Elements are removed from the front of the queue using the `poll()` method. It retrieves and removes the element at the front of the queue. If the queue is empty, it returns `null`.

Alternatively, you can use the `remove()` method, which works similarly but throws an exception if the queue is empty.

3. Accessing the front element: You can access the element at the front of the queue without removing it using the `peek()` method. It returns the element at the front of the queue or `null` if the queue is empty.

4. Checking if the queue is empty: The `isEmpty()` method allows you to check whether the queue is empty or not. It returns `true` if the queue is empty, and `false` otherwise.

An example of a queue is a printer, we the first page that is printed is the first page that is added to the queue. This is a FIFO (First in first out) data structure. This is an ordered collection of elements where the addition of elements happens at one end called the "rear," and the removal of elements occurs at the other end called the "front."

Stack - LIFO (Like a deck of cards)

Last in first out, this is like a deck of cards, you can only see the top card and you can only remove the top card. It is an ordered collection of elements where the addition and removal of elements happen at the same end, commonly referred to as the "top" of the stack. Java provides the `Stack` class that implements the stack data structure. This extends the `Vector` class with five operations that allow a vector to be treated as a stack.

Here are some key characteristics of stacks in Java:

1. Adding elements: Elements are added to the top of the stack using the `push()` method. It places the specified element onto the top of the stack.

2. Removing elements: Elements are removed from the top of the stack using the `pop()` method. It removes and returns the element at the top of the stack. The stack is modified as a result.

3. Accessing the top element: You can access the element at the top of the stack without removing it using the `peek()` method. It returns the element at the top of the stack without modifying the stack.

4. Checking if the stack is empty: The `isEmpty()` method allows you to check whether the stack is empty or not. It returns `true` if the stack is empty, and `false` otherwise.

5. Searching for an element: The `search()` method can be used to search for an element in the stack. It returns the 1-based position of the element from the top of the stack if it is found, or -1 if it is not present.

A use of stacks is in browser history: Web browsers often implement the back and forward navigation using a stack. Each visited webpage is pushed onto the stack, enabling users to navigate back and forth through their browsing history.

```
public class Main {

public static void main (String args[]) {
    Stack<String> deck = new Stack<String>();
```

```
    deck.push("Minas");
    deck.push("Tom");
    deck.push("Susan");


    deck.add("Minas"); // This is using the vector class, this is not the
same as push

          System.out.println(deck);
}
}
```