Bloque práctico 1.1 (Obligatorio): *C++*

Programación modular, herencia polimorfismo.

El objetivo de la práctica es programar una aplicación informática para una empresa de telefonía móvil que desea informatizar la facturación de sus clientes.

1) Programar la clase Cliente y la clase Fecha

Para cada cliente la empresa de telefonía desea guardar la siguiente información:

dni: dni del cliente (sin incluir la letra) que lo identifica de forma única. Este campo es de tipo long int.

nombre: nombre y apellidos del cliente. Este campo es de tipo cadena de caracteres (**char * de C**, no *string* de C++)

fechaAlta: fecha (dd/mm/aa) en la que el cliente se da de alta en la compañía. **Este campo es un objeto de tipo *Fecha*** (clase que permite trabajar con Fechas).

Programe las clases `Fecha` y `Clientes` y los métodos **que sean estrictamente necesarios** para que el siguiente código pueda ser ejecutado y produzca la salida siguiente:

Prueba1.cpp

```
#include <cstdlib>
#include <iostream>
#include "Fecha.h" //definicion de la clase Fecha
#include "Cliente.h" // definicion de la clase Cliente

using namespace std;

int main() {
  Fecha f1(29,2,2001), f3(29,2,2004), f4(29,2,1900); //Fecha f5; //no permitido
  const Fecha f2=f1; //indica que metodo se esta ejecutando aqui
  f1.setFecha(f3.getDia()-3, f3.getMes()-2, 2007); //29-3/2-2/2007 --> f1=26/1/2007
  cout << "Fechas: "; f1.ver(); cout << ", "; f2.ver(); cout << ", ";
                      f3.ver(); cout << ", "; f4.ver(); cout << endl;
  if (f3.bisiesto() && !f2.bisiesto() && f4.bisiesto()==false)
    cout << f3.getAnio() << " es bisiesto, " << f2.getAnio() << " y " << f4.getAnio() << " no\n";
  f4.setFecha(31, 12, 2000); //f4=31/12/2000
  f3=f4++; //indica que método/s se esta ejecutando aqui
  ++f4;
  f1=2+f2+3;
  cout << "Fechas: "; f1.ver(); cout << ", "; f2.ver(); cout << ", ";
                      f3.ver(); cout << ", "; f4.ver(); cout << endl;
  Cliente *p = new Cliente(75547001, "Susana Diaz", f1);
  f1.setFecha(7,10,2015);
  Cliente c(75547999, "Juan Sin Miedo", Fecha(29,2,2000));
  const Cliente j(44228547, "Luis", f1);
  c.setNombre("Juan Palomo");
  if (j==c)
    cout << "\nj y c son iguales\n";
  else
    cout << "\nj y c no son iguales\n";
  cout << p->getDni() << " - " << c.getNombre() << ": " << j.getFecha() << endl;
  cout << *p << "\n" << c << "\n" << j << "\n";
  c = *p;
  p->setNombre("Susanita"); p->setFecha(p->getFecha()+10);
  cout << "\nDatos de los clientes: \n";
  cout << *p << "\n" << c << "\n" << j << "\n";
  delete p;  p = NULL;
  system("PAUSE");  return 0;
}
```

Salida:

```
Fechas: 26/01/2007, 28/02/2001, 29/02/2004, 28/02/1900
2004 es bisiesto, 2001 y 1900 no
Fechas: 05/03/2001, 28/02/2001, 31/12/2000, 02/01/2001

j y c no son iguales
75547001 - Juan Palomo: 07 oct 2015
Susana Diaz (75547001 - 05 mar 2001)
Juan Palomo (75547999 - 29 feb 2000)
Luis (44228547- 07 oct 2015)

Datos de los clientes:
Susanita (75547001 - 15 mar 2001)
Susana Diaz (75547001 - 05 mar 2001)
Luis (44228547- 07 oct 2015)
Presione una tecla para continuar . . .
```

Consideraciones:

a) **Todos los atributos de las clases deben ser privados** (se deben proporcionar métodos `get` y `set` para poder consultarlos y modificarlos, si son necesarios).

b) Programe la clase Fecha y Cliente **de forma que no quede memoria sin liberar** una vez utilizados objetos de dicho tipo. **La clase Fecha** debe ser programada de forma que **se ejecute lo más rápida y eficientemente posible**[1, 2], mientras que **la clase Cliente debe ser lo más robusta posible**[3], de forma que la integridad de la clase no quede expuesta y no haya ningún fallo de seguridad que permita modificar los atributos (propiedades) de un Cliente desde el main( ) sin utilizar explícitamente los métodos públicos diseñados para ello.

c) **Con respecto a la clase Fecha**, a la hora de establecer la fecha, **si el día y/o el mes no es válido se asigna por defecto el día y/o mes válido más cercano al indicado**. En cuanto al año se supone que el año indicado siempre es correcto.

La fecha 29/2/2001 se convierte en 28/2/2001 ya que el 29 no es un día valido para el mes 2 al no ser bisiesto. La fecha 33/0/2002 pasa a ser 31/1/2002 (el mes más cercano al 0 es el 1 y el día más cercano al 33 es el 31). Otros ejemplos: 0/14/2004 → 1/12/2004, 31/09/2007 → 30/09/2007, 29/2/2000 (es correcto al ser el año 2000 bisiesto), 29/2/2100 → 28/2/2100 (el año 2100 no va a ser bisiesto)

Nota: No todos los años múltiplos de 4 son bisiestos (averiguad cuando es bisiesto)

d) **Implemente únicamente los métodos y/o funciones que sean necesarios y que explícitamente se invocan en el main( ) de ejemplo**, es decir, no se permite crear métodos y/o funciones que no sean los invocados en dicho main( ).

e) Siguiendo estrictamente lo indicado en el apartado d) **¿funciona el programa? ¿Sabrías indicar el lugar exacto donde el programa falla y a qué es debido?** Corrige el programa para que funcione correctamente.

Consejos/Ayuda:

1. Una función amiga es más rápida que una no amiga ya que la amiga puede acceder directamente a los atributos del objeto de la clase mientras que la no amiga debe hacerlo indirectamente llamando a los métodos públicos.
2. Pasar un parámetro por referencia es más rápido que pasarlo por valor ya que se evita tener que hacer una copia
3. Las funciones amigas rompen el principio de encapsulamiento de la información, rompiendo la robusted de la clase al permitir que funciones ajenas a ella puedan acceder a su parte privada.
4. Sumar una serie de días a una determinada fecha es equivalente a incrementar dicha fecha tantas veces como días queremos sumar… (¿lo pillas?)