**Assignment 2**: Data Structures

Task: Implement a stack and a queue in JavaScript. Demonstrate their use in a real-world scenario.

Below is an implementation of a stack and a queue in JavaScript, followed by a real-world scenario demonstrating their use.

Stack Implementation:

| **class Stack {**  **constructor() {**  **this.items = [];**  **}**  **push(element) {**  **this.items.push(element);**  **}**  **pop() {**  **if (this.items.length === 0) {**  **return "Underflow";**  **}**  **return this.items.pop();**  **}**  **peek() {**  **return this.items[this.items.length - 1];**  **}**  **isEmpty() {**  **return this.items.length === 0;**  **}**  **size() {**  **return this.items.length;**  **}**  **}**  **// Example usage of Stack**  **const stack = new Stack();**  **stack.push(10);**  **stack.push(20);**  **stack.push(30);**  **console.log("Stack Elements:", stack.items);**  **console.log("Pop:", stack.pop());**  **console.log("Peek:", stack.peek());**  **console.log("Is Empty:", stack.isEmpty());**  **console.log("Stack Size:", stack.size());** |
| --- |

**Queue Implementation:**

| **class Queue {**  **constructor() {**  **this.items = [];**  **}**  **enqueue(element) {**  **this.items.push(element);**  **}**  **dequeue() {**  **if (this.items.length === 0) {**  **return "Underflow";**  **}**  **return this.items.shift();**  **}**  **front() {**  **if (this.items.length === 0) {**  **return "Queue is empty";**  **}**  **return this.items[0];**  **}**  **isEmpty() {**  **return this.items.length === 0;**  **}**  **size() {**  **return this.items.length;**  **}**  **}**  **// Example usage of Queue**  **const queue = new Queue();**  **queue.enqueue(10);**  **queue.enqueue(20);**  **queue.enqueue(30);**  **console.log("Queue Elements:", queue.items);**  **console.log("Dequeue:", queue.dequeue());**  **console.log("Front:", queue.front());**  **console.log("Is Empty:", queue.isEmpty());**  **console.log("Queue Size:", queue.size());** |
| --- |

**Real-world Scenario:**

**Imagine a scenario where you are processing a list of tasks in a web application. You can use a stack to keep track of the tasks that need to be undone (like an "Undo" feature), and a queue to manage a list of tasks that need to be processed in order.**

| **// Real-world scenario using Stack and Queue**  **// Stack for managing undo tasks**  **const undoStack = new Stack();**  **function performTask(task) {**  **// Perform the task**  **console.log("Performing task:", task);**  **// Push the task onto the undo stack**  **undoStack.push(task);**  **}**  **function undoLastTask() {**  **if (!undoStack.isEmpty()) {**  **const undoneTask = undoStack.pop();**  **console.log("Undoing task:", undoneTask);**  **} else {**  **console.log("Nothing to undo");**  **}**  **}**  **// Queue for managing tasks to be processed**  **const taskQueue = new Queue();**  **function addTaskToQueue(task) {**  **taskQueue.enqueue(task);**  **console.log("Task added to the queue:", task);**  **}**  **function processTasks() {**  **while (!taskQueue.isEmpty()) {**  **const task = taskQueue.dequeue();**  **console.log("Processing task:", task);**  **}**  **if (taskQueue.isEmpty()) {**  **console.log("Task queue is empty");**  **}**  **}**  **// Example usage**  **performTask("Task 1");**  **performTask("Task 2");**  **addTaskToQueue("Task 3");**  **addTaskToQueue("Task 4");**  **undoLastTask();**  **processTasks();** |
| --- |

**In this scenario, the undoStack is used as a stack to manage the undo functionality, and the taskQueue is used as a queue to manage a list of tasks to be processed. The example usage demonstrates performing tasks, adding tasks to the queue, undoing the last task, and processing tasks from the queue.**