TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE:
Agosto - Diciembre 2025

CARRERA:
Ingeniería Informática

MATERIA:
Patrones de diseño de software

TÍTULO ACTIVIDAD:
Examen unidad 4 y 5

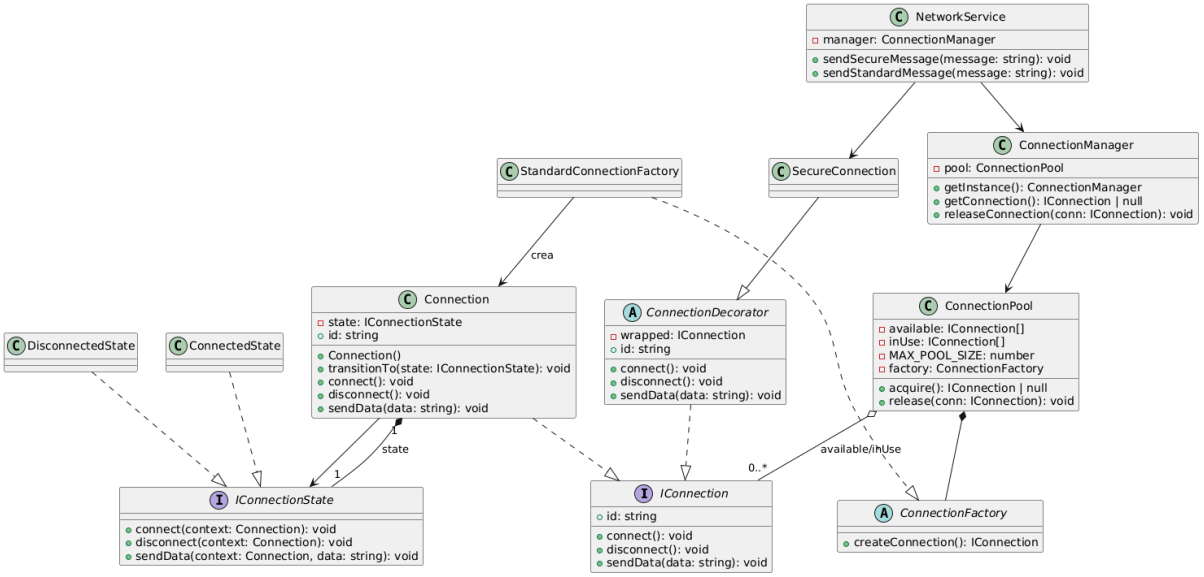
UNIDAD A EVALUAR:
4 y 5

NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:

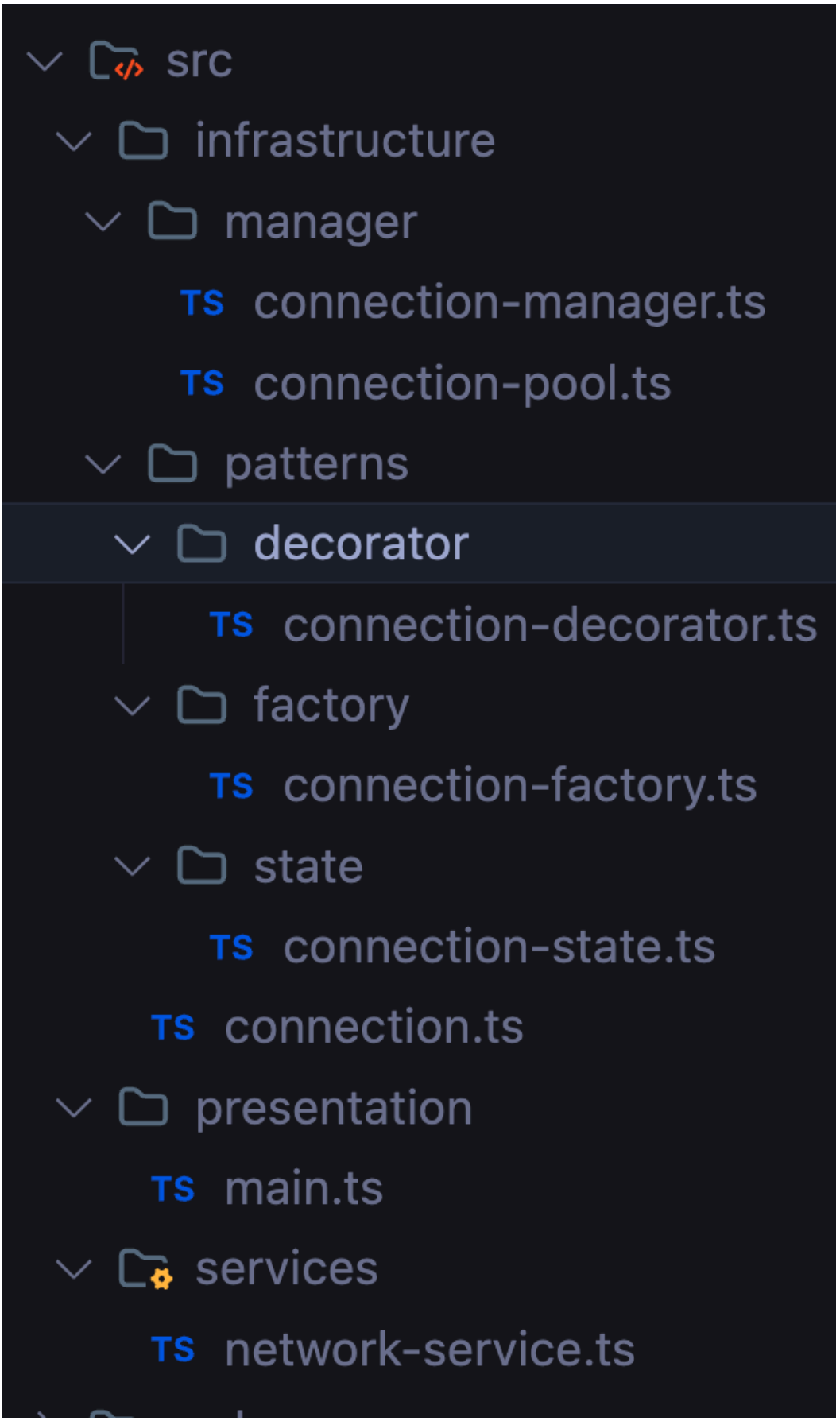
Omar Sanchez Lozada 21212365

NOMBRE DEL MAESTRO (A):
Maribel Guerrero Luis

Diagrama UML:



File Explorer:



Código:

connection-manager.ts

src > infrastructure > manager > connection-manager.ts > ConnectionManager

```ts
import { IConnection } 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(): IConnection | null {
        return this.pool.acquire();
    }

    public releaseConnection(conn: IConnection): void {
        this.pool.release(conn);
    }
}
```

connection-pool.ts

src > infrastructure > manager > connection-pool.ts > ConnectionPool > acquire

```typescript
import { IConnection } from '../connection';
import { ConnectionFactory, StandardConnectionFactory } from '../patterns/factory/connection-factory'

export class ConnectionPool {
    private available: IConnection[] = [];
    private inUse: IConnection[] = [];
    private readonly MAX_POOL_SIZE = 5;
    private factory: ConnectionFactory;

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

    public acquire(): IConnection | null {
        if (this.available.length > 0) {
            const conn = this.available.pop()!;
            this.inUse.push(conn);
            return conn;
        }
        return null;
    }

    public release(conn: IConnection): 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.`);
        }
    }
}
```

connection-decorator.ts

src > infrastructure > patterns > decorator > connection-decorator.ts > ConnectionDecorator > connect

```typescript
import { IConnection } from "../../connection";

export abstract class ConnectionDecorator implements IConnection {
    protected wrapped: IConnection;

    constructor(connection: IConnection) {
        this.wrapped = connection;
    }

    public get id(): string {
        return this.wrapped.id;
    }

    public connect(): void {
        this.wrapped.connect();
    }

    public disconnect(): void {
        this.wrapped.disconnect();
    }

    public sendData(data: string): void {
        this.wrapped.sendData(data);
    }
}

export class SecureConnection extends ConnectionDecorator {
    public sendData(data: string): void {
        const encryptedData = `[ENCRIPTADO] ${data.split('').reverse().join('')}`;
        console.log(`🔒 Encriptando datos para ${this.id}...`);
        super.sendData(encryptedData);
    }
}
```

connection-factory.ts

src > infrastructure > patterns > factory > connection-factory.ts > StandardConnectionFactory > create

```typescript
import { Connection, IConnection } from "../../connection";

export abstract class ConnectionFactory {
    public abstract createConnection(): IConnection;
}

export class StandardConnectionFactory extends ConnectionFactory {
    public createConnection(): IConnection {
        return new Connection();
    }
}
```

connection-state.ts

src > infrastructure > patterns > state > connection-state.ts > ConnectedState > disconnect

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

export interface IConnectionState {
    connect(context: Connection): void;
    disconnect(context: Connection): void;
    sendData(context: Connection, data: string): void;
}

export class ConnectedState implements IConnectionState {
    public connect(context: Connection): void {
        console.log(`⚠️ La conexión ${context.id} ya está establecida.`);
    }

    public disconnect(context: Connection): void {
        console.log(`🔌 Conexión ${context.id} cerrada.`);
        context.transitionTo(new DisconnectedState());
    }

    public sendData(context: Connection, data: string): void {
        console.log(`📡 [${context.id}] Enviando: "${data}"`);
    }
}

export class DisconnectedState implements IConnectionState {
    public connect(context: Connection): void {
        console.log(`🔌 Conexión ${context.id} establecida.`);
        context.transitionTo(new ConnectedState());
    }

    public disconnect(context: Connection): void {
        console.warn(`⚠️ La conexión ${context.id} ya está desconectada.`);
    }

    public sendData(context: Connection, data: string): void {
        console.error(`❌ Error: No se pueden enviar datos. La conexión ${context.id} está inactiva.`);
    }
}
```


connection.ts
src > infrastructure > connection.ts > Connection > connect
10 export class Connection implements IConnection {
14 constructor() {
15 this.id = `conn_${Math.random().toString(36).substring(2, 9)}`;
16 this.state = new DisconnectedState();
17 this.connect();
18 }
19 (method) Connection.transitionTo(state: IConnectionState): void
20 public transitionTo(state: IConnectionState): void {
21 this.state = state;
22 }
23
24 public connect(): void {
25 this.state.connect(this);
26 }
27
28 public disconnect(): void {
29 this.state.disconnect(this);
30 }
31
32 public sendData(data: string): void {
33 this.state.sendData(this, data);
34 }
35 }

network-service.ts

src > services > network-service.ts > NetworkService > sendStandardMessage

```typescript
import { ConnectionManager } from '../infrastructure/manager/connection-manager';
import { SecureConnection } from '../infrastructure/patterns/decorator/connection-decorator';

export class NetworkService {
    private manager: ConnectionManager;

    constructor() {
        this.manager = ConnectionManager.getInstance();
    }

    public sendSecureMessage(message: string): void {
        const conn = this.manager.getConnection();
        if (conn) {
            const secureConn = new SecureConnection(conn);
            secureConn.sendData(message);

            this.manager.releaseConnection(conn);
        } else {
            console.error("No hay conexiones disponibles para envío seguro.");
        }
    }

    public sendStandardMessage(message: string): void {
        const conn = this.manager.getConnection();
        if (conn) {
            conn.sendData(message);
            this.manager.releaseConnection(conn);
        } else {
            console.error("No hay conexiones disponibles.");
        }
    }
}
```

main.ts

src > presentation > main.ts > ...

```ts
import { NetworkService } from '../services/network-service';

const networkService = new NetworkService();

console.log("\nPrueba 1: Envío Estándar");
networkService.sendStandardMessage("Hola Mundo (Texto plano)");

console.log("\nPrueba 2: Envío Seguro");
networkService.sendSecureMessage("Datos Confidenciales (Encriptados)");

console.log("\nPrueba 3: Gestión de Estado");

import { ConnectionManager } from '../infrastructure/manager/connection-manager';
const manager = ConnectionManager.getInstance();
const conn = manager.getConnection();

if (conn) {
    conn.disconnect();
    conn.sendData("Intento fallido");
    conn.connect();
    conn.sendData("Ahora sí funciona");
    manager.releaseConnection(conn);
}
```

Ejecucion de codigo:

```
Inicializando Connection Manager (Singleton)...
🔌 Conexión conn_1obepfp establecida.
🔌 Conexión conn_qxxc7j1 establecida.
🔌 Conexión conn_4xsutdc establecida.
🔌 Conexión conn_fx9q5fp establecida.
🔌 Conexión conn_v46viz0 establecida.

Prueba 1: Envío Estándar
📡 [conn_v46viz0] Enviando: "Hola Mundo (Texto plano)"
🔁 Conexión conn_v46viz0 liberada.

Prueba 2: Envío Seguro
🔒 Encriptando datos para conn_v46viz0...
📡 [conn_v46viz0] Enviando: "[ENCRIPTADO] )sodatpircnE( selaicnedifnoC sotaD"
🔁 Conexión conn_v46viz0 liberada.

Prueba 3: Gestión de Estado
🔌 Conexión conn_v46viz0 cerrada.
❌ Error: No se pueden enviar datos. La conexión conn_v46viz0 está inactiva.
🔌 Conexión conn_v46viz0 establecida.
📡 [conn_v46viz0] Enviando: "Ahora sí funciona"
🔁 Conexión conn_v46viz0 liberada.
📡 [conn_v46viz0] Enviando: "Mensaje desde una nueva conexión"
📡 [conn_fx9q5fp] Enviando: "Otro mensaje desde una conexión reutilizada"
🔁 Conexión conn_fx9q5fp liberada.
pvngu@Omars-MacBook-Pro ExamenUnidad4_Patrones_SanchezLozadaOmar %
```

Conclusiones

En conclusión, se implementaron varios patrones nuevos al proyecto realizado en la unidad 2. El primero de ellos, **Factory Method**, el cual permite crear nuevos tipos de conexiones de una manera sencilla y rápida sin alterar la clase en donde se declara; el patrón **Decorator** se implementó para poder agregar una encriptación a los datos; el patrón **State** permitió eliminar condiciones if-else que se encontraban en la clase Connection para hacerlo mucho más sencillo y que se manejaran en una clase State, la cual nos permite agregar nuevos estados de una manera más sencilla; y por último, la **arquitectura por capas** permitió tener una mejor organización del código al separar procesos largos del archivo principal.