

# **TECNOLOGICO 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

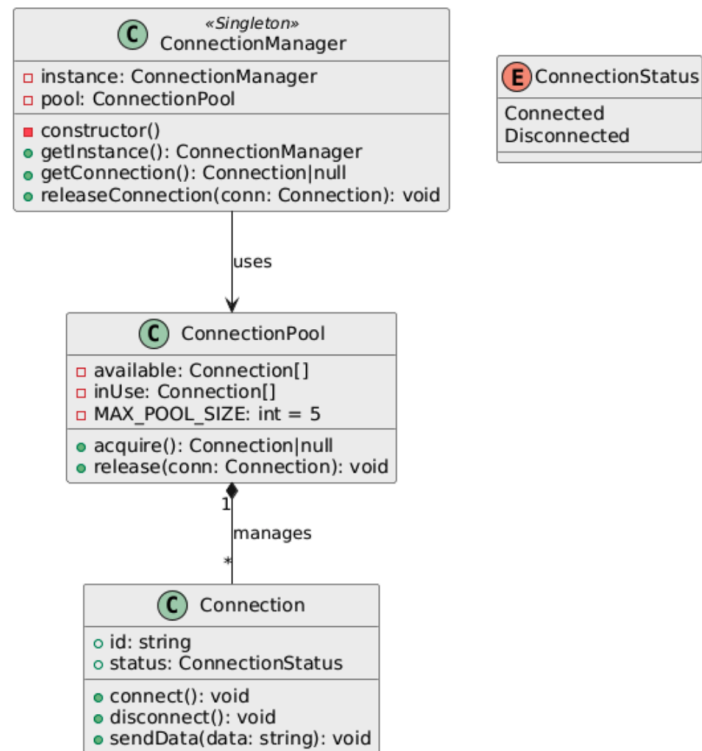
### **NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:**

Omar Sanchez Lozada 21212365

### **NOMBRE DEL MAESTRO (A):**

Maribel Guerrero Luis

## Diagrama UML





## Código

```
TS connection.ts X
src > TS connection.ts > Connection
1   export enum ConnectionStatus {
2       Connected = 'connected',
3       Disconnected = 'disconnected',
4   }
5
6   export class Connection {
7       public id: string;
8       public status: ConnectionStatus = ConnectionStatus.Disconnected;
9
10      constructor() {
11          this.id = `conn_${Math.random().toString(36).substring(2, 9)}`;
12          this.connect();
13      }
14
15      public connect(): void {
16          this.status = ConnectionStatus.Connected;
17          console.log(`🟢 Conexión ${this.id} establecida.`);
18      }
19
20      public disconnect(): void {
21          this.status = ConnectionStatus.Disconnected;
22          console.log(`🔴 Conexión ${this.id} cerrada.`);
23      }
24
25      public sendData(data: string): void {
26          if (this.status === ConnectionStatus.Connected) {
27              console.log(`📤 Enviando datos vía ${this.id}: "${data}"`);
28          } else {
29              console.error(`❌ No se pueden enviar datos. La conexión ${this.id} no está activa.`);
30          }
31      }
32  }
```

```
TS connection-pool.ts X
src > TS connection-pool.ts > ConnectionPool > acquire
1   import { Connection } from './connection';
2
3   export class ConnectionPool {
4       private available: Connection[] = [];
5       private inUse: Connection[] = [];
6       private readonly MAX_POOL_SIZE = 5;
7
8       constructor() {
9           for (let i = 0; i < this.MAX_POOL_SIZE; i++) {
10               this.available.push(new Connection());
11           }
12      }
13
14      public acquire(): Connection | null {
15          if (this.available.length > 0) {
16              const conn = this.available.pop()!;
17              this.inUse.push(conn);
18              console.log(`✅ Conexión ${conn.id} adquirida. Estado del pool: ${this.available.length} disponibles, ${this.inUse.length} en uso.`);
19              return conn;
20          } else {
21              console.warn("⚠️ No hay conexiones disponibles en el pool.");
22              return null;
23          }
24      }
25
26      public release(conn: Connection): void {
27          const index = this.inUse.indexOf(conn);
28          if (index !== -1) {
29              this.inUse.splice(index, 1);
30              this.available.push(conn);
31              console.log(`📥 Conexión ${conn.id} liberada. Estado del pool: ${this.available.length} disponibles, ${this.inUse.length} en uso.`);
32          } else {
33              console.error(`🔥 Error: Intentando liberar una conexión (${conn.id}) que no está en uso.`);
34          }
35      }
36  }
```

TS connection-manager.ts ×

src > TS connection-manager.ts >  ConnectionManager >  releaseConnection

```
1  import { Connection } from './connection';
2  import { ConnectionPool } from './connection-pool';
3
4  export class ConnectionManager {
5      private static instance: ConnectionManager;
6      private pool: ConnectionPool;
7
8      private constructor() {
9          console.log("Inicializando Connection Manager (Singleton)...");
10         this.pool = new ConnectionPool();
11     }
12
13     public static getInstance(): ConnectionManager {
14         if (!ConnectionManager.instance) {
15             ConnectionManager.instance = new ConnectionManager();
16         }
17         return ConnectionManager.instance;
18     }
19
20     public getConnection(): Connection | null {
21         return this.pool.acquire();
22     }
23
24     public releaseConnection(conn: Connection): void {
25         this.pool.release(conn);
26     }
27 }
```

main.ts

src > main.ts > ...

```
1  import { ConnectionManager } from './connection-manager';
2  import { Connection } from './connection';
3
4  const manager = ConnectionManager.getInstance();
5  const manager2 = ConnectionManager.getInstance();
6
7  console.log("\n--- Solicitando 3 conexiones ---");
8  const conn1 = manager.getConnection();
9  const conn2 = manager.getConnection();
10 const conn3 = manager.getConnection();
11
12 if (conn1) {
13     conn1.sendData("Primer paquete de datos");
14 }
15 if (conn2) {
16     conn2.sendData("Segundo paquete de datos");
17 }
18
19 console.log("\n--- Liberando una conexión (conn1) ---");
20 if (conn1) {
21     manager2.releaseConnection(conn1);
22 }
23
24 console.log("\n--- Solicitando una nueva conexión ---");
25 const conn4 = manager.getConnection();
26 if (conn4) {
27     conn4.sendData("Tercer paquete de datos usando una conexión reutilizada");
28 }
29
30 console.log("\n--- Liberando todas las conexiones restantes ---");
31 if (conn2) manager.releaseConnection(conn2);
32 if (conn3) manager.releaseConnection(conn3);
33 if (conn4) manager.releaseConnection(conn4);
34
35 console.log("\n--- Intentando solicitar más conexiones de las disponibles ---");
36 const allConnections: (Connection | null)[] = [];
37 for (let i = 0; i < 6; i++) {
38     console.log(`Solicitud #${i + 1}`);
39     allConnections.push(manager.getConnection());
40 }
```

```

● pvngu@Omars-MacBook-Pro ExamenUnidad2 % npm run dev

> central-911-singleton@1.0.0 dev
> ts-node src/main.ts

Inicializando Connection Manager (Singleton)...
✂ Conexión conn_s0n0rn0 establecida.
✂ Conexión conn_j3rhegu establecida.
✂ Conexión conn_36okwhm establecida.
✂ Conexión conn_qnsulqu establecida.
✂ Conexión conn_debyfgo establecida.

--- Solicitando 3 conexiones ---
✔ Conexión conn_debyfgo adquirida. Estado del pool: 4 disponibles, 1 en uso.
✔ Conexión conn_qnsulqu adquirida. Estado del pool: 3 disponibles, 2 en uso.
✔ Conexión conn_36okwhm adquirida. Estado del pool: 2 disponibles, 3 en uso.
🐼 Enviando datos vía conn_debyfgo: "Primer paquete de datos"
🐼 Enviando datos vía conn_qnsulqu: "Segundo paquete de datos"

--- Liberando una conexión (conn1) ---
🐼 Conexión conn_debyfgo liberada. Estado del pool: 3 disponibles, 2 en uso.

--- Solicitando una nueva conexión ---
✔ Conexión conn_debyfgo adquirida. Estado del pool: 2 disponibles, 3 en uso.
🐼 Enviando datos vía conn_debyfgo: "Tercer paquete de datos usando una conexión reutilizada"

--- Liberando todas las conexiones restantes ---
🐼 Conexión conn_qnsulqu liberada. Estado del pool: 3 disponibles, 2 en uso.
🐼 Conexión conn_36okwhm liberada. Estado del pool: 4 disponibles, 1 en uso.
🐼 Conexión conn_debyfgo liberada. Estado del pool: 5 disponibles, 0 en uso.

--- Intentando solicitar más conexiones de las disponibles ---
Solicitud #1
✔ Conexión conn_debyfgo adquirida. Estado del pool: 4 disponibles, 1 en uso.
Solicitud #2
✔ Conexión conn_36okwhm adquirida. Estado del pool: 3 disponibles, 2 en uso.
Solicitud #3
✔ Conexión conn_qnsulqu adquirida. Estado del pool: 2 disponibles, 3 en uso.
Solicitud #4
✔ Conexión conn_j3rhegu adquirida. Estado del pool: 1 disponibles, 4 en uso.
Solicitud #5
✔ Conexión conn_s0n0rn0 adquirida. Estado del pool: 0 disponibles, 5 en uso.
Solicitud #6
⚠ No hay conexiones disponibles en el pool.

```

## 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.