**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 CRegistroDiario.

**OBJETIVOS:** Clases derivadas y métodos virtuales.

# TEMPORIZACIÓN:

**Publicación del enunciado:** Semana del 12 de noviembre.

**Entrega:** Semana del 3 de diciembre.

**Límite de entrega (con penalización):** Semana del 10 de diciembre.

# BIBLIOGRAFÍA

**Programación orientada a objetos con C++**

Autor: Fco. Javier Ceballos Editorial: RA-MA.

El personal de recepción de una empresa hace un registro diario de los empleados que acuden a trabajar (objetos CEmpleado) y de los clientes que visitan la empresa (objetos CCliente). Este registro diario va a estar modelado por una clase CRegistroDiario.

La clase CEmpleado, además de la funcionalidad heredada de la clase CFicha, tendrá 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.

Las clases CCliente y CEmpleado permitirán construir objetos iniciados con 0, 1, 2,… etc., valores pasados como argumentos, y la implementación de cada uno de los constructores deberá utilizar una lista de iniciadores.

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. Añada también un destructor virtual (aunque para esta práctica no sea necesario) y declare esta clase abstracta.

Se escribirá una nueva clase CRegistroDiario que definirá una matriz, personas, de punteros a objetos de tipo CEmpleado o CCliente, de tamaño nElementosMax. Esta clase incluirá, básicamente, la siguiente funcionalidad:

* Los atributos personas y nElementosMax.
* Un constructor que cree la matriz personas con un tamaño máximo de nElementosMax, valor que será pasado como argumento cuando se invoque a dicho constructor, e inicie cada uno de sus elementos a NULL. Si el valor pasado como argumento es menor o igual que 0, lanzará una excepción de tipo **const char \*** indicando lo sucedido: *El número de elementos es <= 0*.
* Un destructor que libere la memoria ocupada por la matriz de punteros y asignará a

personas el valor NULL.

* El constructor copia y el operador de asignación; no realizar la asignación cuando ambos objetos, origen y destino, sean el mismo objeto. Observe que la clase CRegistroDiario contiene punteros a objetos derivados de CFicha; su valor debe de ser NULL cuando, aún, no tienen un objeto asignado. Eso significa que el operador de asignación y el constructor copia por omisión sólo copiarán direcciones, no duplicarán los objetos apuntados. Para poder copiar objetos derivados de CFicha dinámicamente, la jerarquía de CFicha deberá implementar un método virtual Clonar. El método Clonar devolverá un puntero a una copia del objeto para el cual fue invocada. Escribir el constructor copia en función del operador de asignación.
* Un método AgregarPersona que permita añadir a un objeto CRegistroDiario un objeto CEmpleado o CCliente. Este método recibirá como parámetro un puntero a la clase base CFicha y añadirá a la matriz personas de CRegistroDiario una copia del objeto apuntado por este puntero. El método devolverá true si el objeto se añadió y false si no se pudo realizar esta operación, por ejemplo, porque la matriz ya está llena o porque el valor del parámetro es NULL.
* Un método VisualizarRegistro, que llame al método Visualizar para cada uno de los empleados o clientes de CRegistroDiario para mostrar sus datos.
* 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 un determinado registro corresponde a un empleado o a un cliente.
* Un método Visualizar**Empleados** que muestre sólo los datos de los empleados registrados en CRegistroDiario.
* Para poder acceder a los elementos de un objeto de la clase CRegistroDiario 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; este método deberá también validar que el índice esté dentro de los límites establecidos por la matriz.

Al trabajar con una matriz con un número máximo de elementos, será útil añadir a la clase CRegistroDiario un método RegistroLleno 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 será útil disponer de un método GetNumElementosMax.

La declaración de esta clase y sus funciones inline se escribirán en el fichero

registro.h, y el resto de las definiciones en el fichero registro.cpp.

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

**CHora**

**CUtils**

**CCliente**

**CEmpleado**

**CFicha**

**CRegistroDiario**

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 registro diario.
  5. Mostrar empleados.
  6. Copia de seguridad del registro diario.
  7. Restaurar copia de seguridad.
  8. Salir.

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

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

La opción 4 mostrará todas las personas del registro, empleados y clientes. La opción 5 mostrará solamente los empleados.

La opción 6 hará un duplicado en memoria del objeto registro actual (utilice una sentencia como esta: copia\_registro = new CRegistroDiario(registro)). Una vez hecha la copia de seguridad supondremos que, haciendo pruebas, se altera la composición de ese registro (por ejemplo, añadiendo nuevos clientes y/o empleados) 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.

Verifique que el constructor copia CRegistroDiario funciona correctamente. Para ello, suponiendo que existe un objeto registro de la clase CRegistroDiario, escriba en el lugar que considere adecuado de main un código análogo al siguiente:

**CRegistroDiario rdNuevo(registro);**

Verifique qué sucede cuando, al construir el objeto de la clase CRegistroDiario, el número de personas es menor o igual que cero. Atrape la excepción y muestre el mensaje que envuelve el objeto de excepción.

**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 RegistroLleno puesto que ahora personas es una matriz dinámica. El constructor CRegistroDiario 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 CRegistroDiario 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.**