Figure 1: alt text

Universidad Tecnológica de Metropolitana

Estructura de datos | Parcial 2

Profesor: Ruth Betsaida Martinez Dominguez

Alumno: Abril Contreras Suaste

Tarea: Actividades del 8 al 12

31 de Octubre de 2024

Actividad 8

1. El usuario ingresara los datos para la suma.
2. Crea pilas con los digitos. Los numeros se dividen en dígitos individuales, que se insertan en dos pilas (stack1 y stack2).

```
for (let i = 0; i < num1.length; i++) {
    stack1.push(parseInt(num1[i]));
    visualizeStack('stack1', num1[i]);
}
```

3. Suma los dígitos de los digitos. Se realiza la suma de los números dígito por dígito. Durante el proceso, se gestiona el acarreo y se almacena en carry.

```
const sum = digit1 + digit2 + carry;
resultStack.push(sum % 10);
carry = Math.floor(sum / 10);
```

4. Si hay acarreo, se agrega a la pila de resultados. Cada vez que se genera un acarreo, este se visualiza en la pila de acarreo (carry-stack).

```
visualizeStack('carry-stack', carry > 0 ? carry : '');
```

Actividad 9

1. Creación de pila Se creo la clase pila para gestionar los elementos de la pila (listas). La clase contiene

- push(): Agrega valores a la pila
- replace(): Reemplaza un valor en la pila
- getElements(): Devuelve los elementos de la pila

2. Generación de numeros aleatorios

```
for (let i = 1; i <= 10; i++) {
   pilaNumeros.push(Math.floor(Math.random() * 100));
}
```

3. Mostrar pila La función mostrarPila() muestra los elementos de la pila actualizados en la interfaz HTML. Usa un ciclo forEach para iterar sobre los elementos de la pila y mostrarlos.

```
function mostrarPila() {
    const stackDisplay = document.getElementById("stackDisplay");
    stackDisplay.innerHTML = "";  // Limpiar la vista anterior
    const elementos = pilaNumeros.getElements();
    elementos.forEach(element => {
        const div = document.createElement("div");
        div.className = "element";
        div.textContent = element;
        stackDisplay.appendChild(div);
    });
}
```

Actividad 10

Este código simula una fila de espera para atender clientes en un banco. Los clientes se encolan con sus datos personales y tipo de movimiento, y se calcula su tiempo de espera hasta que son atendidos. 1. Clase Cliente La clase Cliente representa a cada cliente en la fila. Tiene las siguientes propiedades:

- turno: el número de turno del cliente en la fila.
- nombre: nombre del cliente.
- movimiento: tipo de movimiento bancario (por ejemplo, depósito, retiro).
- horaLlegada: momento en que el cliente se encoló. También incluye el método tiempoEspera(), que calcula y devuelve el tiempo en segundos que el cliente ha esperado desde su llegada hasta el momento actual.

2. Clase Cola La clase Cola maneja la cola de clientes, utilizando un array (this.cola) para almacenar a los clientes en orden. turnoActual es un contador que asigna un número de turno único a cada cliente, y limite define la capacidad máxima de la cola. También se utiliza interval para actualizar el tiempo de espera de cada cliente cada segundo.

El método agregarCliente(nombre, movimiento) añade un cliente a la cola, comprobando que no se exceda el límite de capacidad. Si se puede agregar, se crea un nuevo objeto Cliente y se añade a la cola. Luego, se actualiza la vista y se inicia la actualización del tiempo si aún no ha comenzado.

```
class Cola {
    constructor() {
        this.cola = [];
        this.turnoActual = 1;
        this.limite = 10;
        this.interval = null;
    }

    agregarCliente(nombre, movimiento) {
        if (this.cola.length >= this.limite) {
            document.getElementById("mensaje").innerText = "La cola está llena. No se pueder
            return;
        }

        const cliente = new Cliente(this.turnoActual++, nombre, movimiento);
        this.cola.push(cliente);
        this.actualizarVista();
        this.iniciarActualizacionTiempo();
    }
```

3. Método atenderCliente atenderCliente() retira el primer cliente de la cola (FIFO) y calcula su tiempo de espera. Este tiempo se muestra junto con el nombre del cliente que ha sido atendido. Si la cola queda vacía, se detiene

el intervalo de actualización de tiempo.

```
atenderCliente() {
    if (this.cola.length === 0) {
        document.getElementById("mensaje").innerText = "No hay nadie en la fila para ate
        return;
    }

    const clienteAtendido = this.cola.shift();
    const tiempoEspera = clienteAtendido.tiempoEspera();
    document.getElementById("mensaje").innerText = `El cliente ${clienteAtendido.nombre

    if (this.cola.length === 0) {
        clearInterval(this.interval);
    }

    this.actualizarVista();
}
```

4. Métodos de actualización de tiempo iniciarActualizacionTiempo() establece un intervalo de actualización cada segundo para actualizar la vista de la cola. Este método evita crear múltiples intervalos si ya existe uno activo.

```
iniciarActualizacionTiempo() {
    if (this.interval) return;
    this.interval = setInterval(() => {
        this.actualizarVista();
    }, 1000);
}
```

5. Interactuar con el usuario Estos eventos controlan los botones de agregar y atender clientes. Cuando el usuario ingresa un nombre y movimiento y presiona “Agregar Cliente”, se crea un nuevo cliente en la cola. Si los campos están vacíos, muestra un mensaje solicitando la información completa. Al presionar “Atender Cliente”, el primer cliente en la fila es atendido, y el tiempo de espera se muestra al usuario.

```
const bancoCola = new Cola();

document.getElementById("agregarCliente").addEventListener("click", () => {
    const nombre = document.getElementById("nombre").value;
    const movimiento = document.getElementById("movimiento").value;

    if (!nombre || !movimiento) {
        document.getElementById("mensaje").innerText = "Por favor, complete todos los campos
        return;
    }
```

```
    bancoCola.agregarCliente(nombre, movimiento);
    document.getElementById("mensaje").innerText = `Cliente ${nombre} añadido a la cola.`;
});

document.getElementById("atenderCliente").addEventListener("click", () => {
    bancoCola.atenderCliente();
});

class Cliente {
    constructor(turno, nombre, movimiento) {
        this.turno = turno;
        this.nombre = nombre;
        this.movimiento = movimiento;
        this.horaLlegada = new Date();
    }

    tiempoEspera() {
        const ahora = new Date();
        const diferencia = ahora - this.horaLlegada;
        const segundosEspera = Math.floor(diferencia / 1000);
        return segundosEspera;
    }
}
```

Actividad 11

1. Clase Cola Este código implementa una simulación en la que los coches llegan a una cola y esperan a ser pintados. La clase Cola maneja la estructura de datos de la cola de espera, y el resto del código gestiona la interfaz del juego. A continuación, se describen las secciones clave del código:

```
class Cola {
    constructor() {
        this.items = [];
    }

    enqueue(element) {
        this.items.push(element);
    }

    dequeue() {
        if (this.isEmpty()) return "La cola está vacía";
        return this.items.shift();
    }

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

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

    clear() {
        this.items = [];
    }
}
```

La clase Cola se utiliza para manejar la fila de coches esperando a ser pintados. Implementa una estructura FIFO (First In, First Out), asegurando que los coches que llegan primero sean pintados en el mismo orden.

2. Funcion addCarToQueue addCarToQueue simula la llegada de un coche. Si la cola está llena, muestra un mensaje de advertencia. De lo contrario, genera un coche con un color aleatorio y un tiempo de llegada, añadiéndolo a carQueue y actualizando la vista.

```
function addCarToQueue() {
    if (carQueue.size() >= maxCarsInQueue) {
        document.getElementById("gameMessage").innerText = "¡La cola está llena!";
        return;
```

```
    }

    let colors = ['yellow', 'orange', 'red', 'green', 'blue', 'purple'];
    let randomColor = colors[Math.floor(Math.random() * colors.length)];

    let newCar = {
        id: carQueue.size() + 1,
        color: randomColor,
        arrivalTime: new Date().getTime()
    };

    carQueue.enqueue(newCar);
    updateCarQueueView();
}
```

3. Funcion paintCar La función paintCar permite pintar el coche en la parte frontal de la cola (FIFO). Primero verifica que haya un coche en la cola y que el usuario haya seleccionado un color. Luego, compara el color seleccionado con el color del coche. Si coinciden, incrementa el contador de coches pintados y reduce el intervalo de llegada de coches cada tres pintados, aumentando así la dificultad del juego.

```
function paintCar() {
    if (carQueue.isEmpty()) {
        document.getElementById("gameMessage").innerText = "¡No hay coches en la cola!";
        return;
    }

    let firstCar = carQueue.dequeue();
    let colorButtons = document.querySelectorAll(".color-btn");
    let selectedColor = Array.from(colorButtons).find(btn => btn.classList.contains("selecte

    if (!selectedColor) {
        document.getElementById("gameMessage").innerText = "Selecciona un color antes de pin
        return;
    }

    let chosenColor = selectedColor.dataset.color;

    if (chosenColor !== firstCar.color) {
        document.getElementById("gameMessage").innerText = `¡Error! El coche necesita ser pi
    } else {
        let currentTime = new Date().getTime();
        let waitTime = Math.floor((currentTime - firstCar.arrivalTime) / 1000);
        document.getElementById("gameMessage").innerText = `¡Coche pintado de ${firstCar.col

        totalPaintedCars++;
```

```
            totalTime += waitTime;
            updateGameInfo();

            if (totalPaintedCars % 3 === 0 && carInterval > 2000) {
                carInterval -= 2000;
                clearInterval(carQueueInterval);
                startCarQueue();
            }
        }

        updateCarQueueView();
    }
```

Actividad 12

1. NodoAuto NodoAuto representa un nodo de la lista doblemente enlazada circular. Cada nodo almacena los datos de un auto (placa, propietario, hora de entrada) y enlaces a los nodos siguiente y anterior para facilitar el movimiento bidireccional en la lista.

```
class NodoAuto {
    constructor(placa, propietario, horaEntrada) {
        this.placa = placa;
        this.propietario = propietario;
        this.horaEntrada = horaEntrada;
        this.siguiente = null;
        this.anterior = null;
    }
}
```

2. Clase Estacionamiento

La clase Estacionamiento gestiona los autos estacionados mediante una lista doblemente enlazada circular. La propiedad cabeza apunta al primer auto en la lista, mientras que cola apunta al último auto, y totalAutos lleva la cuenta de la cantidad de autos en el estacionamiento.

```
class Estacionamiento {
    constructor() {
        this.cabeza = null;
        this.cola = null;
        this.totalAutos = 0;
    }

    agregarAuto(placa, propietario) { ... }

    sacarAuto(placa) { ... }

    mostrarAutos() { ... }
}
```

agregarAuto permite registrar un nuevo auto en el estacionamiento. Si el estacionamiento está vacío, el auto se convierte en el primero de la lista. Si ya hay autos, se añade al final, ajustando los punteros para mantener la estructura circular y doblemente enlazada.

SacarAuto permite retirar un auto del estacionamiento. Primero busca el auto por su placa. Si lo encuentra, calcula el tiempo estacionado y el costo total (2.00 unidades monetarias por segundo). Luego, ajusta los punteros de los nodos anterior y siguiente para mantener la estructura circular, eliminando el auto de la lista.

mostrarAutos devuelve un array con todos los autos en el estacionamiento, recorriendo la lista desde cabeza hasta cabeza nuevamente. Esto permite mostrar

en pantalla los autos actualmente estacionados.

3. Funciones de retiro de auto Este evento despliega una lista de autos disponibles para salir. Cada auto tiene un botón que, al presionarse, llama a sacarAuto en la clase Estacionamiento para retirarlo. Si el auto se retira con éxito, muestra un mensaje con el costo y tiempo estacionado.