PRÁCTICA 7ª: Clases derivadas y polimorfismo. Derivación de la clase `CCliente` y `CEmpleado` tomando como clase base la clase `CFicha` de la práctica 5ª. Implementación de la clase `CEmpresa`.

OBJETIVOS: Clases derivadas y métodos virtuales.

TEMPORIZACIÓN:
Publicación del enunciado: Semana del 11 de noviembre.
Entrega: Semana del 2 de diciembre.
Límite de entrega (con penalización): Semana del 9 de diciembre.

BIBLIOGRAFÍA
Programación orientada a objetos con C++
Autor: Fco. Javier Ceballos
Editorial: RA-MA.

La clase `CEmpleado` tendrá, además de la funcionalidad heredada de la clase `CFicha`, los siguientes datos miembro privados:

```
string m_sCategoria;   // (ej.: "Administrativo", "Técnico"...)
int m_nAntiguedad;     // (ej.: 3)
```

La cadena `m_sCategoria` estará vacía mientras no haya ninguna categoría especificada. La antigüedad tomará inicialmente el valor `0`.

Los métodos siguientes servirán para modificar estos datos miembro:

```
void SetCategoria(const string& sCategoria);
void SetAntiguedad(int nAntiguedad);
```

La declaración de esta clase y sus métodos inline se escribirán en el fichero `empleado.h`, y el resto de las definiciones en el fichero `empleado.cpp`.

La clase **`CCliente`** (también derivada de `CFicha`) solo añadirá un dato miembro:

```
string m_sDNI;       // (ej.: "12345678V"...)
```

y el correspondiente método `Set...`:

```
void SetDNI(const string& sDNI);
```

El funcionamiento será análogo al de la clase `CEmpleado`.

La declaración de esta clase y sus métodos inline se escribirán en el fichero `cliente.h`, y el resto de las definiciones en el fichero `cliente.cpp`.

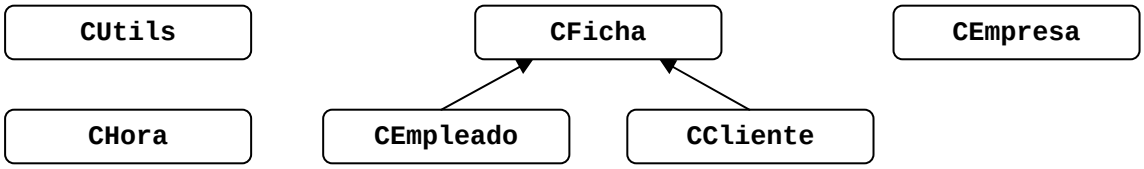
Se añadirá a la clase `CFicha` un método virtual `Visualizar` y se redefinirá en cada una de las clases derivadas (`CCliente` y `CEmpleado`) con el fin de poder mostrar al usuario los datos correspondientes a los objetos de cada clase.
Se declarará una nueva clase `CEmpresa` mediante la cuál se implementará una matriz, `personas`, de punteros a objetos de tipo `CEmpleado` o `CCliente`, con un tamaño máximo de `nElementosMax` que será pasado como argumento al constructor. Habrá que declarar y definir:

- Un constructor que cree la matriz e inicie a `NULL` cada uno de sus punteros. Recibirá como parámetro el número máximo de elementos de la matriz.
- Un destructor que libere la memoria ocupada por la matriz de punteros.
- El constructor copia y el operador de asignación. Observe que la clase `CEmpresa` contiene punteros a objetos. Eso significa que el operador de asignación y el constructor copia por omisión sólo copiarán direcciones, no duplicarán los objetos apuntados. Para escribir la clase de forma que copie y destruya objetos dinámicamente, deberá implementar un método virtual `Clonar` en toda la jerarquía de `CFicha` y definir un destructor virtual en la misma clase. El método `Clonar` devolverá un puntero a una copia del objeto para el cual fue invocada.
- Un método `AgregarPersona` que permita añadir a la matriz los datos de una persona, ya sea empleado o cliente. Este método recibirá como parámetro un puntero a la clase base `CFicha` y añadirá a la matriz `personas` una copia del objeto apuntado por este puntero, creada con el método que corresponda.
- Un método `VisualizarEmpresa`, que llame al método `Visualizar` para cada uno de los empleados o clientes de la matriz.
- Un método `static Es`**`Empleado`** que devuelva `true` si la persona pasada como argumento es un empleado y `false` en caso contrario. Utilice `dynamic_cast<>` para saber si una determinada ficha contiene un empleado o un cliente.
- Un método `Visualizar`**`Empleados`** que muestre los datos de todos los empleados presentes en la matriz `personas`.
- Para poder acceder a los elementos de un objeto de la clase `CEmpresa` como si de una matriz se tratara, se sobrecargará el operador de indexación. Este operador devolverá un puntero al objeto que está en la posición de la matriz indicada por el índice pasado como argumento.

Al trabajar con una matriz con un número máximo de elementos puede serle útil añadir a la clase `CEmpresa` un método `EmpresaLlena` que devuelva `true` cuando la matriz esté llena (último elemento no apunta a `NULL`). Esto facilitará al usuario de esta biblioteca de clases no realizar operaciones de adición sobre la matriz cuando ésta esté llena. También podría añadir un método `GetNumElementosMax`.

La declaración de esta clase y sus funciones inline se escribirán en el fichero `empresa.h`, y el resto de las definiciones en el fichero `empresa.cpp`.

Resumiendo, la arquitectura de su aplicación quedará formada por las siguientes clases:



El programa principal se escribirá en un fichero `práctica7.cpp` y contendrá un menú como el siguiente:

```
1. Introducir empleado.
2. Introducir cliente.
3. Buscar por nombre.
4. Mostrar empresa.
5. Mostrar empleados.
6. Copia de seguridad de la empresa.
7. Restaurar copia de seguridad.
8. Salir.
```

La opción 1 añadirá un empleado al objeto `empresa` y la opción 2 añadirá un cliente.

La opción 3 solicitará el nombre de una persona y la buscará en la `empresa`. Si la encuentra mostrará sus datos indicando si se trata de un empleado o de un cliente.

La opción 4 mostrará todas las personas de la empresa, empleados y clientes. La opción 5 mostrará solamente los empleados.

La opción 6 hará un duplicado en memoria (referenciado por `copia_empresa`) del objeto `empresa` actual. Una vez hecha la copia de seguridad supondremos que, haciendo pruebas, se altera la composición de esa empresa (por ejemplo, añadiendo nuevos clientes) y finalizadas las pruebas, se volverá a la composición que había cuando se hizo la copia de seguridad. La opción 7 permitirá esta última acción.

No se permitirá restaurar una copia de seguridad cuando no haya una.
No se permitirá realizar una copia de seguridad cuando ya exista una.
Cuando se restaure una copia de seguridad esta será destruida.

Verifique que no hay lagunas de memoria.

Realizar una segunda versión sustituyendo la matriz de punteros a objetos por un vector (plantilla `vector<...>`) definido así: `vector<CFicha *> personas`. Como esta clase tiene, entre otros, un método `size` que devuelve el número de elementos del vector, podemos prescindir del atributo `nElementosMax` y del método `GetNumElementosMax`. También podemos prescindir del método `EmpresaLlena` puesto que ahora `personas` es una matriz dinámica. El constructor `CEmpresa` con un argumento de tipo entero puede ahora asegurar una cantidad de memoria para el número de elementos especificado utilizando el método `reserve`. También, podemos añadir a la clase `CEmpresa` un método `NumPersonas` que devuelva el número de elementos que tenga en ese instante el vector. Finalmente, repase la funcionalidad de `vector<...>` como `push_back`, `resize`, `clear`, etc. por si tiene que utilizarlos.