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?)

2) Programar la clase *Contrato y sus clases derivadas*

La empresa de telefonía desea informatizar los contratos de sus clientes, que puede ser de 2 tipos, contrato de tarifa plana y contrato movil:

- **ContratoTP:** Se paga una cantidad fija al mes, independientemente de la cantidad de llamadas realizadas y el número de minutos hablados hasta un cierto límite. Si el cliente supera el límite de minutos, el exceso de minutos se tarifica a un precio fijo e inamovible de 0.15 céntimos/minuto.

  factura= precio tarifa plana + (exceso limite minutos x 0.15)

- **ContratoMovil:** Estos clientes pagan en función del número de minutos que hablan al mes.

  factura = precio minuto x minutos hablados

  Por cuestiones de estudio de mercado, a la compañía le interesa saber la nacionalidad de este tipo de clientes (cree un atributo de tipo char * en este clase concreta para guardar la nacionalidad del cliente).

Actualmente la oferta que tiene la compañía para los clientes de tarifa plana es de 300 minutos por 10 euros (programar la clase de forma que cuando la compañía actualice la oferta de tarifa plana los cambios se reflejen en todos los clientes actuales y futuros de la compañía).

En cuanto a los clientes de tarifa móvil el precio por minuto es específico y particular para cada cliente (un cliente puede tener un precio por minuto diferente a otro cliente).

Para cada contrato la empresa desea guardar la siguiente información:

Número contrato (`int idContrato`): debe ser único y generarse automáticamente
dni contrato: (`long int dniContrato`): permite saber a quién pertenece el contrato
Fecha contrato (`Fecha fechaContrato`)

Una vez un contrato es dado de alta en la aplicación, el número de contrato **no puede ser modificado** (debe generarlo automáticamente la aplicación de forma que no puede haber 2 contratos con el mismo número) aunque si puede ser consultado.

El dni asociado al contrato y la fecha del contrato si pueden ser modificados a posteriori (por supuesto, también pueden ser consultados).

Programe una clase **Contrato** para representar un contrato genérico. A partir de dicha clase y mediante herencia programe las clases derivadas **ContratoTP** y **ContratoMovil** de forma que el siguiente código pueda ser ejecutado y produzca la salida siguiente:

Prueba2.cpp

```
#include <cstdlib>
#include <iostream>
#include <iomanip> //std::setprecision
#include "Fecha.h" //definicion de la clase Fecha
#include "Contrato.h" // definicion de la clase Contrato
#include "ContratoTP.h" // definicion de la clase ContratoTP
#include "ContratoMovil.h" // definicion de la clase ContratoMovil

using namespace std;

int main(int argc, char *argv[]) {
  Fecha f1(29,2,2001), f2(f1), f3(29,2,2004);
  cout << "Fechas: ";
  f1.ver(); cout << ", "; f2.ver(); cout << ", "; f3.ver(); cout << endl;

  Contrato *p = new Contrato(75547111, f1), c(23000111, Fecha(2,2,2002));
  cout << ContratoTP::getLimiteMinutos() << " - " << ContratoTP::getPrecio() << endl;
  ContratoTP ct1(17333256, f1, 250); //habla 250 minutoe
  ContratoTP ct2(12555100, f3, 320); //habla 320 minutos
  ContratoTP ct3(ct1);
  ContratoMovil cm1(17333256, f1, 0.12, 100, "ESPAÑOL"); //habla 100 minutos
  ContratoMovil cm2(17000000, Fecha(3,3,2003), 0.10, 180, "FRANCES"); //habla 180 minutos
  ContratoMovil cm3(cm2);
  p->ver(); cout << "\n"; c.ver(); cout << endl;
  ct1.ver(); cout << endl; ct2.ver(); cout << "\n"; ct3.ver(); cout << "\n";
  cm1.ver(); cout << endl; cm2.ver(); cout << "\n"; cm3.ver(); cout << "\n";
  cout << p->getIdContrato() << ct2.getIdContrato() << cm2.getIdContrato() << endl;

  cout << setprecision(2) << fixed; //a partir de aqui float se muestra con 2 decimales

  cout << "Facturas: " << ct1.factura() <<"-"<< ct2.factura() <<"-"<< cm1.factura() << endl;
  ContratoTP::setTarifaPlana(350, 12); //350 minutos por 12 euros
  p->setDniContrato(cm1.getDniContrato());
  ct3.setFechaContrato(p->getFechaContrato()+1);
  cm3.setNacionalidad(cm1.getNacionalidad());
  cm2.setPrecioMinuto(cm1.getPrecioMinuto()+0.02);
  cm1.setMinutosHablados(ct2.getMinutosHablados()/2);
  ct1.setMinutosHablados(cm3.getMinutosHablados()*2);
  cout << *p <<"\n"<< c << endl;
  cout << ct1 <<endl<< ct2 <<"\n"<< ct3 <<"\n"<< cm1 <<"\n"<< cm2 <<endl<< cm3 << endl;

  system("PAUSE");
  return 0;
}
```

Salida:

```
Fechas: 28/02/2001, 28/02/2001, 29/02/2004
300 - 10

75547111 (1 - 28/02/2001)
23000111 (2 - 02/02/2002)
17333256 (3 - 28/02/2001) 250m, 300(10)
12555100 (4 - 29/02/2004) 320m, 300(10)
17333256 (5 - 28/02/2001) 250m, 300(10)

17333256 (6 - 28/02/2001) 100m, ESPAÑOL 0.12
17000000 (7 - 03/03/2003) 180m, FRANCES 0.1
17000000 (8 - 03/03/2003) 180m, FRANCES 0.1

147
Facturas : 10.00-13.00-12.00

17333256 (1 - 28 feb 2001)
23000111 (2 - 02 feb 2002)
17333256 (3 - 28 feb 2001) 360m, 350(12.00) - 13.50€
12555100 (4 - 29 feb 2004) 320m, 350(12.00) - 12.00€
17333256 (5 - 01 mar 2001) 250m, 350(12.00) - 12.00€

17333256 (6 - 28 feb 2001) 160m, ESPAÑOL 0.12 - 19.20€
17000000 (7 - 03 mar 2003) 180m, FRANCES 0.14 - 25.20€
17000000 (8 - 03 mar 2003) 180m, ESPAÑOL 0.10 - 18.00€

Presione una tecla para continuar . . .
```

Bloque práctico 1.2: *C++*

Polimorfismo y *dinamic_cast< >*.

En esta segunda parte de la práctica 1 utilizaremos otras capacidades adicionales de programación orientada a objetos, tanto básicas como avanzadas, de *C++*. El objetivo de la práctica es programar una clase Empresa polimórfica capaz de agrupar los diferentes tipos de contratos de los clientes en dicho objeto.

0) Antes de empezar.

Cree un nuevo directorio denominado `Empresa` y dentro un proyecto para compilar todos los ficheros necesarios para realizar esta segunda parte de la práctica siguiendo la misma filosofía modular de la práctica anterior (por cada clase creamos un .h y un .cpp).

1) Hacer que `Contrato` muestre un comportamiento polimórfico dinámico: polimorfismo.

Modifica el programa para que éste tenga un comportamiento polimórfico y permita que el siguiente código añadido al final del fichero Prueba2.cpp produzca la salida indicada.

añadir al final de Prueba2.cpp, antes del system ("PAUSE")

```
  Contrato *t[4];
  t[0]=p;  t[1]=&c;  t[2]=&ct2;  t[3]=&cm1;
  cout << "\n-- Datos de los contratos: -- \n";
  t[3]->setDniContrato(75547111);
  for(int i=0; i<4; i++) {
    t[i]->setFechaContrato(t[i]->getFechaContrato()+2);
    t[i]->ver(); cout << endl;
  }
```

Salida:

```
-- Datos de los contratos: --
17333256 (1 - 02/03/2001)
23000111 (2 - 04/02/2002)
12555100 (4 - 02/03/2004) 320m, 350(12.00)
75547111 (6 - 02/03/2001) 160m, ESPAÑOL 0.12
```

2) Hacer que `Contrato` sea una clase abstracta

Realice las modificaciones necesarias para que el compilador restrinja la instanciación de la clase *`Contrato`*, de forma que no se puedan crear objetos de dicha clase.

Pruebe el programa anterior y compruebe que el compilador da error cuando intentamos crear instancias de la clase *`Contrato`*. Comenta las líneas del fichero Prueba2.cpp que ahora dan problemas y compruebe que todo funciona correctamente.

Si eliminamos la posibilidad de crear objetos de la clase *`Contrato`*, ¿es esto equivalente a quitar la clase de nuestro diseño?

3) Programar la clase *Empresa*

Programe la clase `Empresa`, la cual debe permitir crear y eliminar `Clientes` y `Contratos` (ya sean de tipo Tarifa Plana o Movil) en nuestra aplicación.

Para crear un contrato es necesario primero que el cliente sea dado de alta en la aplicación, introduciendo sus datos (los indicados en la clase `Cliente`). Una vez dado de alta un cliente, éste puede contratar las tarifas (Tarifa Plana o Movil) que necesite.

Programe en la clase *Empresa* los siguientes métodos:

- `crearContrato()`: Este método debe tener en cuenta si el contrato es para un nuevo cliente o para uno ya existente, para evitar añadirlo más de una vez. Para ello el método `crearContrato()` pedirá el DNI del titular del contrato y buscará si ya es o no cliente de la empresa. Si es un nuevo cliente (no existe un cliente con dicho DNI en la empresa) lo dará de alta, pidiendo los datos necesarios. Una vez el cliente es identificado o dado de alta, pedirá los datos del tipo de contrato que desea abrir (`ContratoTP` o `ContratoMovil`).

- `cancelarContrato()`: que se encargará de darlo de baja de la Empresa (y eliminarlo de la memoria). Para ello bastará con introducir el número de contrato (`idContrato`) a eliminar y pasarlo como parámetro al método. Si el contrato no existe, el método debe mostrar un mensaje por pantalla indicando tal circunstancia.

- `bajaCliente()`: Si un cliente desea darse de baja, bastará con pasar su `dni` como parámetro al método el cual se encargará de eliminar todos sus contratos y sus datos personales de la Empresa.

Implementa la clase *Empresa* de manera que mantenga un array con la lista de contratos y otro con la lista de clientes que va creando. Hacer que la empresa admita como máximo 100 clientes (uso de array estático) de forma que no se permita superar dicha cifra. En cuanto al número de contratos hacer que éste sea ilimitado (uso de array dinámico, de tamaño inicial de 10, que se duplica cada vez que se llena: 10, 20, 40, 80…)

4) *dynamic_cast<> y typeid*.

Programe los métodos `descuento(float porcentaje)` que permite rebajar la tarifa de todos los contratos Movil en el porcentaje indicado, y `nContratosTP()` que devuelve el número de contratos de Tarifa Plana existentes.

Deberá utilizar la funcionalidad `dynamic_cast<>` y `typeid` que se definen en la cabecera `<typeinfo>` para la programación de estos métodos.

Utilice el operador `dynamic_cast<>` para implementar el primer método para poder hacer una conversión dinámica del tipo base `Contrato` al tipo `ContratoMovil` ya que hay que acceder a funciones sólo existentes en la clase derivada `ContratoMovil` a través de punteros a la clase base `Contrato`. Para el segundo método utilice el operador `typeid` que permite determinar el tipo de un objeto en tiempo de ejecución.

CONSIDERACIONES A TENER EN CUENTA:

Para no alargar la práctica en exceso, a la hora de programar la clase Empresa vamos a suponer que NUNCA vamos a asignar una Empresa a otro Empresa en el `main()` y que NUNCA vamos a pasar una Empresa por copia ni a devolverlo en ningún método.

Por tanto al suponer que NUNCA vamos a utilizar en el `main()` el operador de asignación ni el constructor de copia, no programaremos dichos métodos, aunque la clase `Empresa` tenga atributos de tipo puntero en su interior (esta es una simplificación que vamos a hacerle a la práctica para acortarla un poco).

Teniendo en cuenta lo dicho anteriormente, programe la clase `Empresa` (así como los métodos de dicha clase) de forma que el siguiente código de prueba pueda ser ejecutado y produzca la salida siguiente:

Prueba3.cpp

```
#include <cstdlib>
#include <iostream>
#include <iomanip> //std::setprecision
#include "Fecha.h" //definicion clase Fecha
#include "Cliente.h" // definicion clase Cliente
#include "Contrato.h" // definicion de la clase Contrato
#include "ContratoTP.h" // definicion de la clase ContratoTP
#include "ContratoMovil.h" // definicion de la clase ContratoMovil
#include "Empresa.h" // definicion de la clase Empresa

using namespace std;

int main(int argc, char *argv[])
{
  bool ok;
  Empresa Yoigo;

  cout << setprecision(2) << fixed; //a partir de aqui float se muestra con 2 decimales
  cout << endl << "APLICACION DE GESTION TELEFONICA\n" << endl;
  Yoigo.cargarDatos(); //crea 3 clientes y 7 contratos. metodo creado para no
  Yoigo.ver();         //tener que meter datos cada vez que pruebo el programa
  cout <<"Yoigo tiene " << Yoigo.nContratosTP() << " Contratos de Tarifa Plana\n\n";

  Yoigo.crearContrato(); //ContratoMovil a 37000017 el 01/01/2017 con 100m a 0.25
  Yoigo.crearContrato(); //ContratoTP a 22330014 (pepe luis) el 2/2/2017 con 305m

  ok=Yoigo.cancelarContrato(28);   //este Contrato no existe
  if (ok) cout << "Contrato 28 cancelado\n"; else cout << "El Contrato 28 no existe\n";

  ok=Yoigo.cancelarContrato(4);    //este Contrato si existe
  if (ok)
    cout << "El Contrato 4 ha sido cancelado\n";
  else
    cout << "El Contrato 4 no existe\n";

  ok=Yoigo.bajaCliente(75547001); //debe eliminar el cliente y sus 3 Contratos
  if (ok) cout << "El cliente 75547001 y sus Contratos han sido cancelados\n";
  else cout << "El cliente 75547001 no existe\n";

  Yoigo.ver();

  Yoigo.descuento(20);
  cout << "\nTras rebajar un 20% la tarifa de los ContratosMovil...";
  Yoigo.ver();
  cout <<"Yoigo tiene " << Yoigo. nContratosTP () << " Contratos de Tarifa Plana\n";

  system("PAUSE");
  return 0;
}
```

Salida:

```
APLICACION DE GESTION TELEFONICA

La Empresa tiene 3 clientes y 7 contratos
Clientes:
Peter Lee (75547001 - 28 feb 2001)
Juan Perez (45999000 - 29 feb 2000)
Luis Bono (37000017 - 31 ene 2002)

Contratos:
75547001 (1 - 28 feb 2001) 110m, DANES 0.12 - 13.20€
75547001 (2 - 31 ene 2002) 170m, DANES 0.09 - 15.30€
37000017 (3 - 01 feb 2002) 250m, 300(10.00) - 10.00€
75547001 (4 - 28 feb 2001) 312m, 300(10.00) - 11.80€
45999000 (5 - 31 ene 2002) 202m, ESPAÑOL 0.10 - 20.20€
75547001 (6 - 31 ene 2002) 80m, DANES 0.15 - 12.00€
45999000 (7 - 01 feb 2002) 400m, 300(10.00) - 25.00€
Yoigo tiene 3 contratos de Tarifa Plana

Introduzca dni: 37000017
Tipo de Contrato a abrir (1-Tarifa Plana, 2-Movil): 2
Fecha del contrato
dia: 1 mes: 1 anio: 2017
minutos hablados: 100
Precio minuto: 0.25
Nacionalidad: alemán

Introduzca dni: 22330014
Nombre del cliente: pepe luis
dia: 2 mes: 2 anio: 2017
Tipo de Contrato a abrir (1-Tarifa Plana, 2-Movil): 1
Fecha del contrato
dia: 2 mes: 2 anio: 2017
minutos hablados: 305

El contrato 28 no existe
El contrato 4 ha sido cancelado
El cliente 75547001 y sus contratos han sido cancelados

La Empresa tiene 3 clientes y 5 contratos
Clientes:
Juan Perez (45999000 - 29 feb 2000)
Luis Bono (37000017 - 31 ene 2002)
pepe luis (22330014 - 02 feb 2017)

Contratos:
37000017 (3 - 01 feb 2002 ) 250m, 300(10.00) - 10.00€
45999000 (5 - 31 ene 2002 ) 202m, ESPAÑOL 0.10 - 20.20€
45999000 (7 - 01 feb 2002 ) 400m, 300(10.00) - 25.00€
37000017 (8 - 01 ene 2017 ) 100m, aleman 0.25 - 25.00€
22330014 (9 - 02 feb 2017 ) 305m, 300(10.00) - 10.75€

Tras rebajar un 20% la tarifa de los ContratosMovil...

La Empresa tiene 3 clientes y 5 contratos
Clientes:
Juan Perez (45999000 - 29 feb 2000)
Luis Bono (37000017 - 31 ene 2002)
pepe luis (22330014 - 02 feb 2017)

Contratos:
37000017 (3 - 01 feb 2002 ) 250m, 300(10.00) - 10.00€
45999000 (5 - 31 ene 2002 ) 202m, ESPAÑOL 0.08 - 16.16€
45999000 (7 - 01 feb 2002 ) 400m, 300(10.00) - 25.00€
37000017 (8 - 01 ene 2017 ) 100m, aleman 0.20 - 20.00€
22330014 (9 - 02 feb 2017 ) 305m, 300(10.00) - 10.75€

Yoigo tiene 3 contratos de Tarifa Plana

Presione una tecla para continuar . . .
```