Ejercicio de Programación: Sistema de Emparejamiento de Empleados y Clientes

Descripción del Problema

Debes implementar un sistema que asigne empleados a clientes basándose en compatibilidad de habilidades y presupuesto. El sistema debe encontrar el **máximo número de emparejamientos posibles** donde:

- Cada cliente es asignado a **un solo empleado**
- Cada empleado es asignado a **un solo cliente**
- Un cliente solo puede contratar un empleado si:
  1. La ocupación del empleado coincide con el requerimiento del cliente
  2. El presupuesto del cliente cubre el precio por hora del empleado

Requisitos Técnicos

Implementa la solución usando los siguientes principios y algoritmos:

- **Programación Orientada a Objetos** (POO)
- **Principios SOLID**
- **Algoritmo de Maximum Bipartite Matching** usando el enfoque de Ford-Fulkerson con DFS
- Patrones de diseño para mantener bajo acoplamiento

Componentes Requeridos

Implementa las siguientes clases e interfaces:

1. Clases de Dominio

- `Empleado`: Representa un empleado con propiedades inmutables (nombre, ocupación, precio/hora)

- `Cliente`: Representa un cliente con propiedades inmutables (nombre, ocupación requerida, presupuesto) y método `puede_contratar()`
- `Emparejamiento`: Representa una asignación cliente-empleado validada

2. Interfaces y Implementaciones

- `ILectorArchivos`: Interfaz para cargar datos
- `LectorArchivosTexto`: Implementación que lee empleados/clientes desde archivos CSV con formato:

Listing 1: Formato de archivos

```
# empleados.txt
nombre;ocupacion;precio_por_hora
Juan;Programador;20
...

# clientes.txt
nombre;ocupacion_requerida;presupuesto
Cliente1;Programador;100
...
```

- `IAlgoritmoEmparejamiento`: Interfaz para algoritmos de matching
- `AlgoritmoEmparejamientoBipartito`: Implementa el algoritmo de Ford-Fulkerson usando DFS
- `IVisualizadorResultados`: Interfaz para mostrar resultados
- `VisualizadorConsola`: Muestra emparejamientos en consola

3. Componentes Principales

- `GestorEmparejamientos`: Coordina el flujo completo
- `MenuInteractivo`: Ofrece menú con opciones:
  - Crear archivos de ejemplo y ejecutar
  - Usar archivos personalizados
  - Salir
- Función `main()` que inicia la aplicación

Funcionalidad del Algoritmo

El algoritmo debe:

1. Construir un grafo bipartito donde:
   - **Nodos izquierdos**: Clientes
   - **Nodos derechos**: Empleados

- **Aristas**: Conexiones donde `cliente.puede_contratar(empleado)` es True

2. Aplicar el algoritmo de Ford-Fulkerson usando DFS para encontrar el matching máximo
3. Usar estructuras:
   - `grafo_compatibilidad`: Diccionario de listas de adyacencia
   - `emparejamiento_clientes`: Asignación cliente → empleado
   - `emparejamiento_empleados`: Asignación empleado → cliente

Requerimientos Adicionales

- Manejo de errores robusto (archivos no existentes, formatos inválidos)
- Validación de datos (precios/presupuestos no negativos)
- Separación clara de responsabilidades
- Código modular y extensible
- Documentación clara de clases y métodos

Ejemplo de Salida

La solución debe mostrar resultados como:

```
=== RESULTADOS DEL EMPAREJAMIENTO ===

Emparejamientos encontrados:
  Cliente1 (necesita Programador, presupuesto $100) - Juan (
    ↪ Programador, $20/h)
  ...
Cantidad total de emparejamientos: 2
```

La solución completa es el código Python