

## Lesson 02 Demo 14

### Implementing Stack and Queue Operations Using Deque

**Objective:** To demonstrate the implementation of both stack and queue operations using a deque (double-ended queue) in JavaScript, which showcases its versatility in supporting multiple linear data operations and broadens your understanding of versatile data structures

**Tools required:** Visual Studio Code

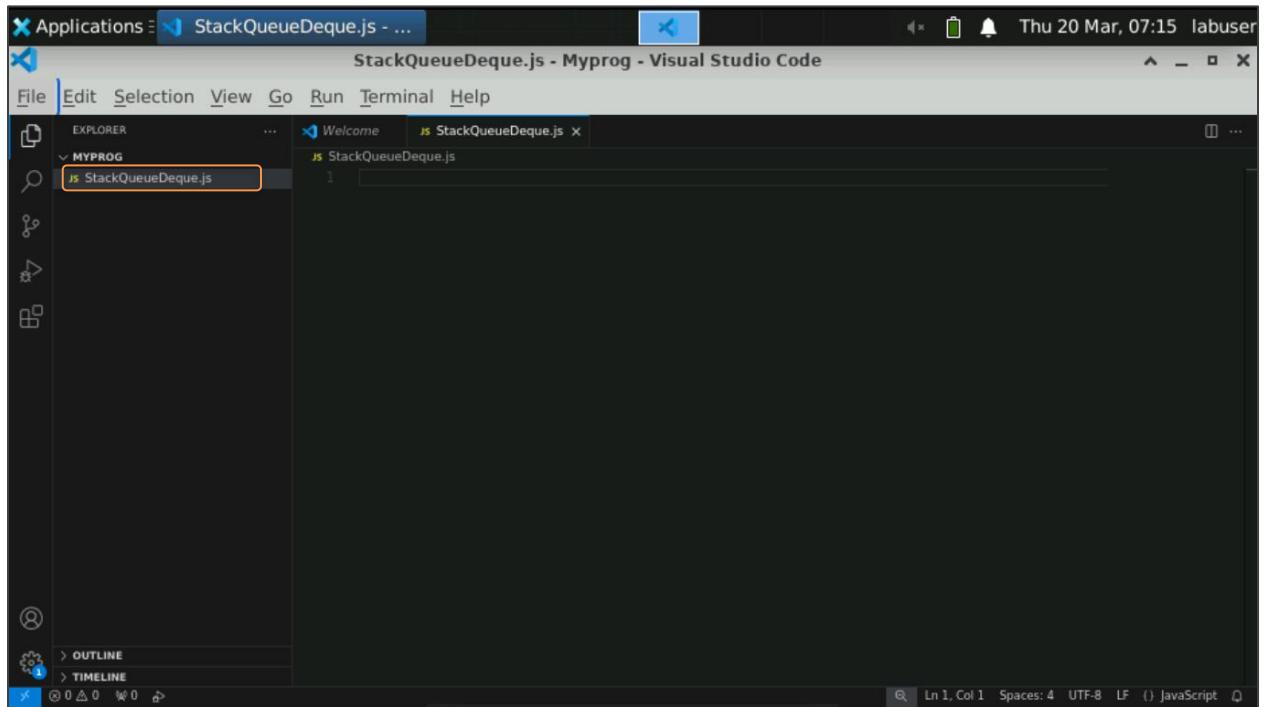
**Prerequisites:** A basic understanding of stacks, queues, deques, and JavaScript

Steps to be followed:

1. Create a JavaScript file and execute it

#### Step 1: Create a JavaScript file and execute it

1.1 Open the Visual Studio Code editor and create a JavaScript file named **StackQueueDeque.js**



1.2 Add the following code to the file:

```
// Deque implementation
class Deque {
    constructor() {
        this.items = [];
    }

    // Methods for Stack implementation
    push(item) {
        this.items.push(item);
    }

    pop() {
        if (this.isEmpty()) {
            return undefined;
        }
        return this.items.pop();
    }

    // Methods for Queue implementation
    enqueue(item) {
        this.items.push(item);
    }

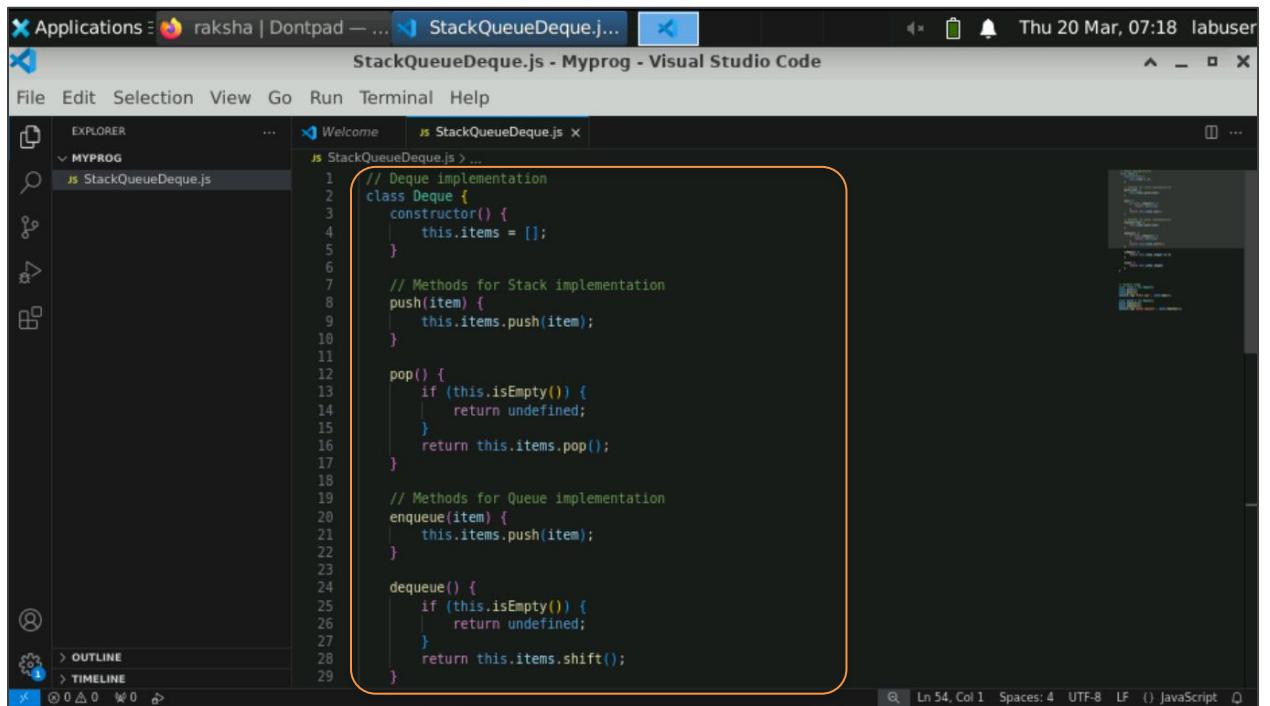
    dequeue() {
        if (this.isEmpty()) {
            return undefined;
        }
        return this.items.shift();
    }

    isEmpty() {
        return this.items.length === 0;
    }

    size() {
        return this.items.length;
    }
}
```

```
// Example usage
const stack = new Deque();
stack.push(1);
stack.push(2);
console.log('Stack pop:', stack.pop());

const queue = new Deque();
queue.enqueue(1);
queue.enqueue(2);
console.log('Queue dequeue:', queue.dequeue());
```



A screenshot of the Visual Studio Code interface. The title bar shows "Applications" and "raksha | Dontpad — StackQueueDequeue.js - Myprog - Visual Studio Code". The date "Thu 20 Mar, 07:18 labuser" is also visible. The main area displays the following JavaScript code:

```
// Deque implementation
class Deque {
  constructor() {
    this.items = [];
  }

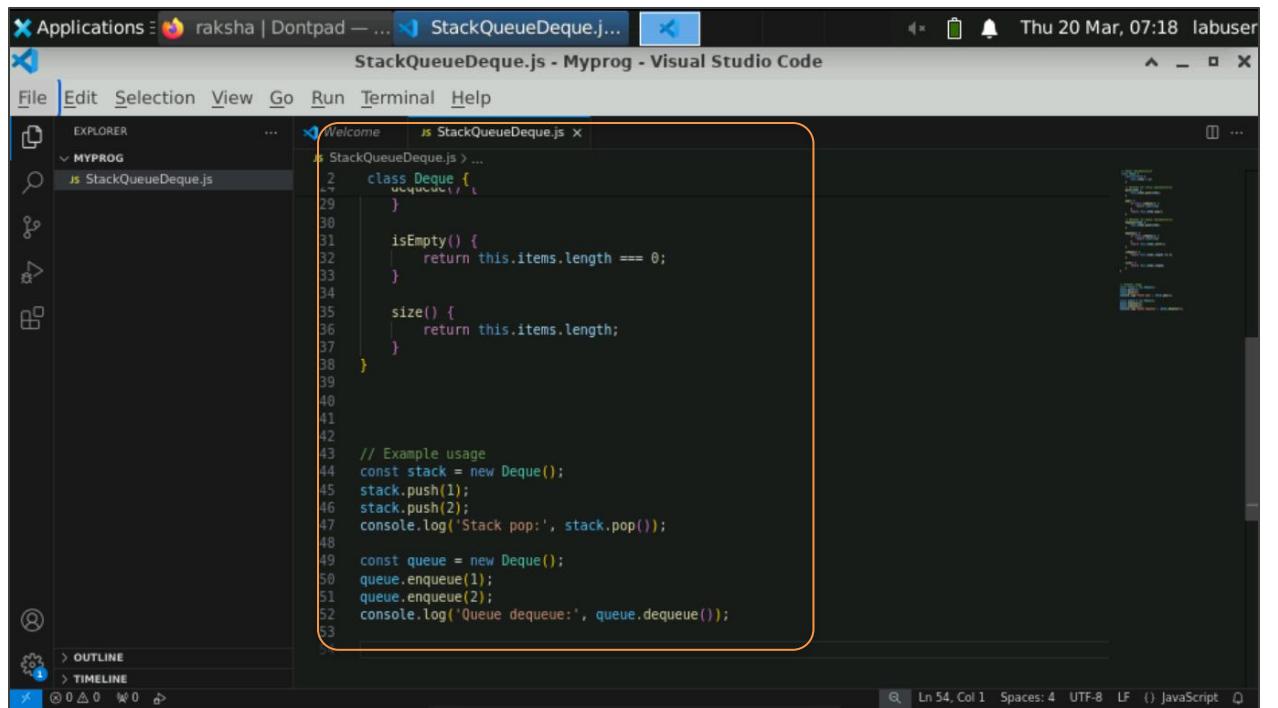
  // Methods for Stack implementation
  push(item) {
    this.items.push(item);
  }

  pop() {
    if (this.isEmpty()) {
      return undefined;
    }
    return this.items.pop();
  }

  // Methods for Queue implementation
  enqueue(item) {
    this.items.push(item);
  }

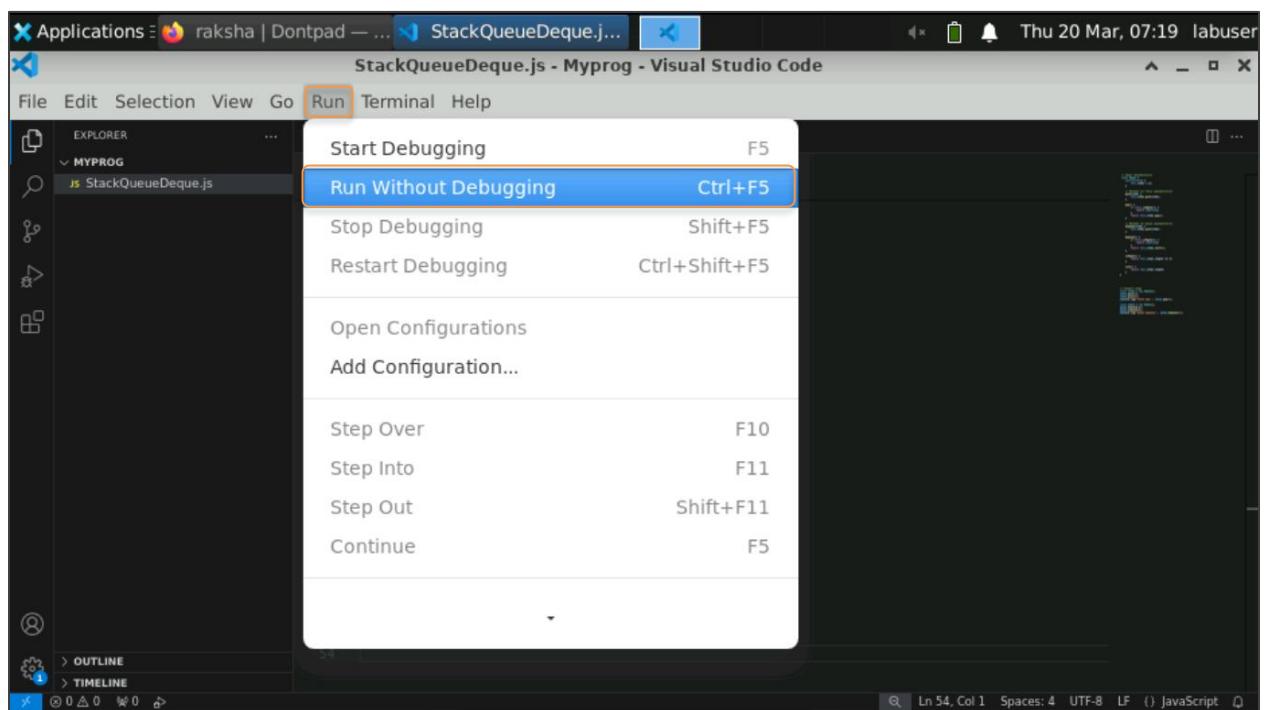
  dequeue() {
    if (this.isEmpty()) {
      return undefined;
    }
    return this.items.shift();
  }
}
```

The code is annotated with a yellow rounded rectangle highlighting the entire class definition from the first line to the last line. The status bar at the bottom right shows "Ln 54, Col 1 Spaces: 4 UTF-8 LF () JavaScript".



```
class Deque {  
    constructor() {  
        this.items = [];  
    }  
  
    isEmpty() {  
        return this.items.length === 0;  
    }  
  
    size() {  
        return this.items.length;  
    }  
}  
  
// Example usage  
const stack = new Deque();  
stack.push(1);  
stack.push(2);  
console.log('Stack pop:', stack.pop());  
  
const queue = new Deque();  
queue.enqueue(1);  
queue.enqueue(2);  
console.log('Queue dequeue:', queue.dequeue());
```

1.3 Click Run and then Run Without Debugging. Select Node.js to check the output in the DEBUG CONSOLE.



A screenshot of the Visual Studio Code interface. The title bar shows 'Applications raksha | Dontpad — StackQueueDeque.js - Myprog - Visual Studio Code'. The status bar at the bottom right indicates 'Thu 20 Mar, 07:19 labuser'. The main area displays a JavaScript file named 'StackQueueDeque.js'. A context menu is open over the code, with the 'Select debugger' option highlighted. Other options visible in the dropdown include 'Node.js', 'VS Code Extension Development', 'Web App (Chrome)', 'Web App (Edge)', and 'Install an extension for JavaScript...'. The code itself contains implementations for stack and queue operations using a deque.

```
1 // Stack
2 class Deque {
3     constructor() {
4         this.items = [];
5     }
6
7     enqueue(item) {
8         this.items.push(item);
9     }
10
11     dequeue() {
12         return this.items.shift();
13     }
14
15     isEmpty() {
16         return this.items.length === 0;
17     }
18
19     size() {
20         return this.items.length;
21     }
22 }
23
24 // Example usage
25 const stack = new Deque();
26 stack.enqueue(1);
27 stack.enqueue(2);
28 console.log('Stack pop:', stack.pop());
29
30 const queue = new Deque();
31 queue.enqueue(1);
32 queue.enqueue(2);
33 console.log('Queue dequeue:', queue.dequeue());
```

1.4 View the output in the **DEBUG CONSOLE** as shown below:

A screenshot of the Visual Studio Code interface, similar to the previous one but with the 'DEBUG CONSOLE' tab selected in the bottom navigation bar. The title bar shows 'Applications raksha | Dontpad — StackQueueDeque.js - Myprog - Visual Studio Code'. The status bar at the bottom right indicates 'Thu 20 Mar, 07:20 labuser'. The main area displays the same 'StackQueueDeque.js' file. The DEBUG CONSOLE tab is active, showing the output of running the script: '/usr/bin/node ./StackQueueDeque.js', followed by 'Stack pop: 2' and 'Queue dequeue: 1'. The code in the editor remains the same as in the previous screenshot.

```
1 // Stack
2 class Deque {
3     constructor() {
4         this.items = [];
5     }
6
7     enqueue(item) {
8         this.items.push(item);
9     }
10
11     dequeue() {
12         return this.items.shift();
13     }
14
15     isEmpty() {
16         return this.items.length === 0;
17     }
18
19     size() {
20         return this.items.length;
21     }
22 }
23
24 // Example usage
25 const stack = new Deque();
```

By following these steps, you have successfully implemented stack and queue operations using a deque in JavaScript, broadening your understanding of versatile data structures and their applications in programming.