TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE:
Agosto - Diciembre 2025

CARRERA:
Ingeniería Informática

MATERIA:
Diseño y evaluación de interfaces de usuario

TÍTULO ACTIVIDAD:
Temas Avanzados de desarrollo de software

Actividad

Examen unidad 2

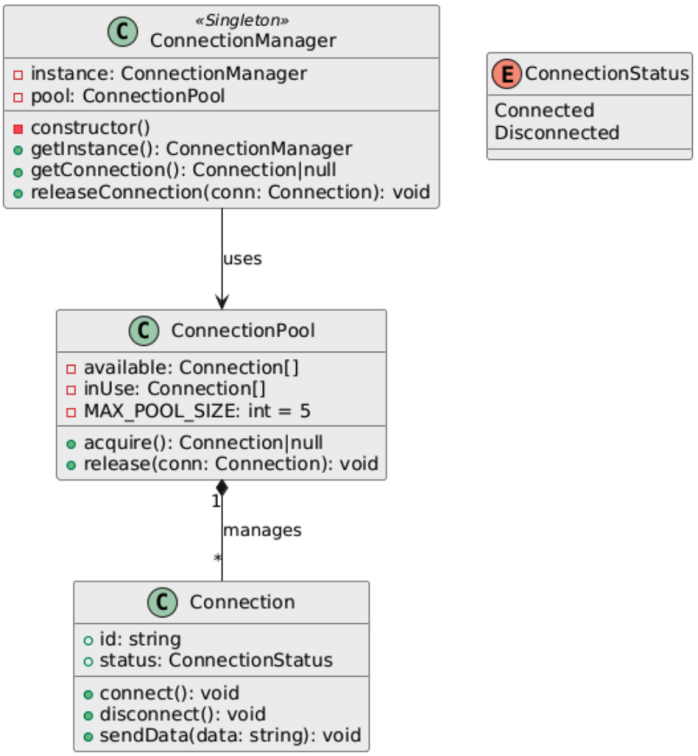
UNIDAD A EVALUAR:
2

NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:

Omar Sanchez Lozada 21212365

NOMBRE DEL MAESTRO (A):
Maribel Guerrero Luis

Diagrama UML



Código

connection.ts

src > connection.ts > Connection

```ts
export enum ConnectionStatus {
    Connected = 'connected',
    Disconnected = 'disconnected',
}

export class Connection {
    public id: string;
    public status: ConnectionStatus = ConnectionStatus.Disconnected;

    constructor() {
        this.id = `conn_${Math.random().toString(36).substring(2, 9)}`;
        this.connect();
    }

    public connect(): void {
        this.status = ConnectionStatus.Connected;
        console.log(`🔌 Conexión ${this.id} establecida.`);
    }

    public disconnect(): void {
        this.status = ConnectionStatus.Disconnected;
        console.log(`🔌 Conexión ${this.id} cerrada.`);
    }

    public sendData(data: string): void {
        if (this.status === ConnectionStatus.Connected) {
            console.log(`📡 Enviando datos vía ${this.id}: "${data}"`);
        } else {
            console.error(`❌ No se pueden enviar datos. La conexión ${this.id} no está activa.`);
        }
    }
}
```

connection-pool.ts

src > connection-pool.ts > ConnectionPool > acquire

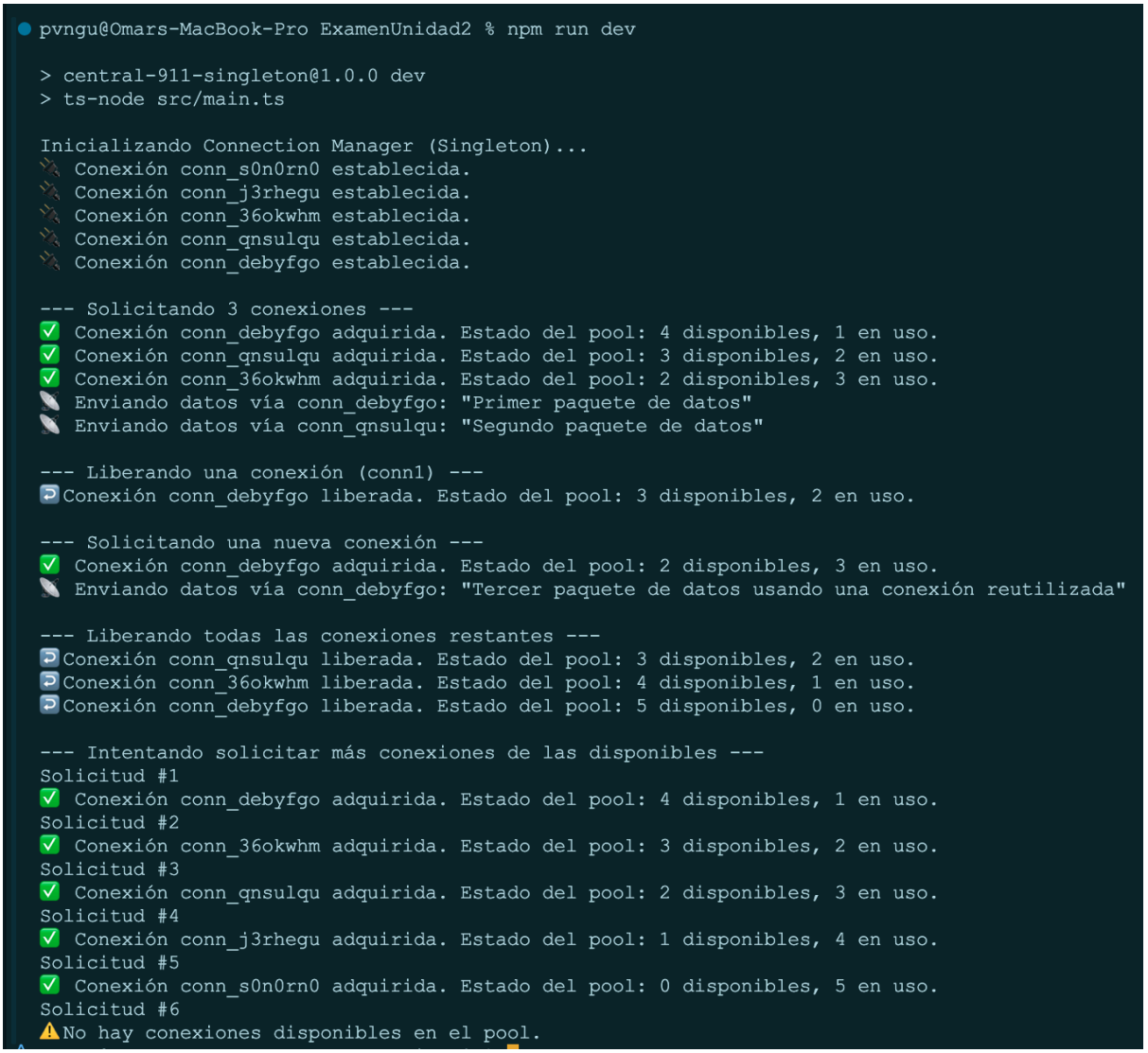
```ts
import { Connection } from './connection';

export class ConnectionPool {
    private available: Connection[] = [];
    private inUse: Connection[] = [];
    private readonly MAX_POOL_SIZE = 5;

    constructor() {
        for (let i = 0; i < this.MAX_POOL_SIZE; i++) {
            this.available.push(new Connection());
        }
    }

    public acquire(): Connection | null {
        if (this.available.length > 0) {
            const conn = this.available.pop()!;
            this.inUse.push(conn);
            console.log(`✅ Conexión ${conn.id} adquirida. Estado del pool: ${this.available.length} disponibles, ${this.inUse.length} en uso.`);
            return conn;
        } else {
            console.warn("⚠️ No hay conexiones disponibles en el pool.");
            return null;
        }
    }

    public release(conn: Connection): void {
        const index = this.inUse.indexOf(conn);
        if (index !== -1) {
            this.inUse.splice(index, 1);
            this.available.push(conn);
            console.log(`↩️ Conexión ${conn.id} liberada. Estado del pool: ${this.available.length} disponibles, ${this.inUse.length} en uso.`);
        } else {
            console.error(`🔥 Error: Intentando liberar una conexión (${conn.id}) que no está en uso.`);
        }
    }
}
```


connection-manager.ts
src > connection-manager.ts > ConnectionManager > releaseConnection
import { Connection } from './connection';
import { ConnectionPool } from './connection-pool';
export class ConnectionManager {
private static instance: ConnectionManager;
private pool: ConnectionPool;
private constructor() {
console.log("Inicializando Connection Manager (Singleton)...");
this.pool = new ConnectionPool();
}
public static getInstance(): ConnectionManager {
if (!ConnectionManager.instance) {
ConnectionManager.instance = new ConnectionManager();
}
return ConnectionManager.instance;
}
public getConnection(): Connection | null {
return this.pool.acquire();
}
public releaseConnection(conn: Connection): void {
this.pool.release(conn);
}
}

main.ts

src > main.ts > ...

```ts
import { ConnectionManager } from './connection-manager';
import { Connection } from './connection';

const manager = ConnectionManager.getInstance();
const manager2 = ConnectionManager.getInstance();

console.log("\n--- Solicitando 3 conexiones ---");
const conn1 = manager.getConnection();
const conn2 = manager.getConnection();
const conn3 = manager.getConnection();

if (conn1) {
    conn1.sendData("Primer paquete de datos");
}
if (conn2) {
    conn2.sendData("Segundo paquete de datos");
}

console.log("\n--- Liberando una conexión (conn1) ---");
if (conn1) {
    manager2.releaseConnection(conn1);
}

console.log("\n--- Solicitando una nueva conexión ---");
const conn4 = manager.getConnection();
if (conn4) {
    conn4.sendData("Tercer paquete de datos usando una conexión reutilizada");
}

console.log("\n--- Liberando todas las conexiones restantes ---");
if (conn2) manager.releaseConnection(conn2);
if (conn3) manager.releaseConnection(conn3);
if (conn4) manager.releaseConnection(conn4);

console.log("\n--- Intentando solicitar más conexiones de las disponibles ---");
const allConnections: (Connection | null)[] = [];
for (let i = 0; i < 6; i++) {
    console.log(`Solicitud #${i + 1}`);
    allConnections.push(manager.getConnection());
}
```



Conclusiones

En conclusión, implementar los patrones Singleton y Object Pool en este proyecto permitió gestionar de manera eficiente y centralizada las conexiones, evitando duplicidad y optimizando el uso de recursos. Gracias a estas buenas prácticas, el sistema mantiene un control claro sobre la creación, reutilización y liberación de conexiones, mejorando el rendimiento y la organización del código.