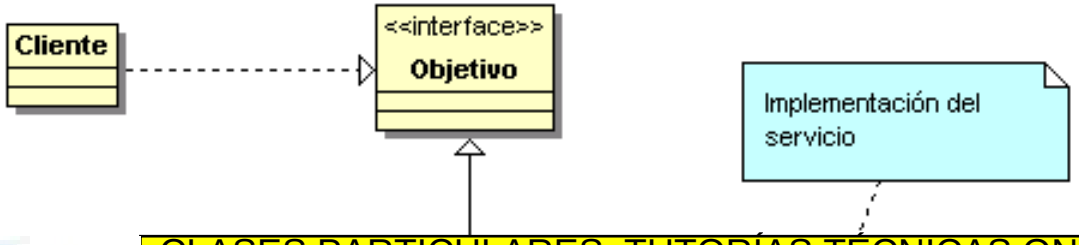
Capítulo VI: DOO
II Parte

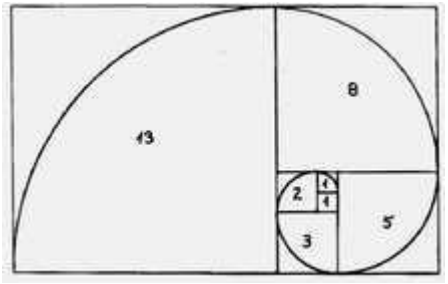
carlos.platero@upm.es (C-305)

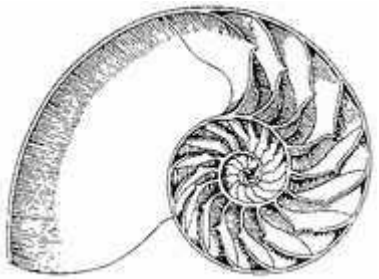
Adaptador(GoF)

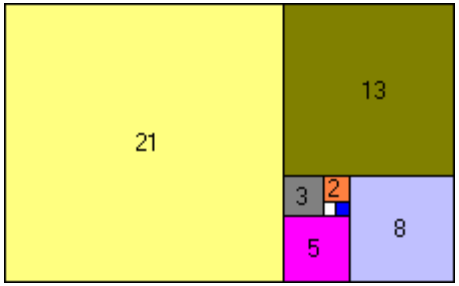
- *Problema:* ¿Cómo resolver interfaces incompatibles, o proporcionar una interfaz estable para componentes parecidos con diferentes interfaces?
- *Solución:* Convierta la interfaz original de una componente en otra, mediante un objeto adaptador intermedio.
- El propósito de este patrón es convertir la interfaz de una clase en otra interfaz que es la que esperan los clientes.

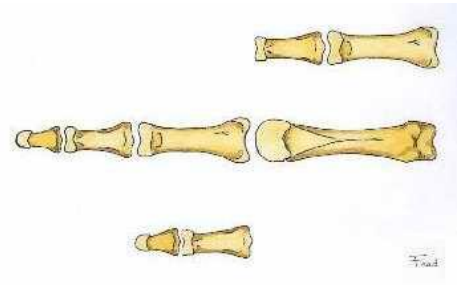


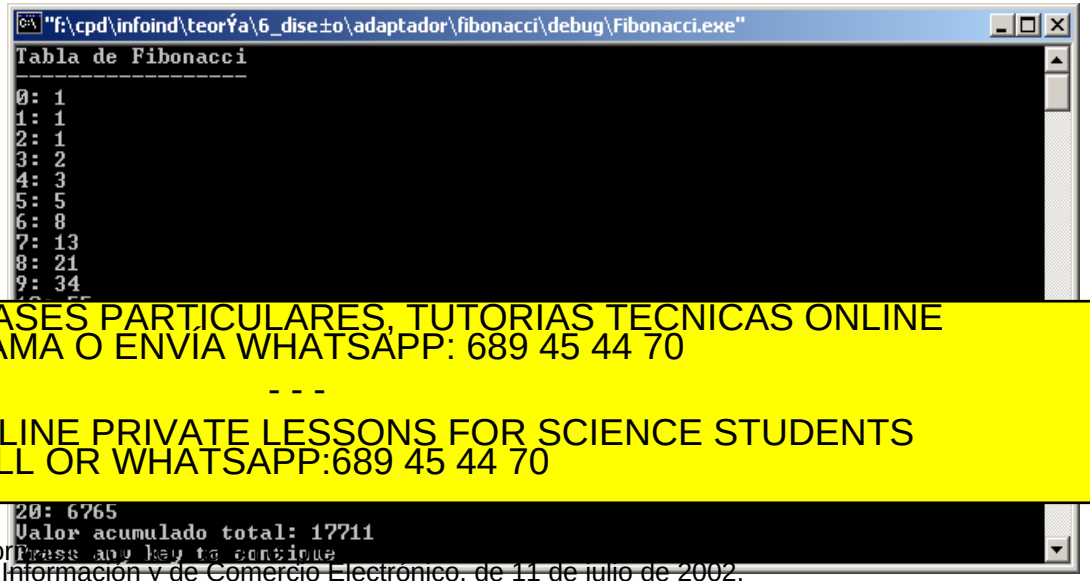
Ejemplo de Fibonacci











Ejemplo Adaptador

Un generador de la serie de Fibonacci ha sido dado. Adaptarlo para emplear los algoritmos dados por las STL.

```
#ifndef FIBONACCIGENERATOR_H
#define FIBONACCIGENERATOR_H
class FibonacciGenerator {
  int n;
  int val[2];
public:
  FibonacciGenerator() : n(0) { val[0] = val[1] = 1; }
  int operator()() {
    int result = n > 2 ? val[0] + val[1] : 1;
    ++n;
    val[0] = val[1];
    val[1] = result;
    return result;
```

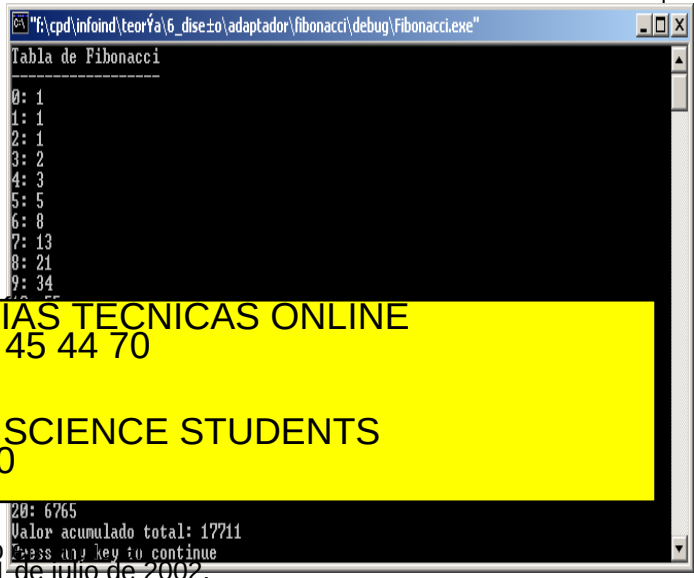
$$F_n = F_{n-1} + F_{n-2}$$

$$\{ \quad 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad ... \}$$

Ejemplo Adaptador

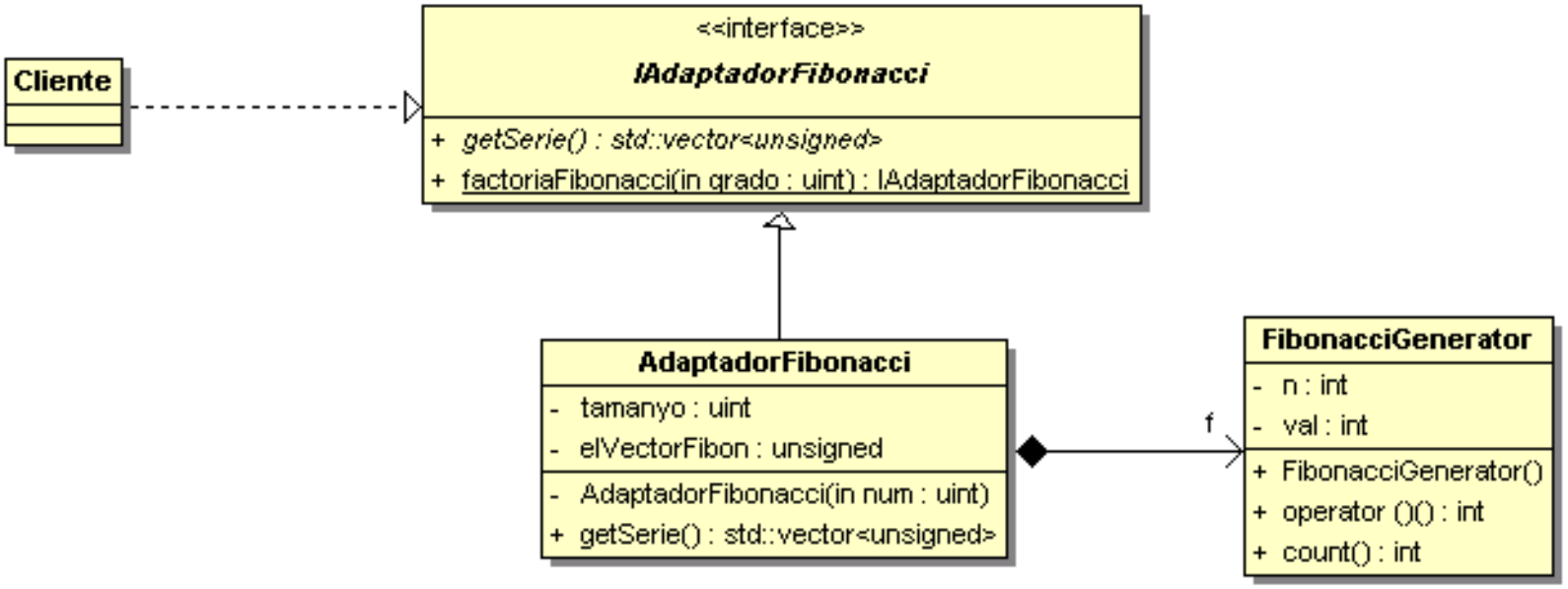
- Código de test

```
int main()
{
const unsigned numeroFibo = 20;
IAdaptadorFibonacci *pAdaptadorFibo
        =IAdaptadorFibonacci::factoriaFibonacci(numeroFibo);

cout << "Tabla de Fibonacci" <<endl;
cout << "------------------" <<endl;

for_each(pAdaptadorFibo->getSerie().begin(),pAdaptadorFibo->getSerie().end(),
          imprimir);
  //Ver GRASP "No hable con extraños" y el código maduro

return 0;

}
void imprimir(unsigned numFibo)
{
        static unsigned indice;
```



$F_n = F_{n-1} + F_{n-2}$ {1 1 1 2 3 5 8 13 ...}

Ejemplo Adaptador



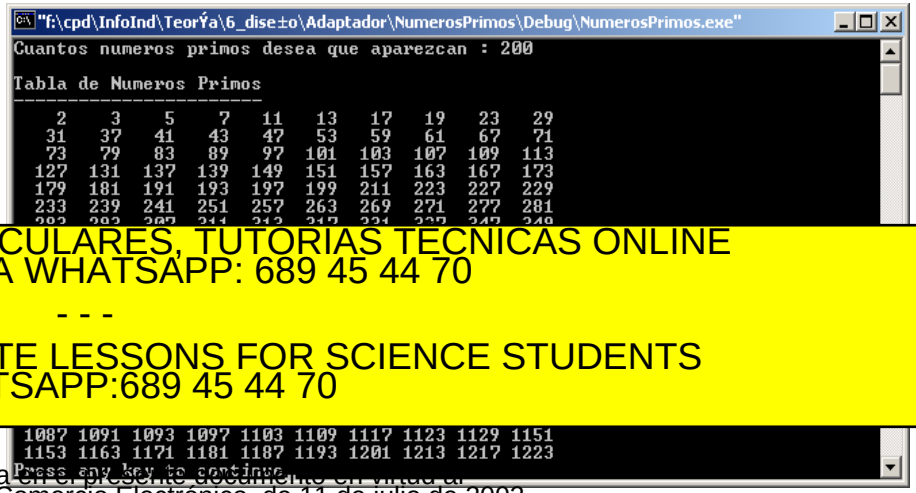
Ejemplo Adaptador

```
#include <vector>
#include "FibonacciGenerator.h"

class IAdaptadorFibonacci
{
public:
virtual std::vector<unsigned> & getSerie() = 0;
          static IAdaptadorFibonacci
          *factoriaFibonacci(unsigned grado);
};

class AdaptadorFibonacci: public IAdaptadorFibonacci
{
          FibonacciGenerator f;
          unsigned tamanyo;
          std::vector<unsigned> elVectorFibon;
          friend class IAdaptadorFibonacci;
          AdaptadorFibonacci(unsigned num):
          tamanyo(num) {
          for (unsigned i=0;i<=tamanyo;i++)
            elVectorFibon.push_back(f());
           }
public:
            virtual std::vector<unsigned> & getSerie()
```

Problema 3: Tabla de números primos

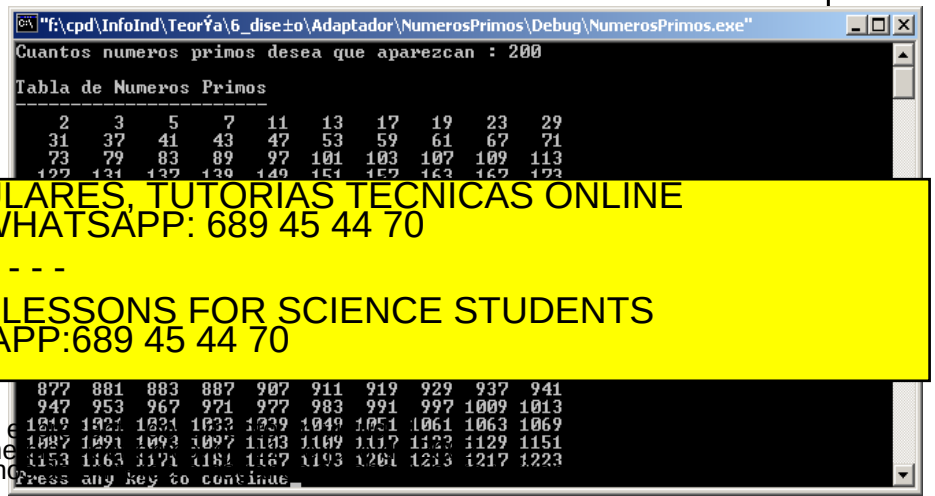
En el cuadro se entrega el código sobre un generador de números primos. Se trata de diseñar un componente tal que el cliente le entregue el número límite de números primos, *n*, y el servidor retorne con un vector que contenga los *n* primeros números primos. En la figura se presenta el resultado del cliente. Se pide:

1. Realizar ingeniería inversa sobre el generador de números primos.
2. Obtener el diagrama de clase de diseño, DCD, así como el diagrama de secuencia del componente servidor.
3. Implementación del cliente en C++.
4. Implementación de las clases en C++ del componente servidor.

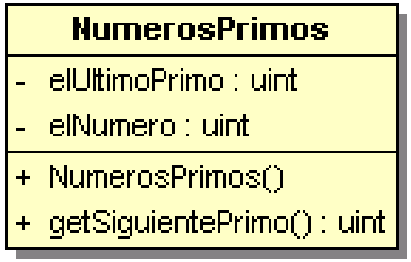


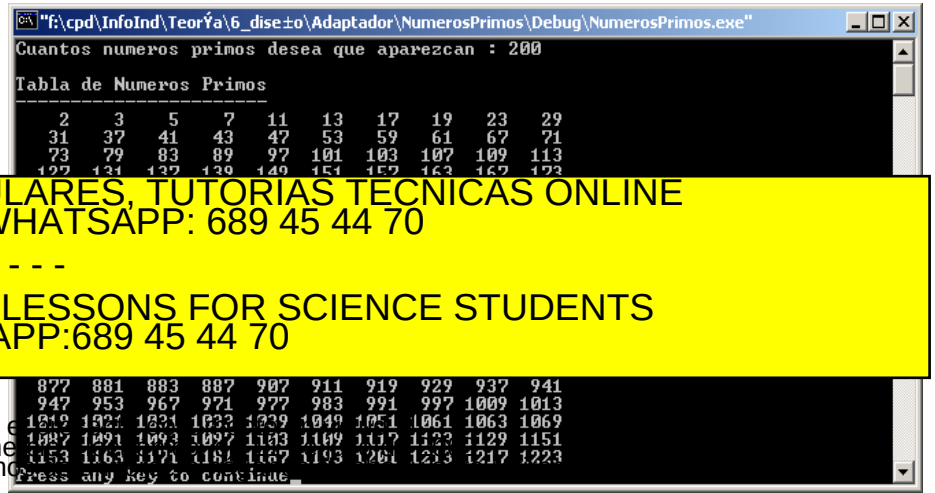
Problema 3: Tabla de números primos

```
#ifndef NUMEROSPRIMOS_H
#define NUMEROSPRIMOS_H

class NumerosPrimos {
  unsigned elUltimoPrimo;
  unsigned elNumero;
public:
  NumerosPrimos() : elUltimoPrimo(1) { elNumero = elUltimoPrimo+1;}
  unsigned getSiguientePrimo()
  {
        do{
           for(unsigned divisor = 2;elNumero % divisor !=  0; divisor++) ;
            if (divisor == elNumero)
              elUltimoPrimo = elNumero;
            else
              elNumero++;
        } while ( elUltimoPrimo != elNumero );
        elNumero++;
        return (elUltimoPrimo);
  }

};
#endif
```

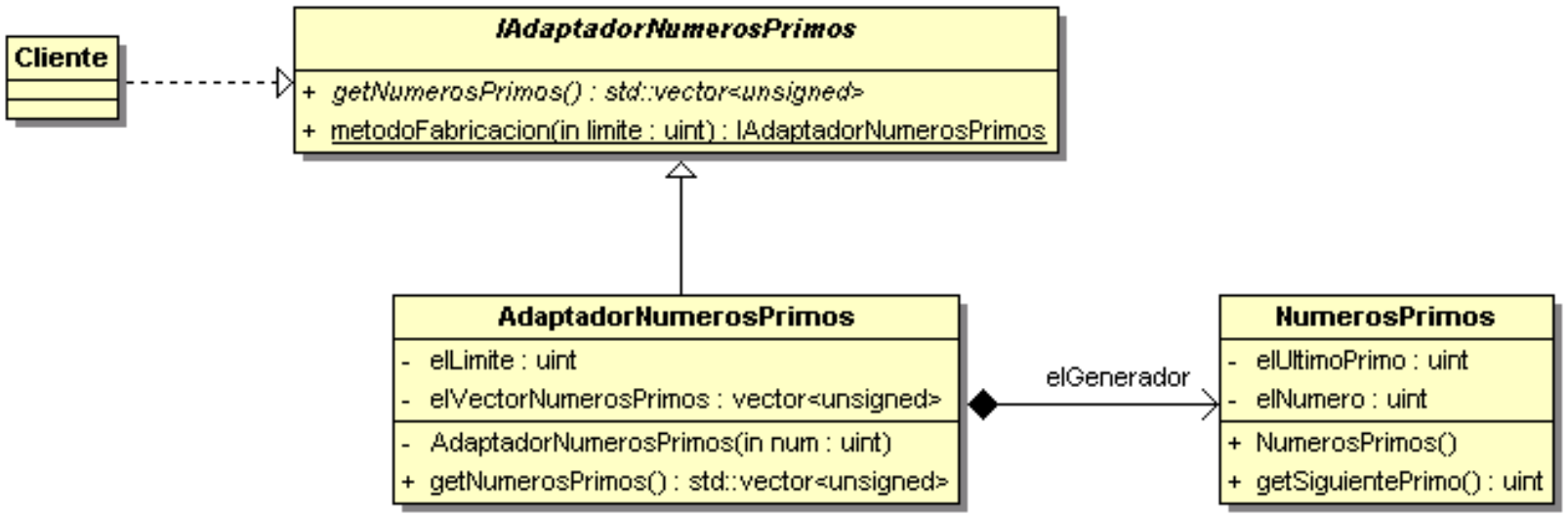


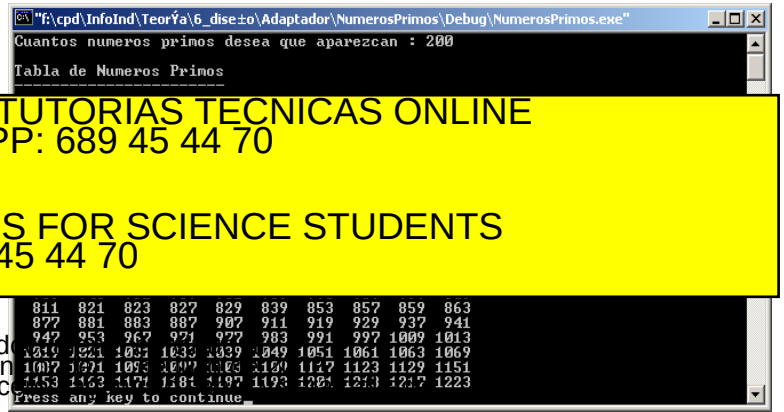
Problema 3: Tabla de números primos



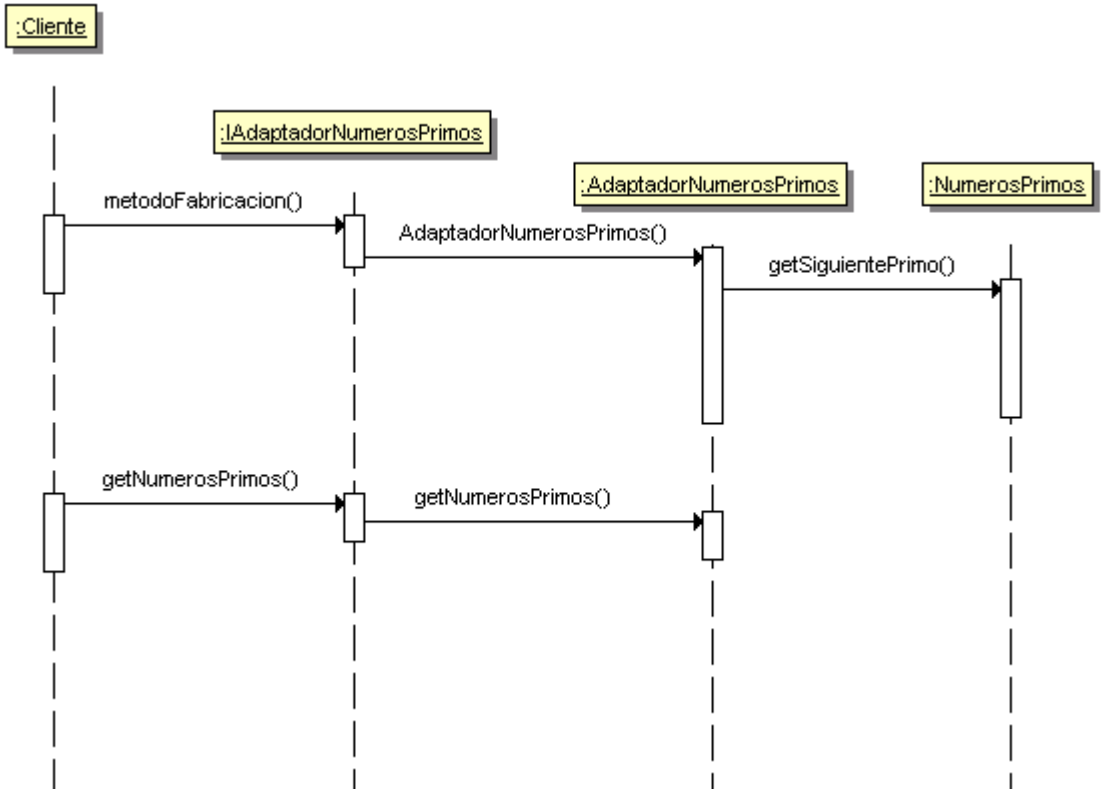


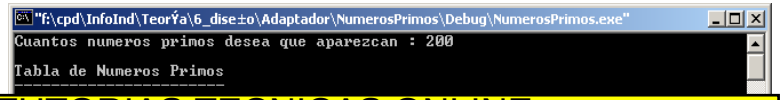
Problema 3: Tabla de números primos





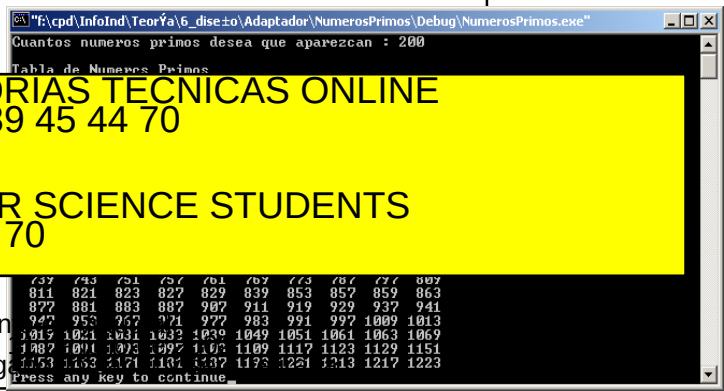
Problema 3: Tabla de números primos





Problema 3: Tabla de números primos

```
#include <iostream>
#include "AdaptadorNumerosPrimos.h"
#include <numeric>
#include <algorithm>

IAdaptadorNumerosPrimos* IAdaptadorNumerosPrimos::metodoFabricacion(unsigned limite)
{
        return (new AdaptadorNumerosPrimos(limite));

}
using namespace std;
void imprimir(unsigned);

int main()
{
  unsigned elLimiteNumerosPrimos;
  cout<<"Cuantos numeros primos desea que aparezcan : ";
  cin >> elLimiteNumerosPrimos;

  IAdaptadorNumerosPrimos *pAdaptadorNumPrimos =
         IAdaptadorNumerosPrimos::metodoFabricacion(elLimiteNumerosPrimos);

  cout << endl <<"Tabla de Numeros Primos" <<endl;
  cout << "-----------------------" <<endl;

  for_each(pAdaptadorNumPrimos->getNumerosPrimos().begin(),
              pAdaptadorNumPrimos->getNumerosPrimos().end(),imprimir);
  delete pAdaptadorNumPrimos;
  return 0;
}
```

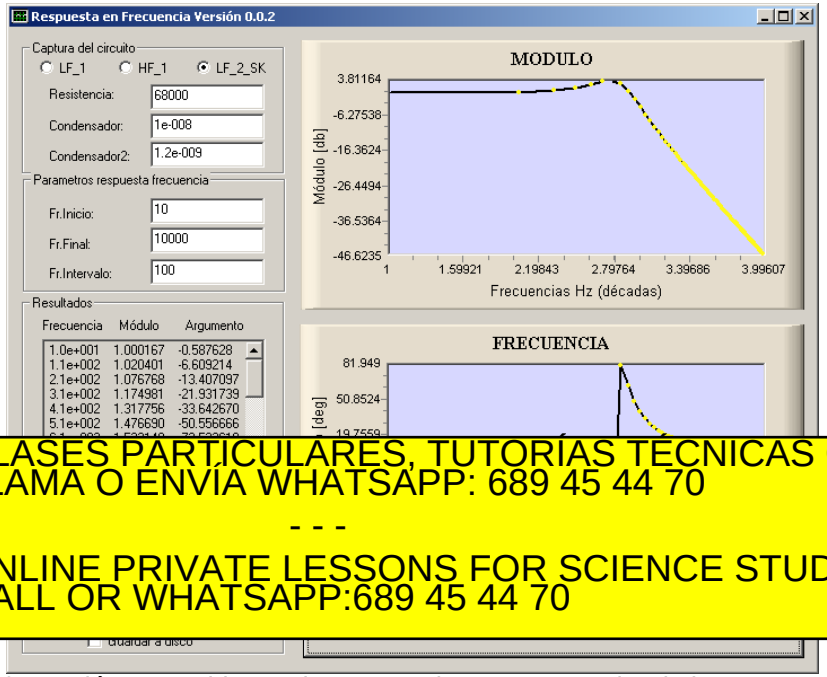


Problema 3: Tabla de números primos

```
#include <vector>
#include "NumerosPrimos.h"

class IAdaptadorNumerosPrimos
{
public:
    virtual std::vector<unsigned> & getNumerosPrimos() = 0;
    static IAdaptadorNumerosPrimos *metodoFabricacion(unsigned);
};

class AdaptadorNumerosPrimos: public IAdaptadorNumerosPrimos
{
    NumerosPrimos elGenerador;
    unsigned elLimite;
    std::vector<unsigned> elVectorNumerosPrimos;
    friend class IAdaptadorNumerosPrimos;
    AdaptadorNumerosPrimos(unsigned num): elLimite(num)
            {
          for (unsigned i=1;i<=elLimite;i++)
                    elVectorNumerosPrimos.push_back

          (elGenerador.getSiguientePrimo());
            }
public:
```

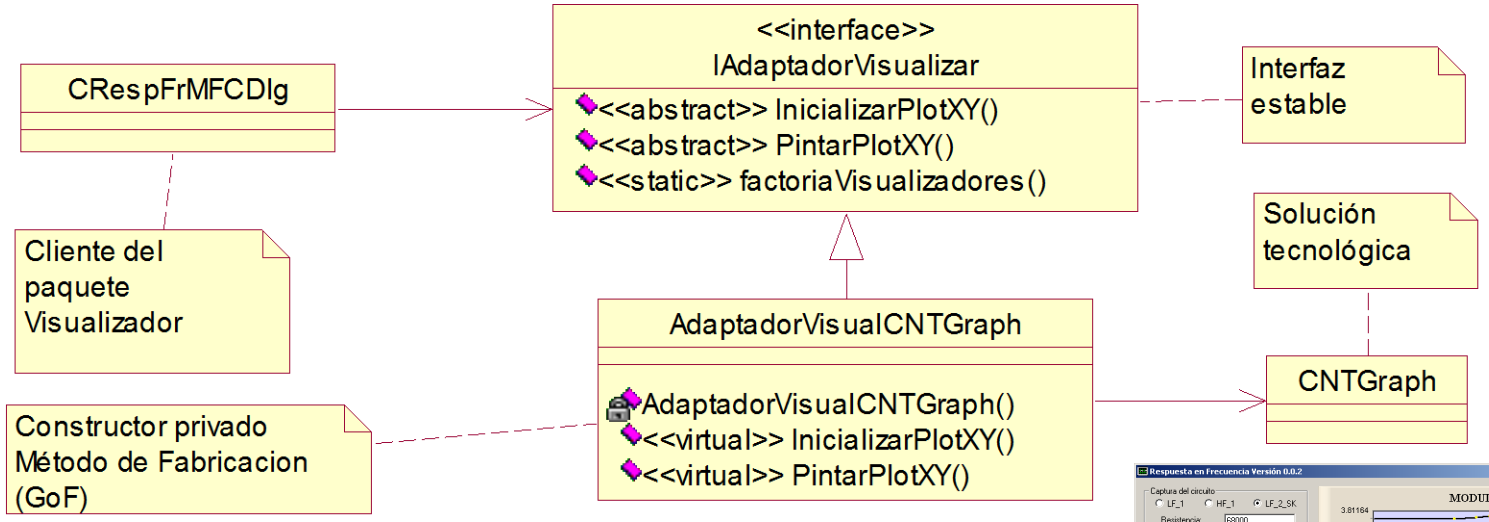
Ejemplo 6.13

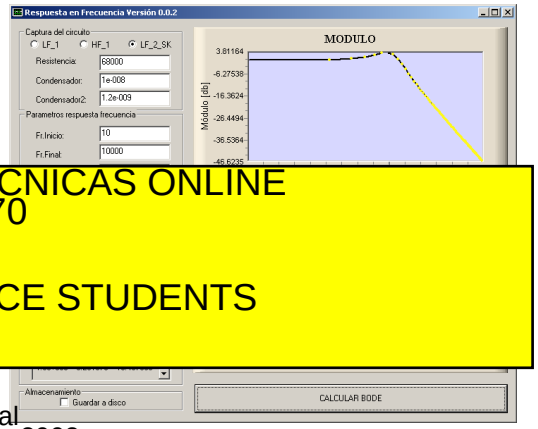
La aplicación de Respuesta en Frecuencia no depende sólo del algoritmo de calcular el módulo y argumento de un filtro, sino también de su visualización en un diagrama de Bode. Realizar un diseño para el paquete de representación gráfica.



Ejemplo 6.13

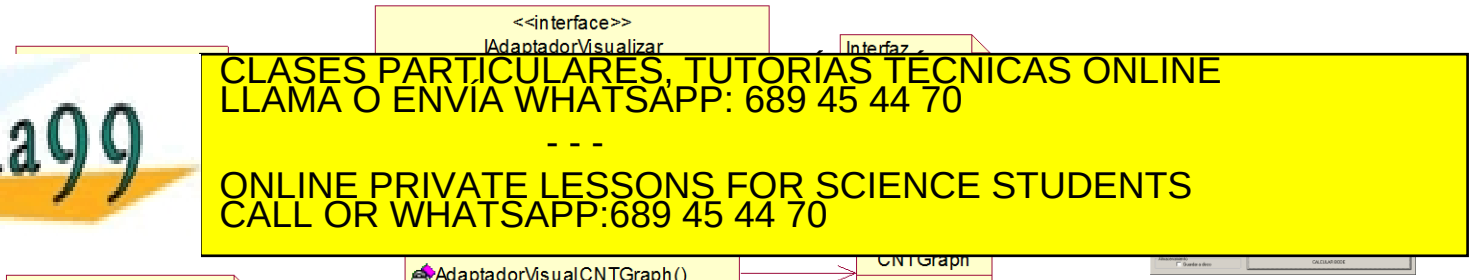
- Solución *NTGraph*. Es un punto caliente (dependencia tecnológica, prestaciones futuras)
- Patrón VP y Adaptador GoF





Ejemplo 6.13

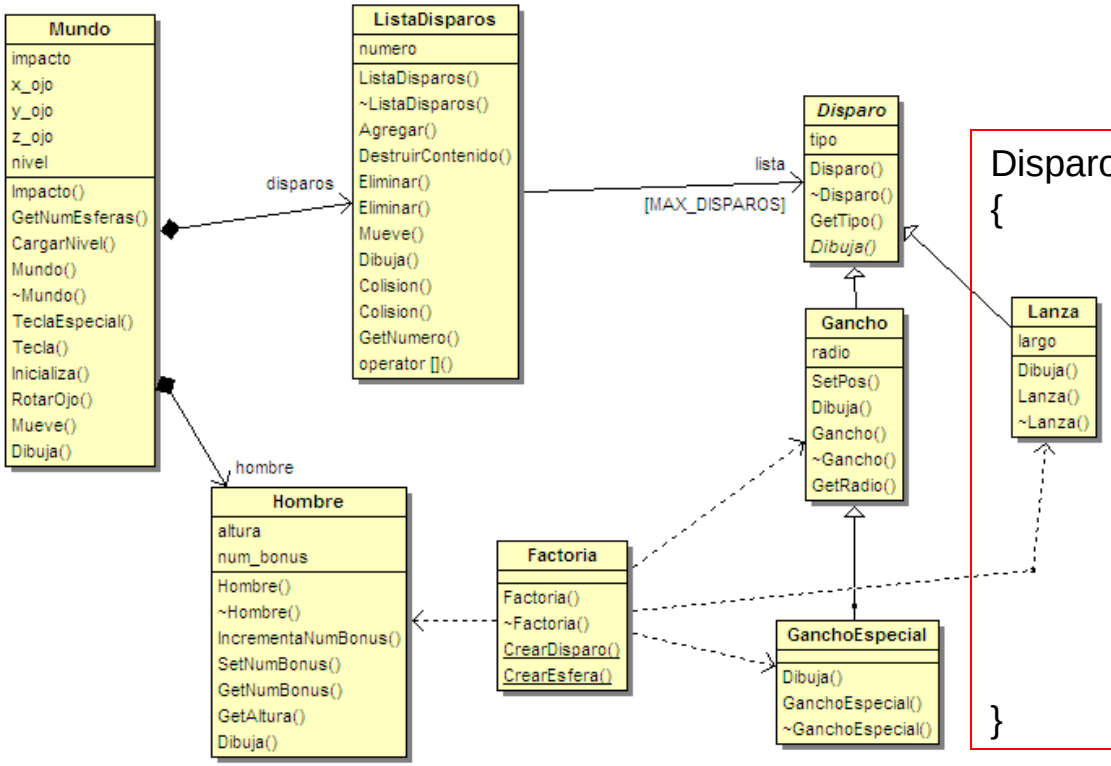
```
#include "../../ntgraph.h"
//Tipos de visualizadores
enum PlataformaVisual{NTGRAPH} ;

class IAdaptadorVisualizar
{
public:
          virtual void InicializarPlotXY(void) = 0;
          virtual void PintarPlotXY(float,float,float, double *)= 0;
          //Factoria de Visualizadores
          static IAdaptadorVisualizar *factoriaVisualizadores(enum
                    PlataformaVisual,CNTGraph *p1 = NULL);

};
class AdaptadorVisualCNTGraph : public IAdaptadorVisualizar
{
          CNTGraph *graph;
          AdaptadorVisualCNTGraph(CNTGraph *gr): graph(gr){}
          friend class IAdaptadorVisualizar;
public:
          virtual void InicializarPlotXY(void);
          virtual void PintarPlotXY(float,float,float, double *);
};
```



Factoría (GRASP)

- *Problema*: ¿Quién debe ser responsable de la creación de los objetos cuando existen consideraciones especiales, como una lógica de creación compleja, el deseo de separar las responsabilidades de la creación para manejar la cohesión, etc.?
- *Solución*: Crear un objeto de Fabricación Pura denominado Factoría que maneje la creación.

Ejemplo: Juego del Pang




```
Disparo* Factoria::CrearDisparo(Hombre h)
{
        Disparo* d=0;

        if(h.GetNumBonus()==1)
                d=new GanchoEspecial;
        else if(h.GetNumBonus()>=2)
                d=new Lanza;
        else
                d=new Gancho;

        Vector2D pos=h.GetPos();
        d->SetPos(pos.x,pos.y);

        return d;
}
```

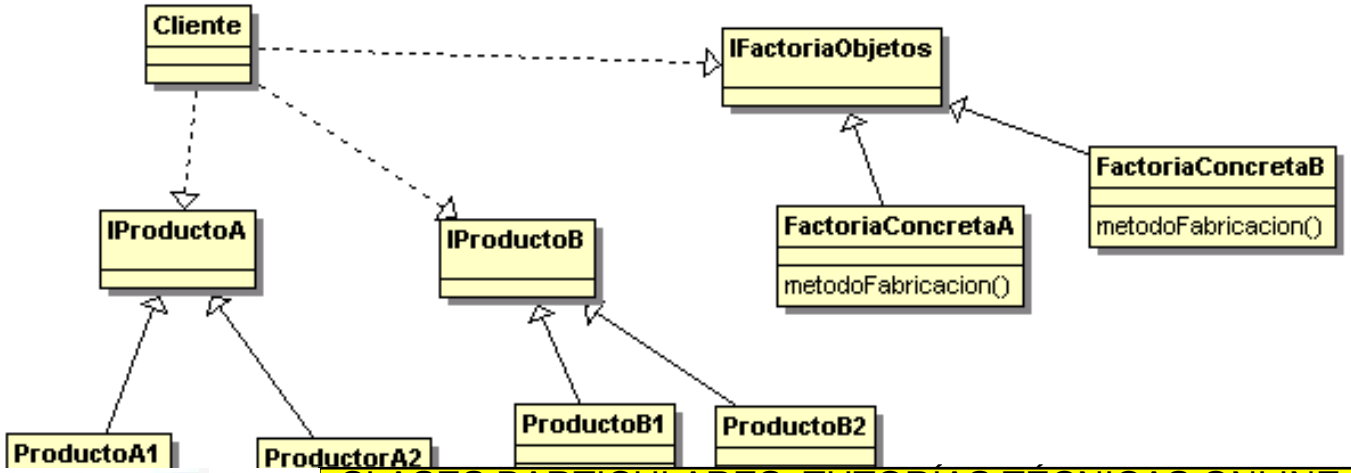
```
void Mundo::Tecla(unsigned char key)
{
  switch(key) {
                                    break;
  }
}
```

Factoría (GRASP)

- Cuando se hace VP ¿ quién debe de tener la responsabilidad de crear la instancia adecuada?
  - El paquete 'no', excede de sus responsabilidades.
  - Es un decisión externa
- Factoría: lectura de fuente externa y cargar la instancia de la clase adecuada de forma dinámica.
- Ventajas de las Factorías
  - Separación de responsabilidades en la creación compleja en objetos de apoyo cohesivo.
  - Ocultan la lógica de creación potencialmente compleja
  - Permite introducir estrategias para mejorar el rendimiento de la

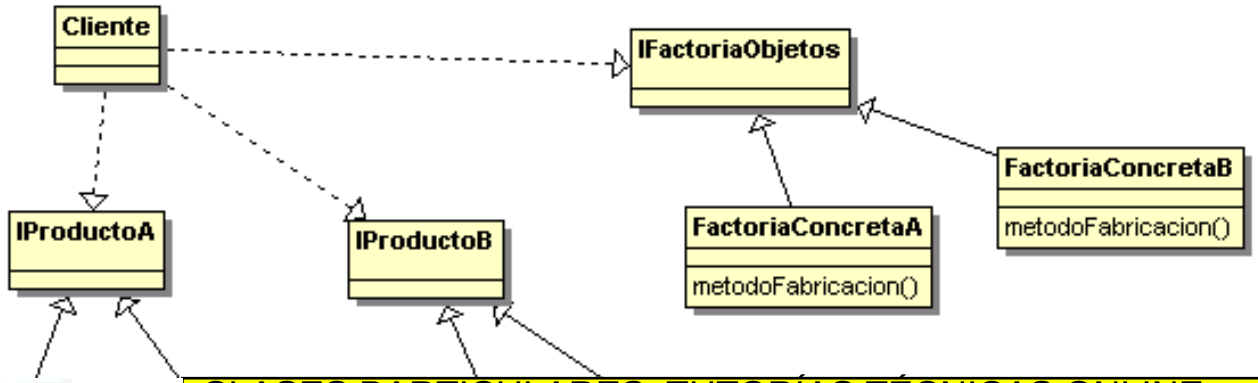
Factoría Abstracta (GoF)

- Se emplea cuando:
  - un sistema debe ser independiente de cómo se crean, componen y representan sus productos.
  - un sistema debe ser configurado como una familia de productos entre varios.
  - quiere proporcionar una biblioteca de clases de productos y sólo quiere revelar sus interfaces, no sus implementaciones.



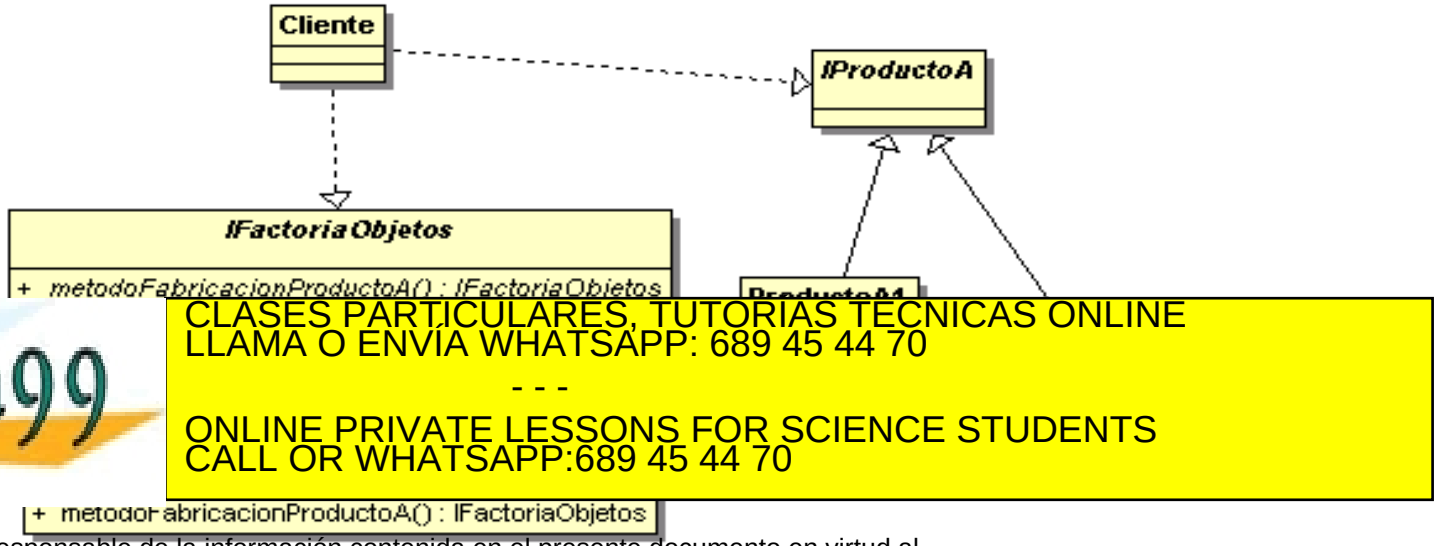
Factoría Abstracta (GoF)

- Los roles desempeñados son:
  - Cliente: sólo usa interfaces declarados por las clases de Fabricación Abstracta y Productos Abstractos
  - Producto Abstracto: declara una interfaz para un tipo de objeto (p.ej. IClaseA)
  - Producto Concreto: define un objeto producto para que sea creado por la Factoría correspondiente. Implementa la interfaz de Producto Abstracto.
  - Factoría Abstracta: declara una interfaz para operaciones que crean objetos productos abstractos
  - Factoría Concreta: implementa las operaciones para crear objetos producto concretos

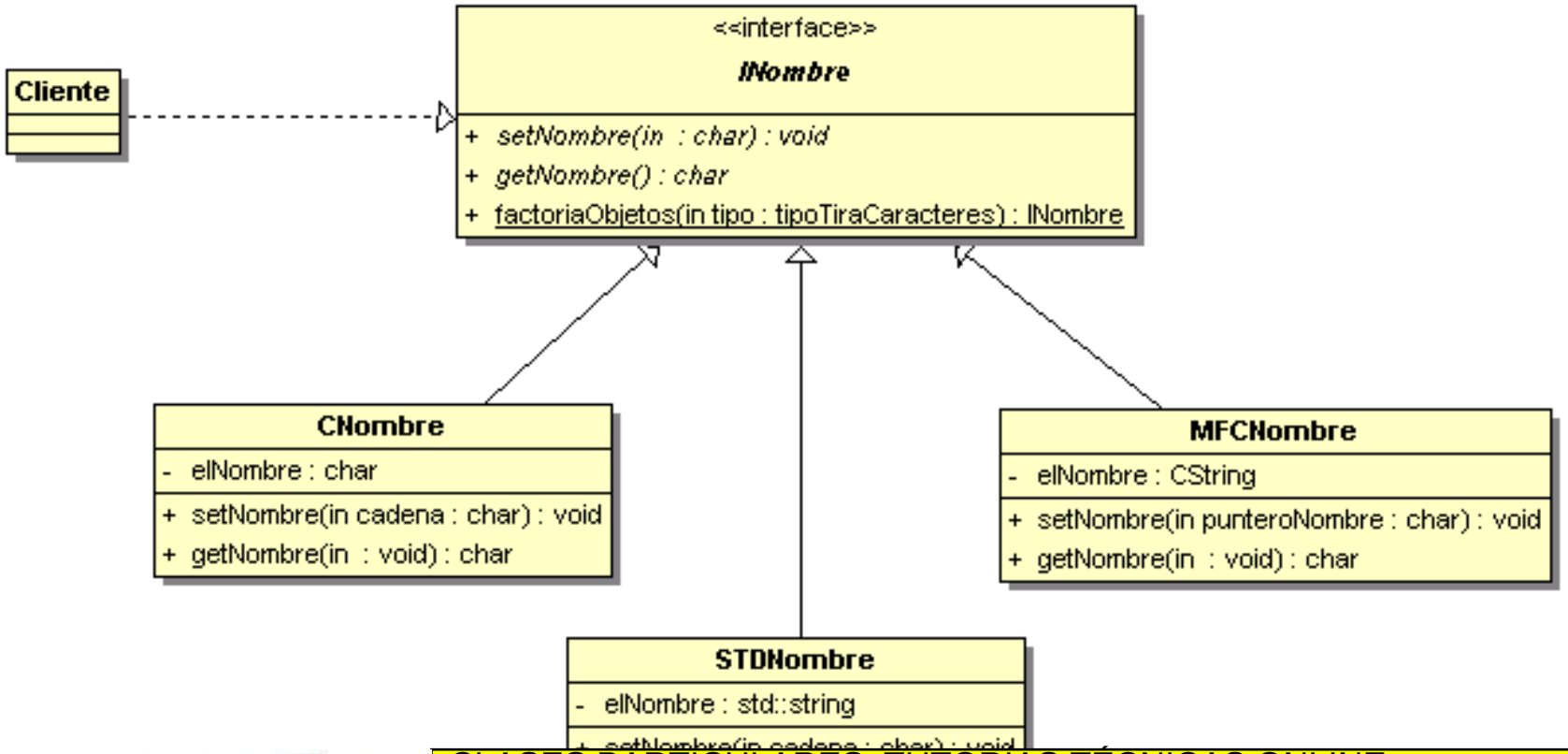


Método de Fabricación (GoF)

- Se define una interfaz de factoría para crear los objetos, pero se deja que sean las subclases quienes decidan qué clase instanciar
- Los roles que se desempeñan en este patrón son:
  - Producto: Interfaz de los objetos que crea el método de fabricación
  - ProductoConcreto: Realización de la interfaz Producto
  - Factoría: Declara el servicio de MétodoFabricación que retorna un objeto de tipo producto abstracto
  - FactoríaConcreto: redefine el método de fabricación para devolver una instancia de un ProductoConcreto



Ejemplo de interfaz visto capítulo 4



Ejemplo de interfaz

```
#ifndef _INOMBRE_INC_
#define _INOMBRE_INC_

enum Plataforma{ESTANDAR_STL, ESTILO_C, CADENA_MFC};

class INombre {
public:
  virtual void setNombre (const char *) = 0;
  virtual const char * getNombre () = 0;
  //Factoria de objetos
  static INombre *factoriaObjetos
      (enum  Plataforma);
};
#endif
```

```
#ifndef _INC_CNOMBRE_
#define _INC_CNOMBRE_

#include <string>
#include "INombre.h"

class CNombre : public INombre
{
public:
 virtual void setNombre(const char *cadena)
```

```
};
```

```
#ifndef _INC_STDNOMBRE_
#define _INC_STDNOMBRE_

#include <string>
#include "INombre.h"

class STDNombre : public INombre
{
public:
 virtual void setNombre(const char *cadena)
```

```
};
#endif
```

Ejemplo de interfaz y método de fabricación

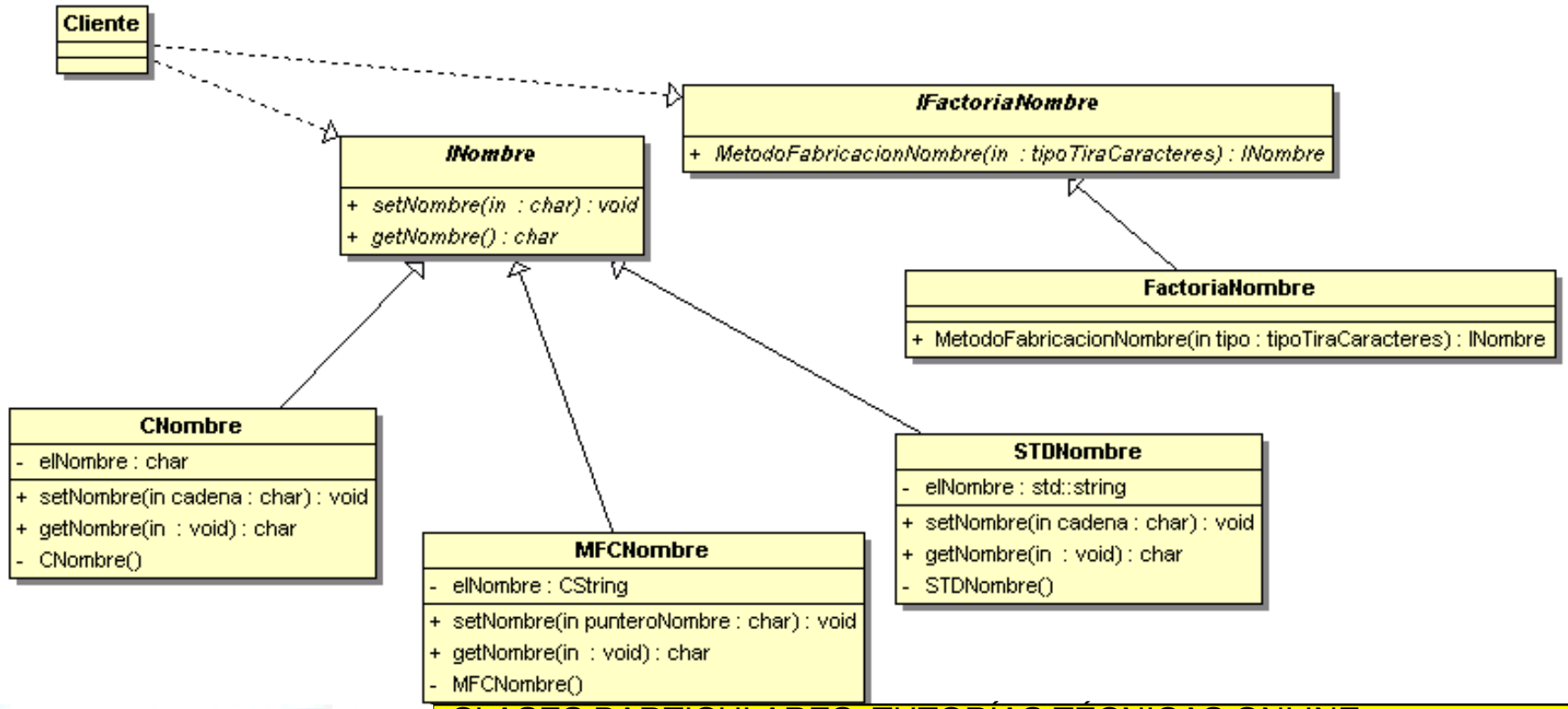
```
#include <iostream>
#include "../includes/STDNombre.h"
#include "../includes/CNombre.h"
#include "../includes/MFCNombre.h"

//Método único para producir los objetos nombres
INombre* INombre::factoriaObjetos(enum Plataforma tipo)
{
  if(tipo == ESTANDAR_STL) return new STDNombre;
  else if(tipo == ESTILO_C) return new CNombre;
  else if(tipo == CADENA_MFC) return new MFCNombre;
  else return NULL;

}
using namespace std;
int main ( void )
{
            INombre *pNombre1 = INombre::factoriaObjetos(ESTANDAR_STL);
            INombre *pNombre2 = INombre::factoriaObjetos(ESTILO_C);
            INombre *pNombre3 = INombre::factoriaObjetos(CADENA_MFC);

            pNombre1->setNombre("Manolo Gonzalez");
            pNombre2->setNombre("Pedro Lopez");
            pNombre3->setNombre("Ana Rodriguez");

}
```

Ejemplo de Factoría y Método de Fabricación

Realizar una factoría de objetos de Nombres. El cliente utilizará la clase abstracta *INombre* y las clases concretas serán empleando *std::string*, *CString* y en estilo C

```
#include <iostream>
#include "../includes/STDNombre.h"
#include "../includes/CNombre.h"
#include "../includes/MFCNombre.h"

using namespace std;

int main ( void )
{
      IFactoriaNombre *pFactoria = new (FactoriaNombre);
      INombre *pNombre1 = pFactoria->MetodoFabricacionNombre (ESTANDAR_STL);
      INombre *pNombre2 = pFactoria->MetodoFabricacionNombre (ESTILO_C);
      INombre *pNombre3 = pFactoria->MetodoFabricacionNombre (CADENA_MFC);

      pNombre1->setNombre("Manolo Gonzalez");
      pNombre2->setNombre("Pedro Lopez");
      pNombre3->setNombre("Ana Rodriguez");

      cout << pNombre1->getNombre() << endl;
```

Ejemplo de Método de Fabricación



Ejemplo de Método de Fabricación

```
enum Plataforma{ESTANDAR_STL, ESTILO_C,
CADENA_MFC};

class INombre {
public:
  virtual void setNombre (const char *) = 0;
  virtual const char * getNombre () = 0;
};
```

```
#include <string>
#include "INombre.h"
class FactoriaNombre;
class STDNombre : public INombre
{
public:
 virtual void setNombre(const char *cadena)
           { elNombre = cadena; }
  virtual const char * getNombre (void)
           { return (elNombre.c_str());}
private:
          std::string elNombre;
          STDNombre () {}
```

```
#include "STDNombre.h"
#include "CNombre.h"
#include "MFCNombre.h"

class IFactoriaNombre
{
public:
 virtual INombre* MetodoFabricacionNombre
(enum Plataforma) = 0;

};

class FactoriaNombre: public IFactoriaNombre
{
public:
 virtual INombre* MetodoFabricacionNombre
(enum Plataforma tipo)
 {
 if(tipo==ESTANDAR_STL)return new STDNombre;
 else if(tipo==ESTILO_C) return new CNombre;
 else if(tipo==CADENA_MFC) return new
          MFCNombre;
 else return NULL;
```

Problema impresoras

Se tienen las siguientes clases pertenecientes a unas librerías C++, desarrolladas por los fabricantes de impresoras, y que sirven para imprimir un archivo cualquiera en una impresora de un determinado fabricante.

```
class EpsonPrinterDriver
{
public:
    bool Print(char filename[]);
};
class HPControladorImpresora
{
public:
    //este método devuelve 1 en caso de éxito y -1 en caso de error
    int ImprimeFichero(char* nombre_fichero);
};
```

Problema impresoras

Por tanto, LAS DOS CLASES ANTERIORES NO SE PUEDEN MODIFICAR, TOCAR O CAMBIAR. Se desea hacer un programa que permita al usuario teclear el nombre de un archivo, el nombre de la impresora de destino, y que el programa utilice automáticamente la clase de la librería correspondiente. La función main de ese programa (incompleta) seria:

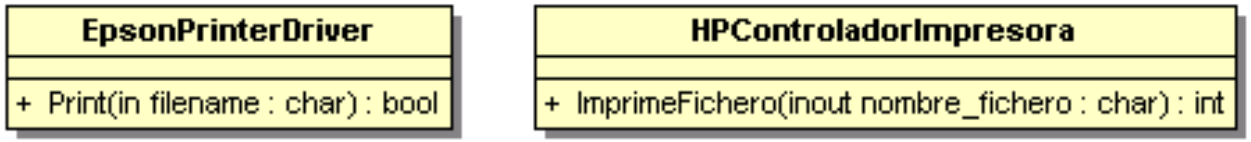
```
int main()
{
    char fichero[255],nombre_impresora[255];
    cout<<"Introduzca en nombre de fichero: "; cin>>fichero;
    cout<<"Introduzca nombre impresora: HP o EPSON: "; cin>>nombre_impresora;

    Impresora* impresora=NULL;
    //AQUÍ COMPLETAR CODIGO DE CREACION DE LA IMPRESORA ADECUADA
    // en funcion de "nombre_impresora"
    if(impresora==NULL)
    {
      cout<<"Impresora no existe"<<endl;
      return -1;
    }
    if(impresora->Imprime(fichero))
      cout<<"Impresion correcta"<<endl;
```

Problemas impresoras

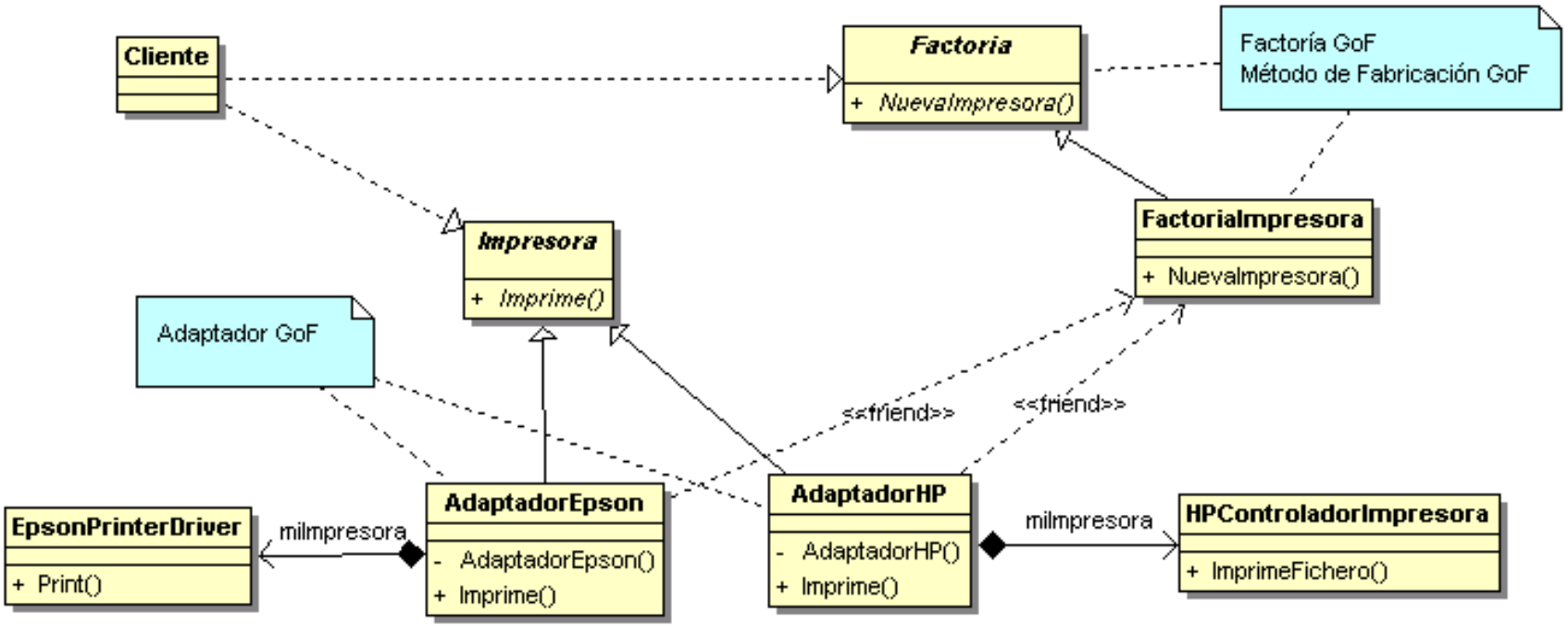
SE PIDE:

1. Diagrama UML de las clases existente
2. Diagrama de Clases de Diseño (DCD) de la solución.
3. Explicación (breve, con notas en el anterior diagrama) de los patrones usados
4. Implementación C++ de la solución propuesta. (no olvidar completar el main)

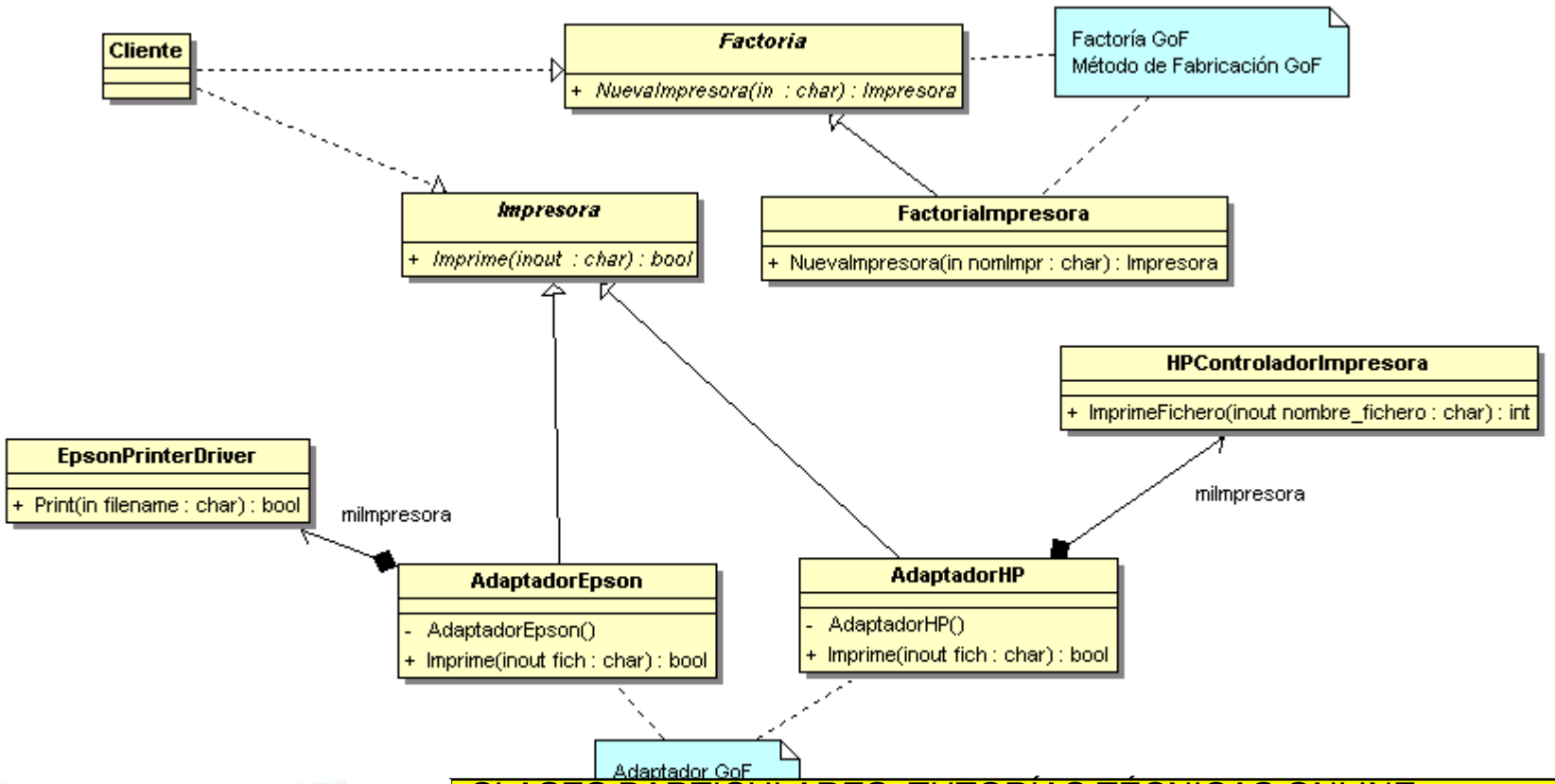
Problema impresoras (DCD)



Problema impresoras (DCD)



Problema impresoras (DCD)



Problema impresoras (código test)

```
int main()
{
        char fichero[255],nombre_impresora[255];

        cout<<"Introduzca en nombre de fichero: "; cin>>fichero;
        cout<<"Introduzca impresora: HP o EPSON: ";cin>>nombre_impresora;

        Impresora* impresora=NULL;
        Factoria* factoria = new FactoriaImpresora;

        impresora= factoria->NuevaImpresora(nombre_impresora);
        if(impresora==NULL){
                cout<<"Impresora no existe"<<endl;
                return -1;
        }
        if(impresora->Imprime(fichero))
                cout<<"Impresion correcta"<<endl;

        return 0;
}
```

Problema impresoras

```
class Factoria{
public:
        virtual Impresora * NuevaImpresora(const char *) = 0;
};

class FactoriaImpresora: public Factoria
{
public:
        Impresora * NuevaImpresora(const char *nomImpr)
        {
                if(!strcmp(nomImpr,"HP"))   return new AdaptadorHP;
                if(!strcmp(nomImpr,"EPSON"))return new AdaptadorEpson;
                return NULL;

        }
```

Problemas impresora

```
class Impresora{
public:
        virtual bool Imprime(char *)=0;
};

class AdaptadorEpson : public Impresora{
        friend class FactoriaImpresora;
        AdaptadorEpson() {}
        EpsonPrinterDriver miImpresora;
public:
        virtual bool Imprime(char *fich){
                return(miImpresora.Print(fich));
        }
};

class AdaptadorHP : public Impresora{
        friend class FactoriaImpresora;
        AdaptadorHP() {}
        HPControladorImpresora miImpresora;
                                                    }
};
```

Ejemplo

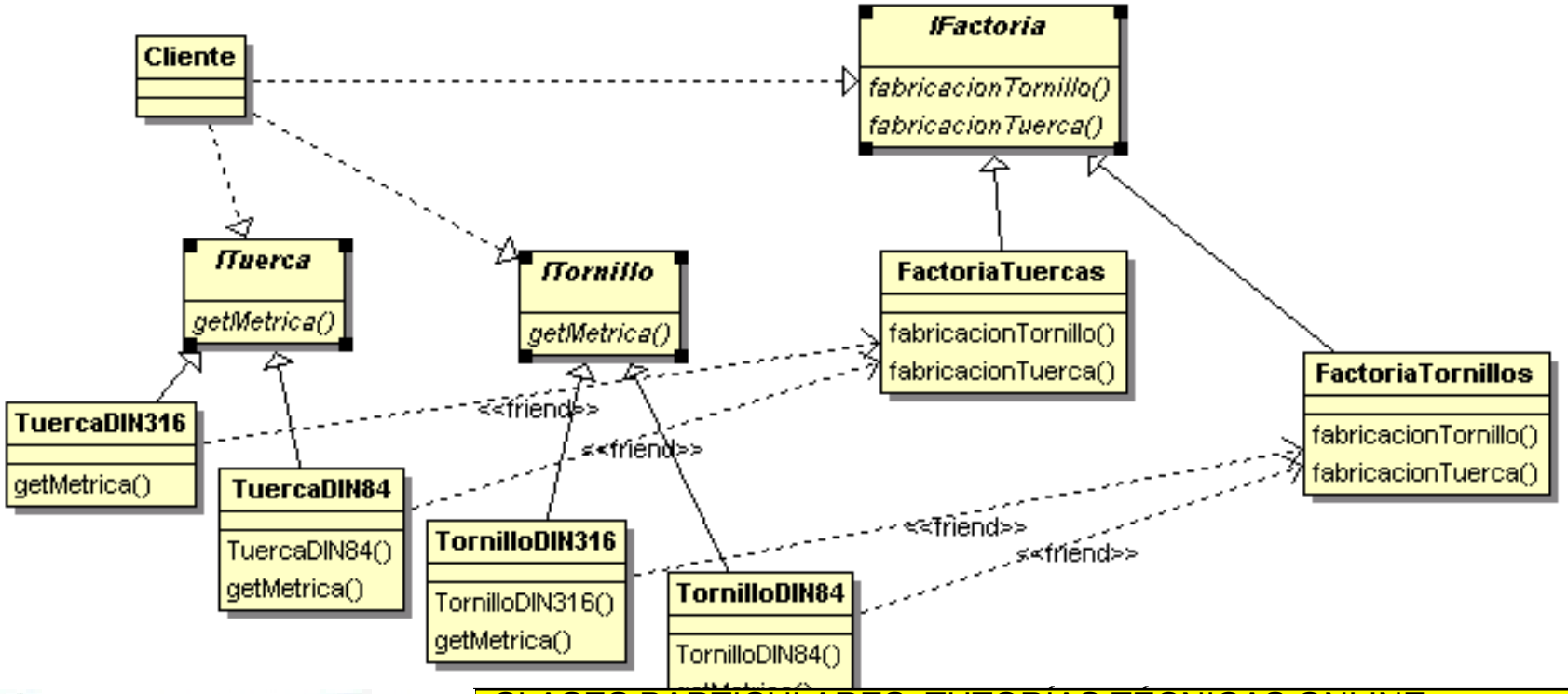
Se pretende simular el evento de sacar de una bolsa un tornillo y una tuerca y saber si se pueden ensamblar. La bolsa puede contener tornillos y tuercas de diferentes métricas. Hágase para el caso concreto de elementos DIN84 y DIN316.


```
"C:\cpd\InfoInd\TeorÝa\6_dise±o\Factorias\Ferreteria\Debug\...
Simulacion de sacar un tornillo y una tuerca de forma aleatoria de una bolsa
La bolsa contiene tornillos y tuercas de metrica DIN84 y DIN316
Pulsar c o C para sacar tornillo y tuerca
c
Ensamblaje correcto: metrica DIN84
Pulsar c o C para sacar tornillo y tuerca
c
Ensamblaje correcto: metrica DIN316
Pulsar c o C para sacar tornillo y tuerca
c
Ensamblaje incorrecto
Pulsar c o C para sacar tornillo y tuerca
c
Ensamblaje correcto: metrica DIN316
Pulsar c o C para sacar tornillo y tuerca
```

Ejemplo: código de test

```
#include "IFactoria.h"
#include <iostream>
#include <stdlib.h>

int main()
{
 IFactoria *pFactoriaTornillos = new FactoriaTornillos;
 IFactoria *pFactoriaTuercas   = new FactoriaTuercas;
 std::cout<<"Simulacion de sacar tornillo y tuerca de forma aleatoria" <<std::endl;
 std::cout<<"La bolsa contiene tornillos y tuercas DIN84 y DIN316"<<std::endl;
 std::cout<<"Pulsar c o C para sacar tornillo y tuerca"<<std::endl;
 char opcion;       std::cin>> opcion;
 while(opcion == 'c' || opcion == 'C') {
     ITornillo *pTornillo =
          pFactoriaTornillos->fabricacionTornillo(rand() % 2 == 1 ? DIN84 : DIN316);
     ITuerca   *pTuerca =
          pFactoriaTuercas->fabricacionTuerca(rand() % 2 == 1 ? DIN84 : DIN316);
     if(pTornillo->getMetrica() == pTuerca->getMetrica()){
            char *mensaje = pTuerca->getMetrica() == DIN84 ? "DIN84" : "DIN316";
            std::cout<<"Ensamblaje correcto: metrica " << mensaje <<std::endl;
     }else
            std::cout<<"Ensamblaje incorrecto" << std::endl;

          std::cout<<"Pulsar c o C para sacar tornillo y tuerca"<<std::endl;
```

Ejemplo de Factorias Abstractas y métodos de fabricación



Ejemplo de Factorias Abstractas y métodos de fabricación

```
typedef enum{DIN84, DIN316} metrica;
class FactoriaTuercas;
class ITuerca
{
public:
   virtual metrica getMetrica() = 0;

};
class TuercaDIN84 : public ITuerca
{
   metrica laMetrica;
   friend class FactoriaTuercas;
   TuercaDIN84() {laMetrica = DIN84;}
public:
virtual metrica getMetrica() {return
laMetrica;}
};
class TuercaDIN316 : public ITuerca
{
   metrica laMetrica;
   friend class FactoriaTuercas;
   TuercaDIN316() {laMetrica = DIN316;}
public:
   virtual metrica getMetrica() {return
```

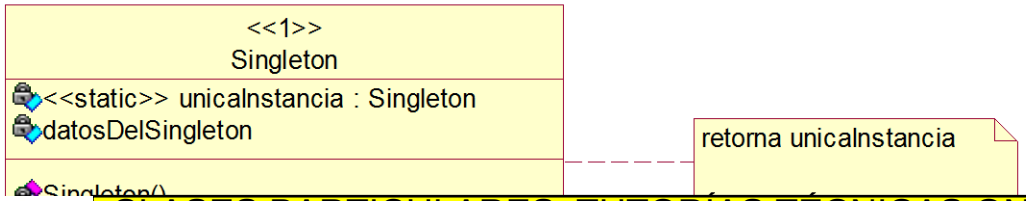
```
class FactoriaTornillos;
class ITornillo
{
public:
   virtual metrica getMetrica() = 0;
};
class TornilloDIN84 : public ITornillo
{
   metrica laMetrica;
   friend class FactoriaTornillos;
   TornilloDIN84() {laMetrica = DIN84;}
public:
virtual metrica getMetrica() {return laMetrica;}
};
class TornilloDIN316 : public ITornillo
{
  metrica laMetrica;
  friend class FactoriaTornillos;
  TornilloDIN316() {laMetrica = DIN316;}

public:
virtual metrica getMetrica() {return laMetrica;}
};
```

Ejemplo de Factorias Abstractas y métodos de fabricación

```
class IFactoria
{
public:
          virtual ITornillo * fabricacionTornillo(metrica) = 0;
          virtual ITuerca   * fabricacionTuerca(metrica) = 0;
};
class FactoriaTornillos : public IFactoria
{
public:
          virtual ITornillo * fabricacionTornillo(metrica laMetrica)
          {
                    if(laMetrica == DIN84) return new TornilloDIN84;
                    else if (laMetrica == DIN316) return new TornilloDIN316;
                    else return 0;
          }
          virtual ITuerca   * fabricacionTuerca(metrica laMetrica)
                    {return 0;}
};
class FactoriaTuercas: public IFactoria
{
public:
          virtual ITornillo * fabricacionTornillo(metrica laMetrica)
          {return 0;}

          virtual ITuerca   * fabricacionTuerca(metrica laMetrica)
                    {return new TuercaDIN84;}
};
```

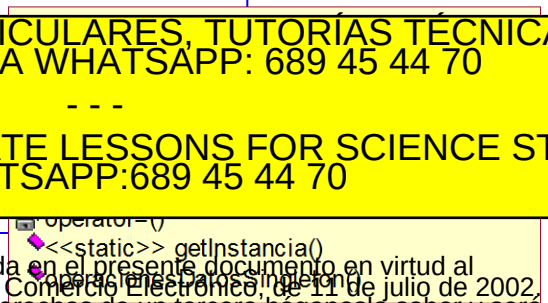
Singleton (GoF)

- *Problema*: ¿Cómo garantizar que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella? .
- Solución: Definir un método estático de la clase que devuelva el singleton.
- ¿quién crea a la Factoría?
  - Sólo se necesita una única instancia de Factoría y ésta puede ser llamada desde distintos lugares del código.
- Impresora, Convertidor A/D,...
- La clave del Singleton es evitar que el cliente tenga el control sobre la vida del objeto

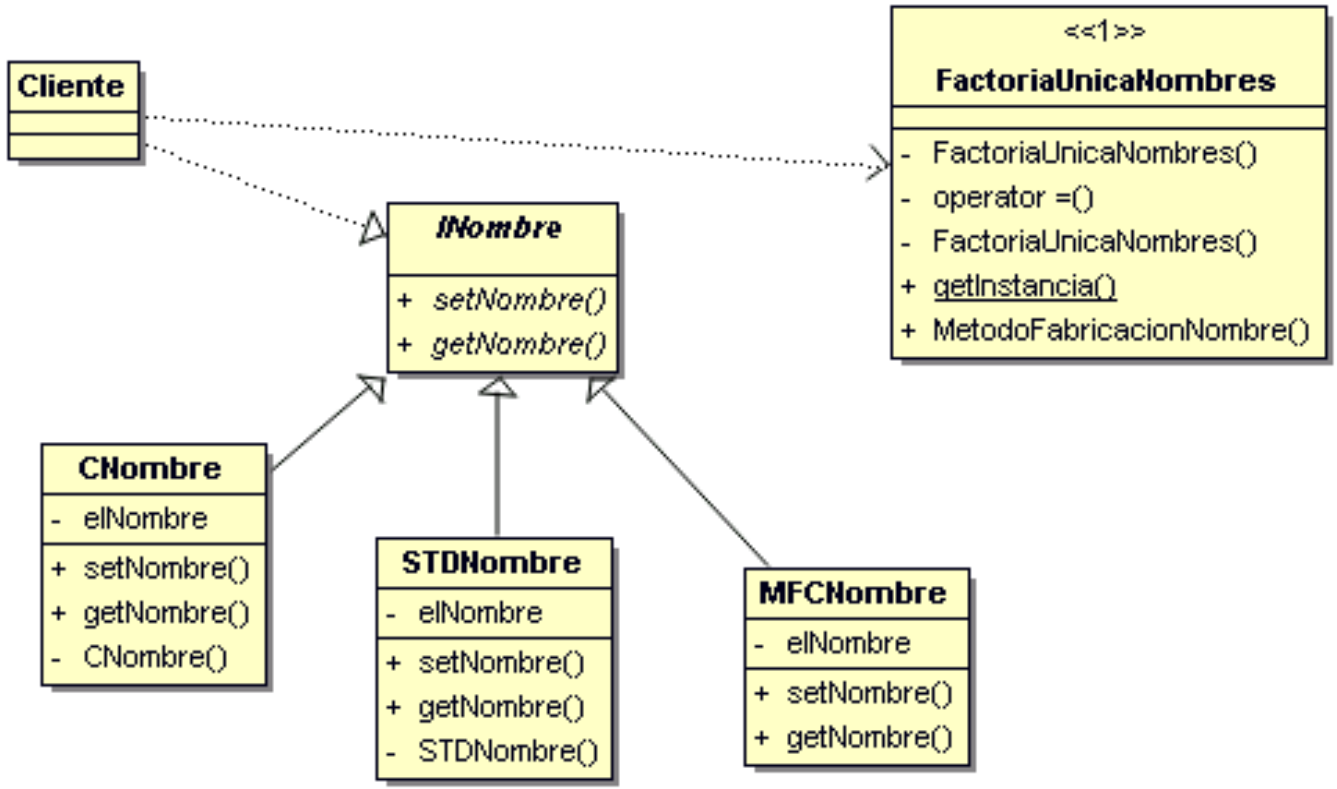


Ejemplo Singleton

```
#include <iostream>
using namespace std;

class Singleton {
  int i; //Dato por ejemplo
  Singleton(int x) : i(x) { }
  void operator=(Singleton&);// desactivar
  Singleton(const Singleton&);// desactivar
public:
  static Singleton& getInstancia() {
    static Singleton unicaInstancia(47);
         return unicaInstancia;  }
  int getValor() { return i; }
  void setValor(int x) { i = x; }
};

int main() {
  Singleton& s = Singleton::getInstancia();
  cout << s.getValor() << endl;
  Singleton& s2 = Singleton::getInstancia();
```



Ejemplo Factoría-Singleton



Ejemplo Factoría-Singleton

```
#include <iostream>
#include "../includes/INombre.h"
#include "../includes/FactoriaUnicaNombres.h"
using namespace std;
int main ( void )
{
          void otrosNombres (void);
          FactoriaUnicaNombres &laFactoria = FactoriaUnicaNombres::getInstancia();
          INombre *pNombre1 = laFactoria.MetodoFabricacionNombre (ESTANDAR_STL);
          INombre *pNombre2 = laFactoria.MetodoFabricacionNombre (ESTILO_C);

          pNombre1->setNombre("Manolo Gonzalez");
          pNombre2->setNombre("Pedro Lopez");
          cout << pNombre1->getNombre() << endl;
          cout << pNombre2->getNombre() << endl;
          delete pNombre1, pNombre2;
          otrosNombres();
          return 0;
}

void otrosNombres ( void )
{
          //Solo utiliza referencias abstractas
          FactoriaUnicaNombres &laMismaFactoria = FactoriaUnicaNombres::getInstancia();
```

Ejemplo Factoría-Singleton

```
#ifndef _IFACTORIA_INC_
#define _IFACTORIA_INC_

#include "STDNombre.h"
#include "CNombre.h"
#include "MFCNombre.h“

class FactoriaUnicaNombres
{
    FactoriaUnicaNombres(){};
    void operator=(FactoriaUnicaNombres&);  // Para desactivar
    FactoriaUnicaNombres(const FactoriaUnicaNombres&); // Para desactivar

public:
    static FactoriaUnicaNombres& getInstancia() {
            static FactoriaUnicaNombres unicaInstancia;
            return unicaInstancia;
    }

    INombre* MetodoFabricacionNombre (tipoTiraCaracteres tipo){
        if(tipo == ESTANDAR_STL) return new STDNombre;
                                                    }
#endif
```

Ejemplo Factoría-Singleton

```
typedef enum {ESTANDAR_STL, ESTILO_C, CADENA_MFC} tipoTiraCaracteres;

class INombre {
public:
        virtual void setNombre (const char *) = 0;
        virtual const char * getNombre () = 0;


};

class CNombre : public INombre
{
public:

        virtual void setNombre(const char *cadena)
        { strcpy (elNombre, cadena); }

        virtual const char * getNombre (void)
        { return (elNombre);}

private:
```

Ejercicio de examen

Para una prueba de concepto sobre la gestión de vehículos de un municipio se ha diseñado el siguiente código, al cual se ha añadido el resultado de la ejecución de esta versión. En todos los vehículos se registra la matrícula. Si el vehículo es un camión se anota el máximo de toneladas que puede cargar. Para el resto se almacena el número de pasajeros del automóvil. Se pide

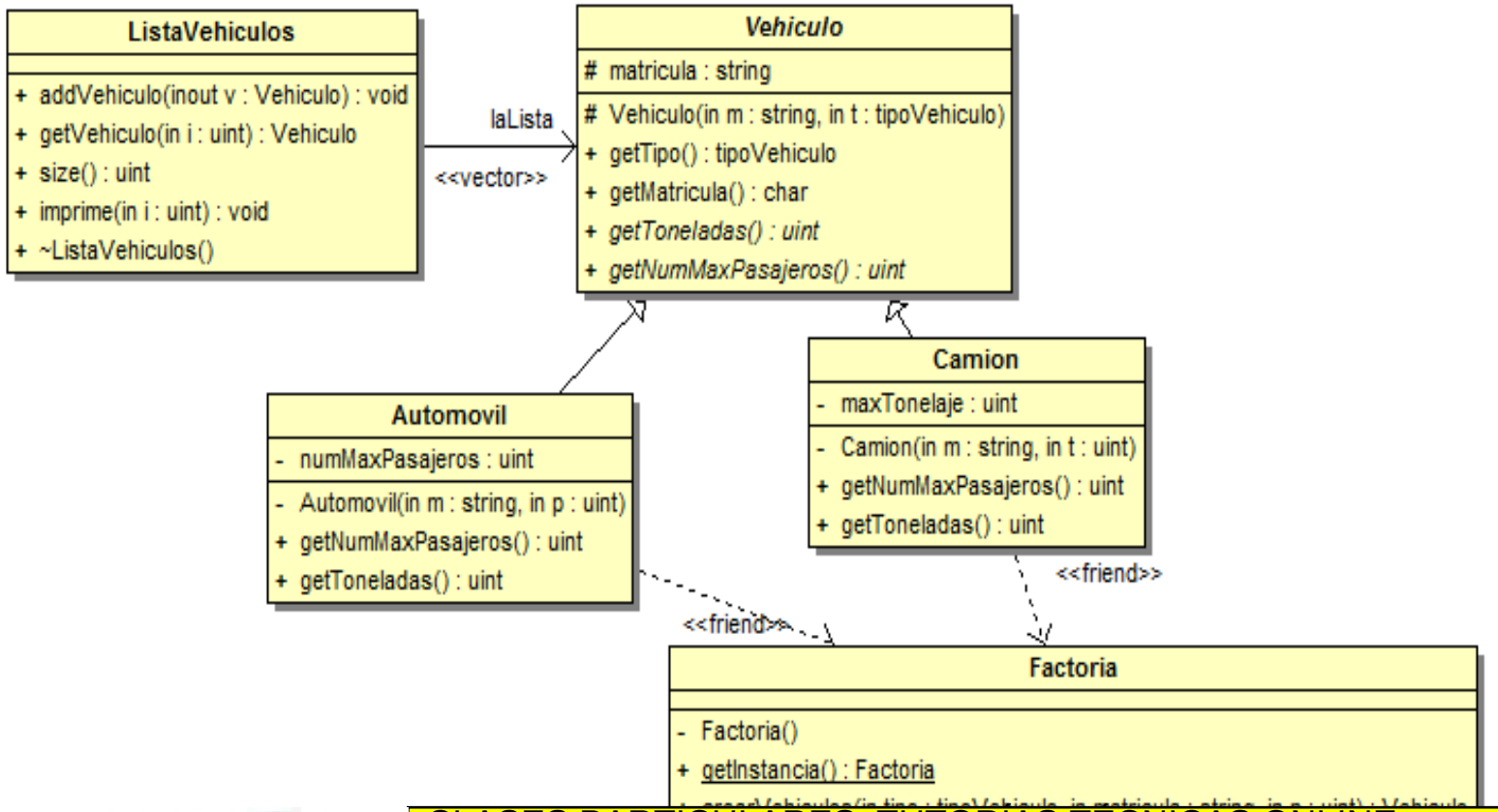
1. Diagrama de clase de diseño DCD en UML de las clases *Vehiculo*, *Automovil*, *Camion*, *ListaVehiculos* y *Factoria*. Indique los patrones empleados (5 puntos).
2. Implementación de estas clases en C++ (5 puntos).

Ejercicio de examen

```
#include <iostream>
#include <vector>
using namespace std;
typedef enum{AUTO,CAMION} tipoVehiculo;
class Vehiculo{…};
class Automovil{…};
class Camion{…};
class ListaVehiculos{…};
class Factoria{…};
void add_autos(ListaVehiculos &);
void add_camiones(ListaVehiculos &);

int main()
{
            ListaVehiculos lista;
            add_autos(lista);
            add_camiones(lista);
            for(unsigned i=0;i<lista.size();i++)
                        lista.imprime(i);

            return 0;
}

void add_autos(ListaVehiculos & lista)
{
 Factoria laFactoria = Factoria::getInstancia();
 lista.addVehiculo(laFactoria.crearVehiculos(AUTO,"1234 AAA", 5));
 lista.addVehiculo(laFactoria.crearVehiculos(AUTO,"5678 EEE", 7));
}
```

```
Matricula: 1234 AAA
Auto con numero maximo pasajeros: 5
Matricula: 5678 EEE
Auto con numero maximo pasajeros: 7
Matricula: 4321 BBB
Camion con maximo de toneladas: 20
Matricula: 8765 CCC
Camion con maximo de toneladas: 15
```

Ejercicio de examen



Ejercicio de examen

```
class Vehiculo{
protected:
          string matricula;
          tipoVehiculo tipo;
          Vehiculo(const string m,tipoVehiculo t): matricula(m),tipo(t){}
public:
          tipoVehiculo getTipo() {return tipo;}
          const char * getMatricula() {return matricula.c_str();}
          virtual unsigned getToneladas() = 0;
          virtual unsigned getNumMaxPasajeros() = 0;
};
class Automovil : public Vehiculo{
          friend class Factoria;
          unsigned numMaxPasajeros;
          Automovil(const string m,const unsigned p):Vehiculo(m,AUTO),numMaxPasajeros(p) {}
public:
          unsigned getNumMaxPasajeros() { return numMaxPasajeros; }
          unsigned getToneladas() { return 0; }
};
class Camion : public Vehiculo{
          friend class Factoria;
          unsigned maxTonelaje;
          Camion(const string m,const unsigned t):Vehiculo(m,CAMION),maxTonelaje(t) {}
public:
```

Ejercicio de examen

```
class ListaVehiculos{
          vector<Vehiculo *> laLista;
public:
          void addVehiculo(Vehiculo *v) {laLista.push_back(v);}
          Vehiculo * getVehiculo(unsigned i) { return laLista[i];}
          unsigned size() {return laLista.size();}
          void imprime(unsigned i) {
           cout<<"Matricula: "<< laLista[i]->getMatricula()<<endl;
           if(laLista[i]->getTipo()==AUTO)
             cout<<"Auto con numero maximo pasajeros: " <<
                    laLista[i]->getNumMaxPasajeros()<<endl;
           else
             cout<<"Camion con maximo de toneladas: " <<
                    laLista[i]->getToneladas()<<endl;
          }
          ~ListaVehiculos() { for(unsigned i=0; i<laLista.size();i++) delete laLista[i]; }
};
class Factoria
{
          Factoria(){}
public:
    static Factoria& getInstancia(){
            static Factoria unicaInstancia;
            return unicaInstancia;
                    }
          }
```

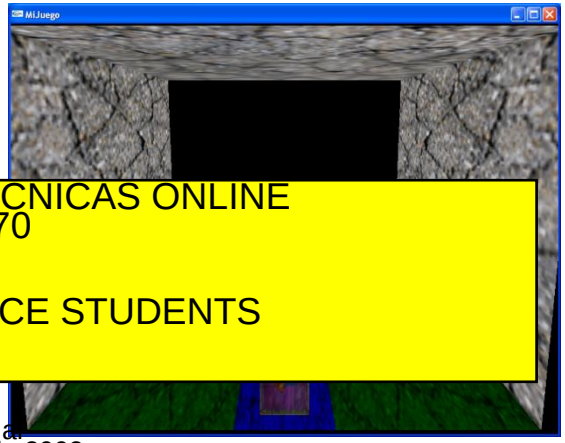
Problema 7

Realizar un juego de batalla los hombres contra los dragones. Los hombres lanzan cuchillos y los dragones bolas de fuego. Los dragones se mueven en el área izquierda de la pantalla y los hombres en el lado derecho. En mitad de la batalla aparecen paredes móviles que se desplazan en el centro de la pantalla. El número de luchadores puede ser variable y dinámico. Se pide:

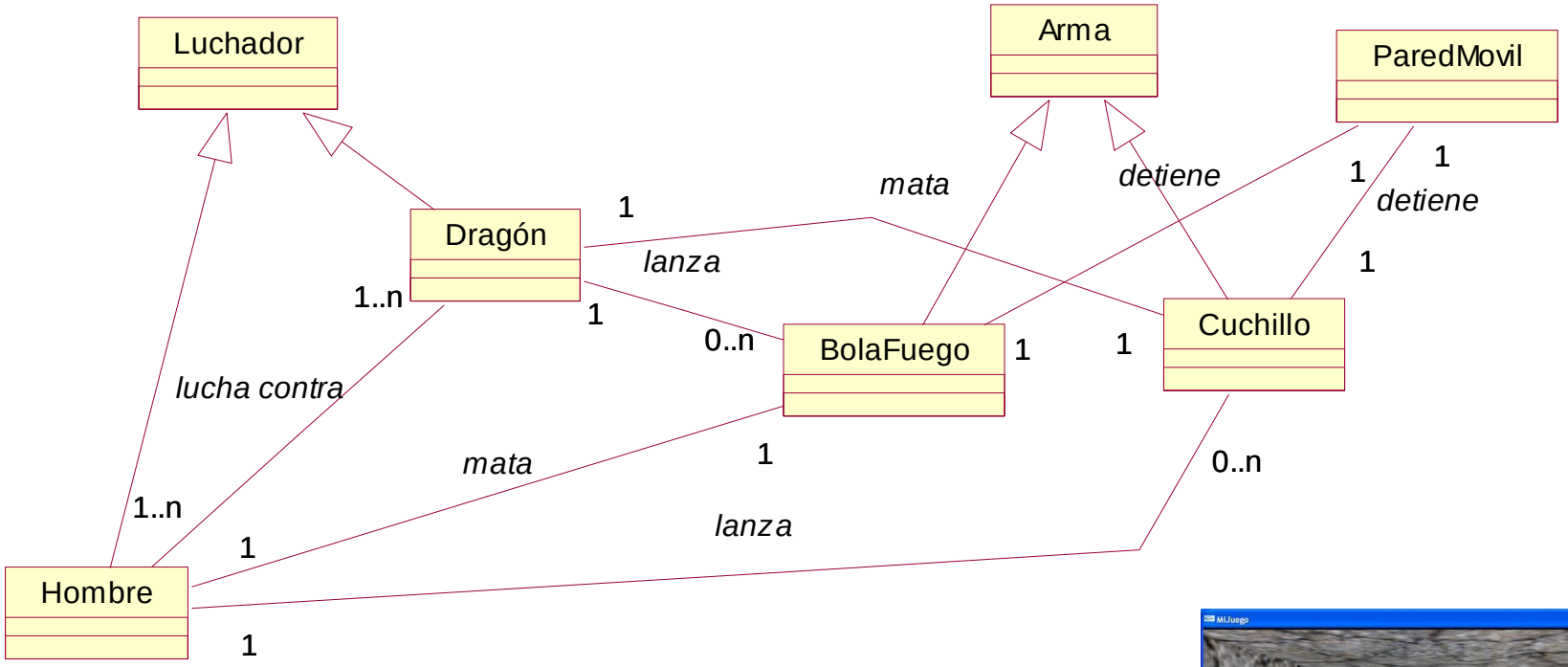
1. Jerarquía a dos niveles de las características principales.
2. Modelo del dominio.
3. Diagrama de clases de diseño.
4. Implementación en C++ de los ficheros de cabecera de las clases.

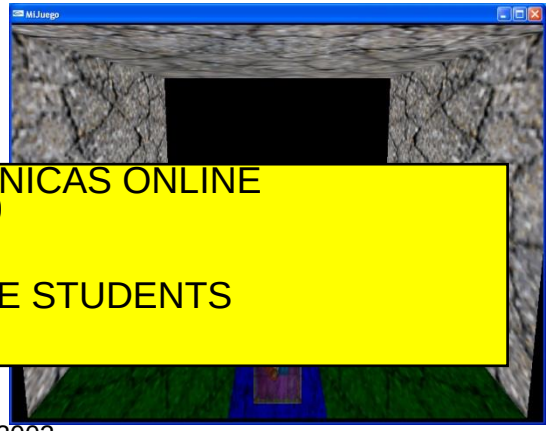
Problema 7

1. Video juego de los hombres que luchan contra los dragones
   - 1.1 Los hombres se mueven en un área restringida de la derecha.
   - 1.2 Los dragones se mueven en un área restringida de la izquierda.
   - 1.3 Los hombres lanzan cuchillos que se clavan en la pared o que matan al dragón o que pasan sin hacer daño.
   - 1.4 Los dragones lanzan bolas de fuego que no pueden atravesar las paredes y que si tocan a un hombre lo mata.
   - 1.5 Los dragones desaparecen de la pantalla al morir.
   - 1.6 Los hombres desaparecen de la pantalla al morir.

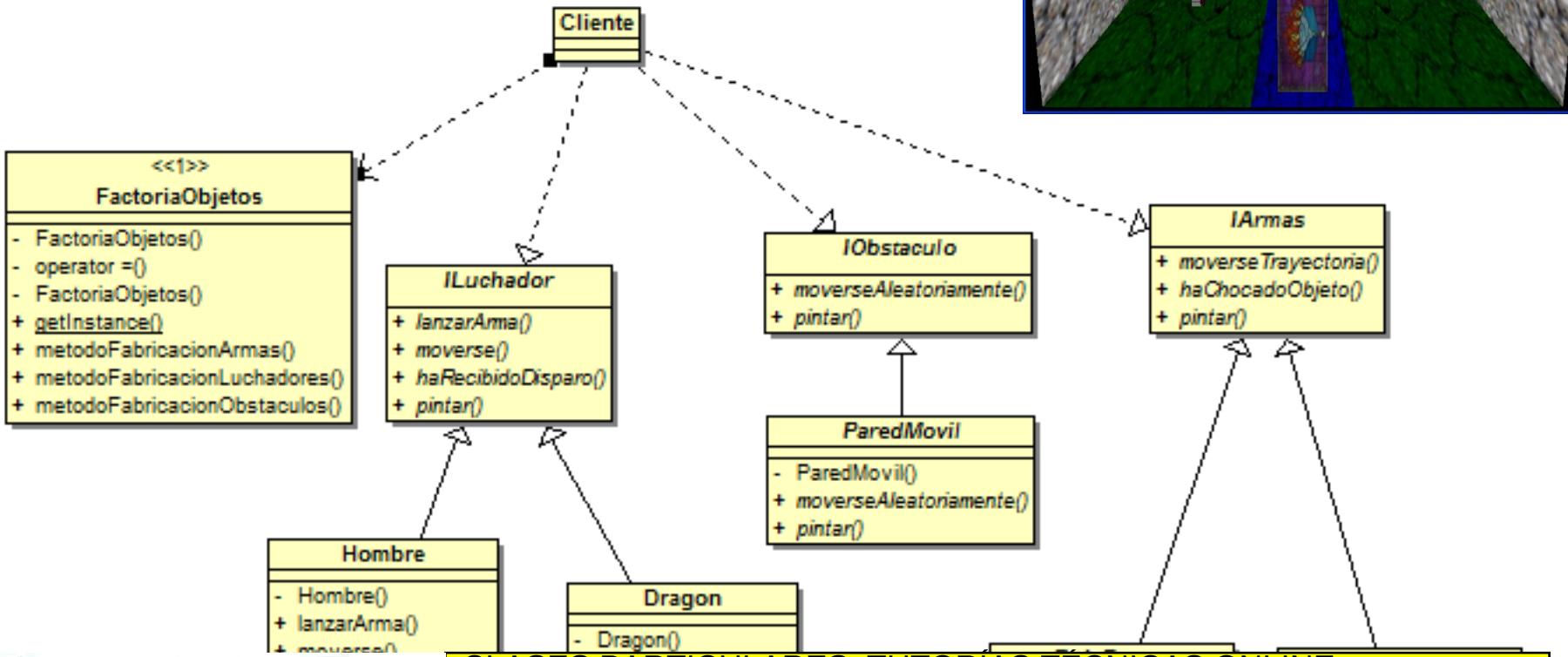


Problema 7





Ejemplo



Problema 7

```
#ifndef _LUCHADORES_INC_
#define _LUCHADORES_INC_

class FactoriaObjetos;

typedef enum{HOMBRE,DRAGON}
TipoLuchadores;

class ILuchador
{
public:
 virtual void lanzarArma() = 0;
 virtual void moverse() = 0;
 virtual bool haRecibidoDisparo() = 0;
 virtual void pintar() = 0;
};

class Hombre : public ILuchador
{
 friend class FactoriaObjetos;
 Hombre();
public:
 virtual void lanzarArma();
 virtual void moverse();
 virtual bool haRecibidoDisparo();
 virtual void pintar();
};

class Dragon : public ILuchador
{
 friend class FactoriaObjetos;
```

```
#ifndef _ARMAS_INC_
#define _ARMAS_INC_

class FactoriaObjetos;

typedef enum {BOLAFUEGO,CUCHILLO}
TipoArmas;

class IArmas
{
public:
virtual void moverseTrayectoria() = 0;
virtual bool haChocadoObjeto() = 0;
virtual void pintar() = 0;
};

class BolaFuego : public IArmas
{
friend class FactoriaObjetos;
BolaFuego();
 public:
virtual void moverseTrayectoria();
virtual bool haChocadoObjeto();
virtual void pintar();
};

class Cuchillo : public IArmas
{
friend class FactoriaObjetos;
Cuchillo();
public:
```

Problema 7

```
#ifndef _OBSTACULOS_INC_
#define _OBSTACULOS_INC_

class FactoriaObjetos;

typedef enum {PAREDMOVIL} TipoObstaculos;

class IObstaculo
{
public:
virtual void moverseAleatoriamente() = 0;
virtual void pintar() = 0;
};

class ParedMovil : public IObstaculo
{
friend class FactoriaObjetos;
ParedMovil();
public:
virtual void moverseAleatoriamente() = 0;
virtual void pintar() = 0;
};
```

```
#ifndef _FACTORIA_INC_
#define _FACTORIA_INC_

#include "Luchadores.h"
#include "Armas.h"
#include "Obstaculos.h"

class FactoriaObjetos {
 FactoriaObjetos(){}; // Para desactivar
 void   operator=(FactoriaObjetos&) {};
 FactoriaObjetos(const FactoriaObjetos&) {};
public:
 static FactoriaObjetos& getInstance() {
 static FactoriaObjetos unicaInstancia;
 return unicaInstancia;
}
IArma* metodoFabricacionArmas (TipoArmas tipo) {
if(tipo == BOLAFUEGO) return new BolaFuego;
      else if(tipo == CUCHILLO) return new Cuchillo;
      else return NULL;}
ILuchador* metodoFabricacionLuchadores (TipoLuchador tipo)
{
      if(tipo == HOMBRE) return new Hombre;
      else if(tipo == DRAGON) return new Dragon;
```

Ejemplo Pang

Una nueva mejora se propone para el videojuego *Pang*. Se pretende implementar una ÚNICA factoría para la creación compleja de objetos (disparos, esferas, …). Siguiendo el Proceso Unificado, se pide:

1. Ingeniería inversa de la versión actual del videojuego sobre las clases de los disparos.
2. DCD de la nueva mejora, indicando los patrones que se emplean.
3. Ficheros de cabecera en C++ de la nueva versión.
4. Utilice las nuevas prestaciones en el servicio Mundo::Tecla(), sabiendo que *hombre.GetNumBonus()* retorna el número de *bonus*. Si tiene más de uno se generará lanzas, con un *bonus* se creará ganchos especiales y sin *bonus* se construirán ganchos.

Ejemplo PANG

```
#if !defined(_DISPARO_INC_)
#define _DISPARO_INC_

typedef
enum{GANCHO,GANCHO_ESPECIAL,LANZA}
tipoDisparo;

class Disparo
{
public:
              Disparo();
              virtual ~Disparo();


              tipoDisparo GetTipo();

              virtual void Dibuja()=0;

protected:
              tipoDisparo tipo;
};
```

```
#if !defined(_LANZA_INC_)
#define _LANZA_INC_


#include "Disparo.h"

class Lanza : public Disparo
{
public:
              void Dibuja();
              Lanza();
              virtual ~Lanza();

protected:
              float largo;

};

#endif
```

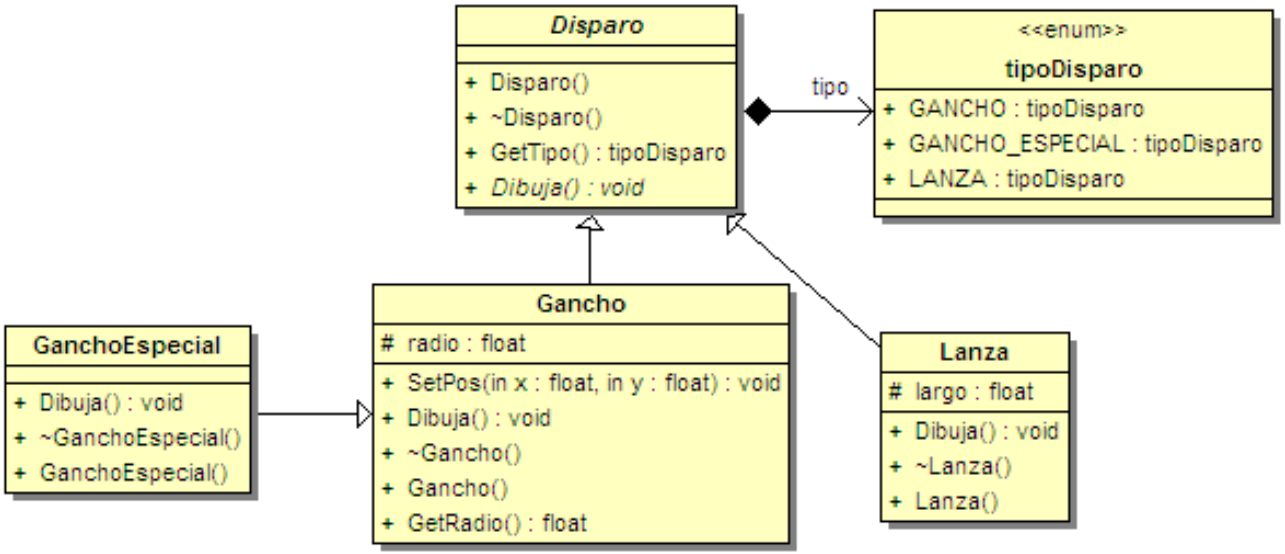
```
#if !defined(_GANCHO_INC_)
#define _GANCHO_INC_

#include "Disparo.h"

class Gancho : public Disparo
{
public:
   void SetPos(float x, float y);
   void Dibuja();
   Gancho();
```

```
#if
!defined(_GANCHO_ESPECIAL_INC_)
#define _GANCHO_ESPECIAL_INC_

#include "Gancho.h"

class GanchoEspecial : public
Gancho
{
public:
              void Dibuja();
```

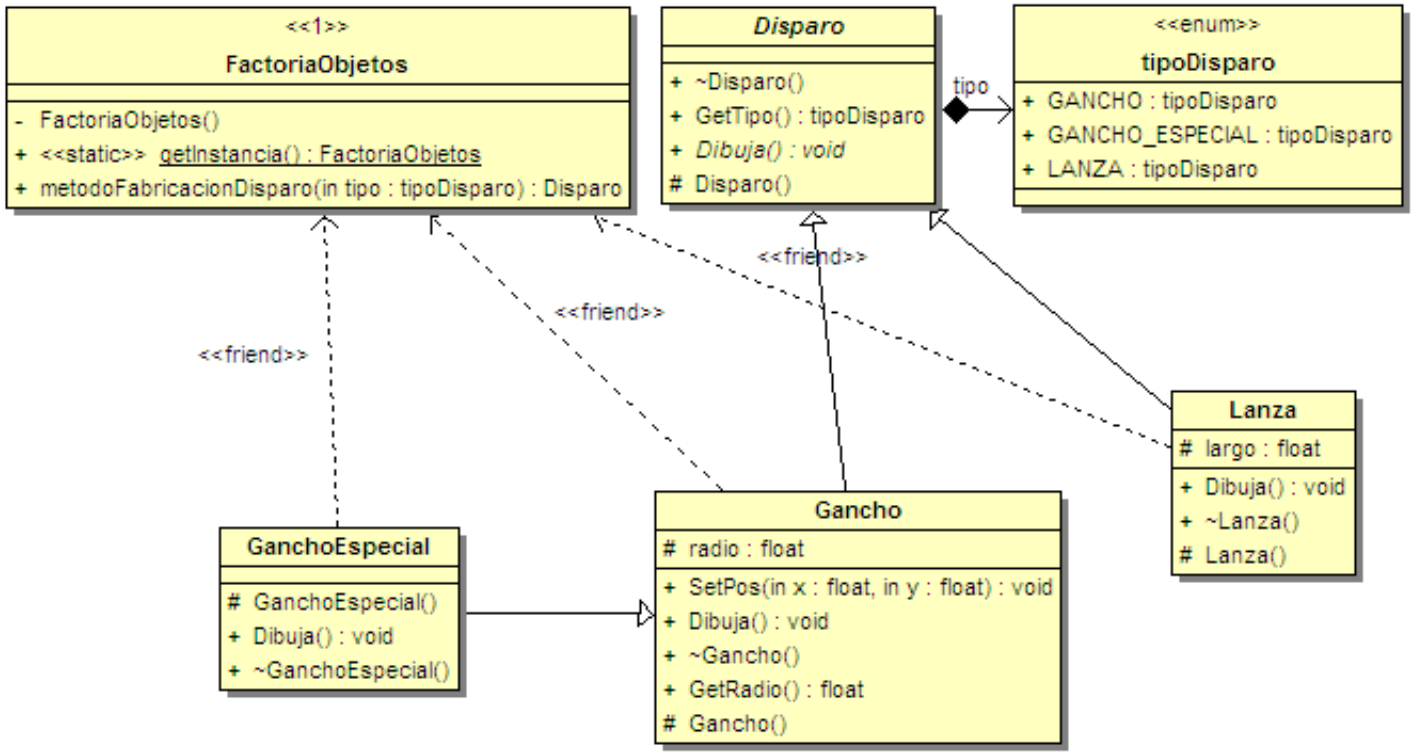
Ejemplo PANG

```
void Mundo::Tecla(unsigned char key)
{
          switch(key)
          {
                    case ' ': if(disparos.GetNumero()<MAX_DISPAROS)
                    {
                     Disparo* d=Factoria::CrearDisparo(hombre);
                     disparos.Agregar(d);
                    }
                    break;
          }
}
```

Ejemplo Pang



Ejemplo PANG



Ejemplo PANG

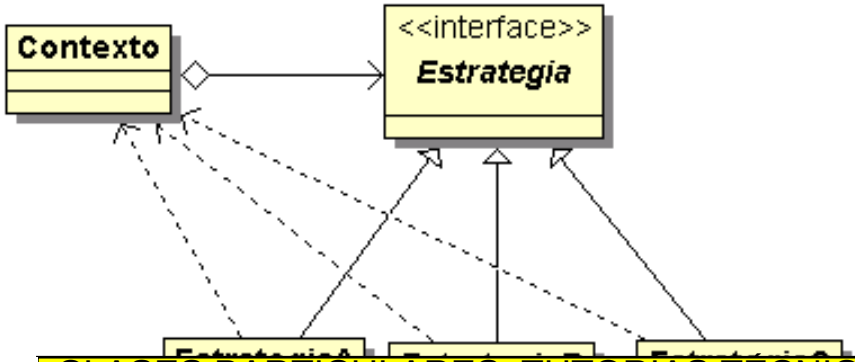
```
#if !defined(_FACTORIA_INC_)
#define _FACTORIA_INC_

#include "GanchoEspecial.h"
#include "Lanza.h"

class FactoriaObjetos
{
          FactoriaObjetos() {}
public:
          static FactoriaObjetos &getInstancia() {
                    static FactoriaObjetos laFactoria;
                    return (laFactoria);
          }
          Disparo * metodoFabricacionDisparo(tipoDisparo tipo) {
                    if(tipo == GANCHO) return new Gancho;
                    else if(tipo == GANCHO_ESPECIAL) return new
                                                        GanchoEspecial;
                    else if(tipo == LANZA) return new Lanza;
                    else return 0;
          }
};

#endif
```

Ejemplo PANG

```
void Mundo::Tecla(unsigned char key)
{
 switch(key)
 {
  case ' ': if(disparos.GetNumero()<MAX_DISPAROS)
  {
          FactoriaObjetos laFactoria =  FactoriaObjetos::getInstancia();
          Disparo *d;

          if(hombre.GetNumBonus()==1)
                    d = laFactoria. metodoFabricacionDisparo(GANCHO_ESPECIAL);
          else if (hombre.GetNumBonus()>=2)
                    d = laFactoria.metodoFabricacionDisparo(LANZA);
          else
                    d = laFactoria.metodoFabricacionDisparo(GANCHO_ESPECIAL);

          disparos.Agregar(d);
          }
          break;
   }
}
```

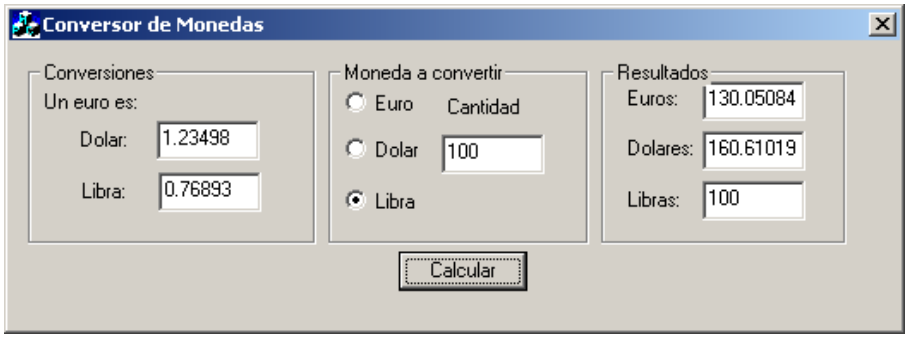
Estrategia

- *Problema*: ¿Cómo diseñar algoritmos que están relacionados? ¿Cómo diseñar que estos algoritmos se puedan variar dinámicamente?
- *Solución:* Defina cada algoritmo o estrategia en una clase independiente, con una interfaz común.
- La consecuencia de esta estructura es la variación dinámica de los algoritmos sin que los clientes se vean afectados.
- ¿Quién decide crear la estrategia adecuada?
  - Una factoría



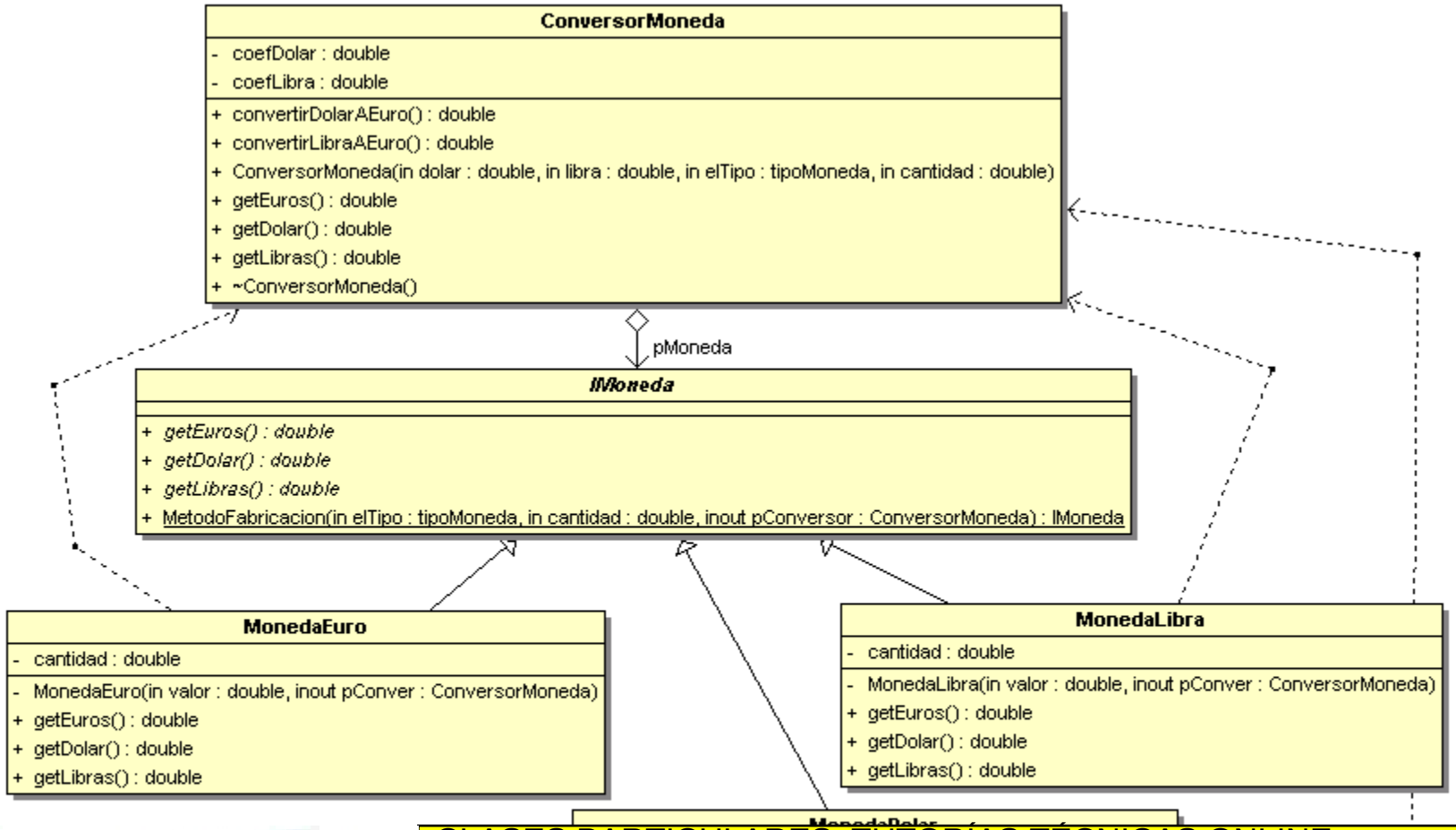
Ejemplo estrategia: conversor de monedas

Siguiendo el Proceso Unificado, diseñe una aplicación que sea un conversor de monedas. En una primera versión inicial, considere sólo euros, dólares y libras esterlinas. A la aplicación se le facilitará los valores de conversión entre las monedas y la cantidad de una moneda concreta a convertir en el resto de monedas.




```
void CConversorMonedasDlg::OnCalcular()
{
	// TODO: Add your control notification handler code here
	UpdateData(TRUE);

	ConversorMoneda elConversor(this->m_factorDolar,this->m_factorLibra,
			this->m_elTipo,this->m_Cantidad);
	this->m_Euros = elConversor.getEuros();
	this->m_Dolares = elConversor.getDolar();
	this->m_Libras = elConversor.getLibras();
```

Ejemplo estrategia: conversor de monedas



Ejemplo estrategia: conversor de monedas

```
typedef enum {EURO,DOLAR,LIBRA} tipoMoneda;
class ConversorMoneda;
class IMoneda
{
public:
          virtual double getEuros() = 0;
          virtual double getDolar() = 0;
          virtual double getLibras() = 0;
          static IMoneda* MetodoFabricacion(tipoMoneda,double,ConversorMoneda *);
};

class MonedaEuro: public IMoneda
{
          double cantidad; ConversorMoneda *pConversor;
          MonedaEuro(double valor, ConversorMoneda *pConver):
          cantidad(valor), pConversor(pConver) {}
          friend class IMoneda;
public:
          virtual double getEuros() {return cantidad;}
          virtual double getDolar()
          {return cantidad/pConversor->convertirDolarAEuro();}
          virtual double getLibras()
          {return cantidad/pConversor->convertirLibraAEuro();}
};
```

Ejemplo estrategia: conversor de monedas

```
class ConversorMoneda
{
          IMoneda *pMoneda; double coefDolar; double coefLibra;
public:
          double convertirDolarAEuro()  {return 1/coefDolar;}
          double convertirLibraAEuro() {return 1/coefLibra;}
          ConversorMoneda(double dolar, double libra, tipoMoneda elTipo, double cant):
          coefDolar(dolar), coefLibra(libra)
          {
                    pMoneda = IMoneda::MetodoFabricacion(elTipo, cant, this);
          }
          double getEuros() {return pMoneda->getEuros(); }
          double getDolar() {return pMoneda->getDolar(); }
          double getLibras(){return pMoneda->getLibras(); }
          ~ConversorMoneda()
          {         if(pMoneda !=0) delete pMoneda; }
};
```

Ejercicio: conversor de longitudes

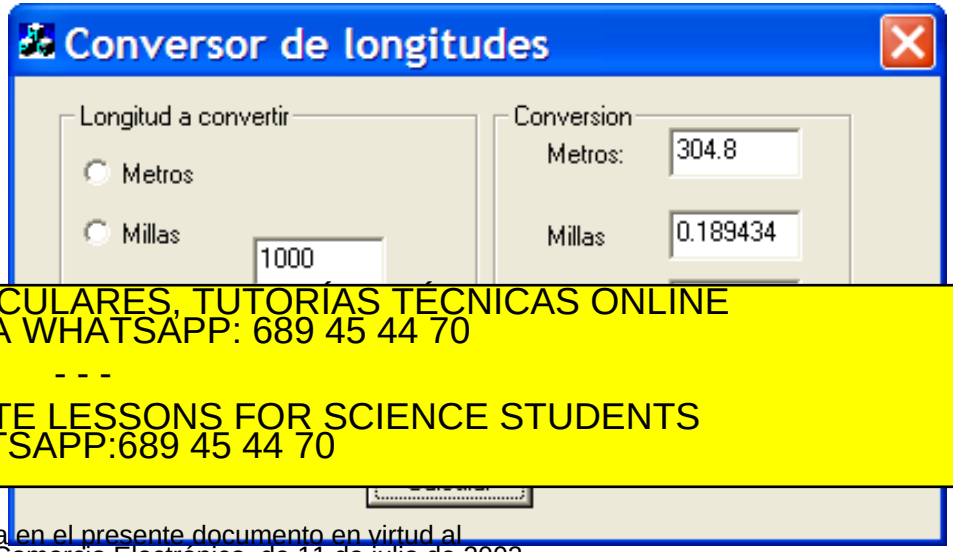
- Desarrollar una aplicación que convierta las magnitudes de longitud de un sistema a otro, sabiendo que:

  1 milla = 1609 m

  1 pulgada = 25.4 mm

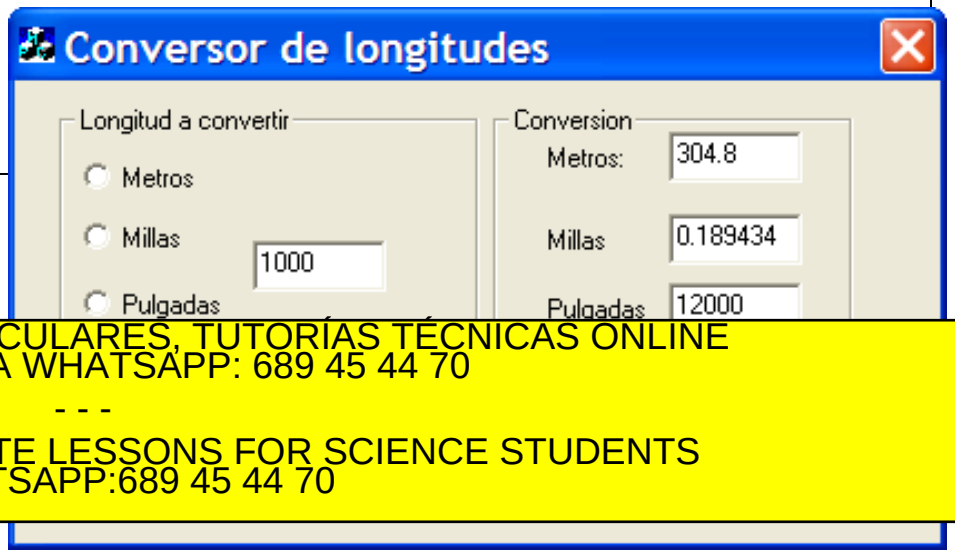
  1 pie = 30.48 cm

- Se pide:
  1. AOO: Modelo del dominio.
  2. DOO: Diagrama de clases de diseño. Indicar los patrones que se están aplicando.
  3. Implementación en C++.

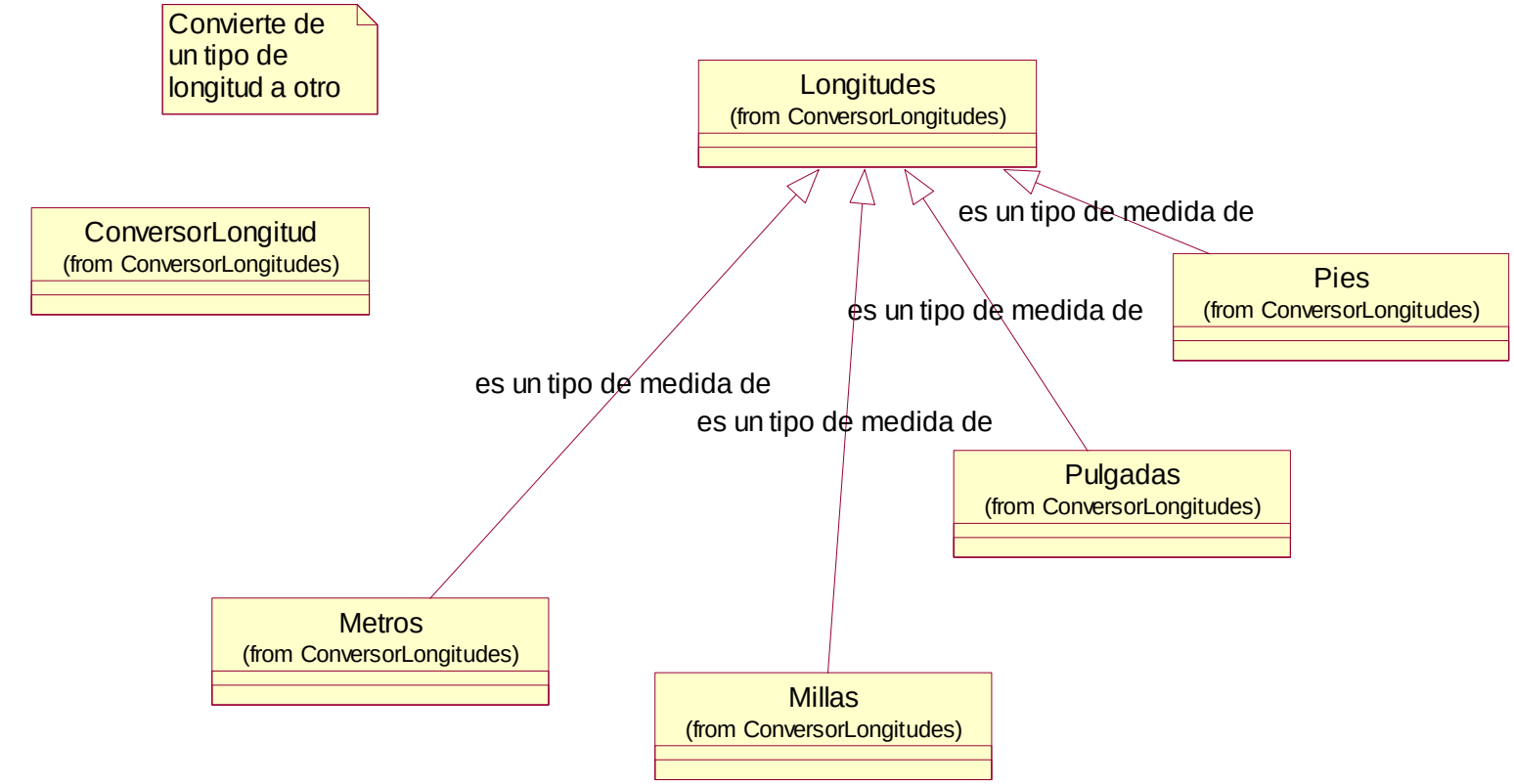


Ejercicio: conversor de longitudes

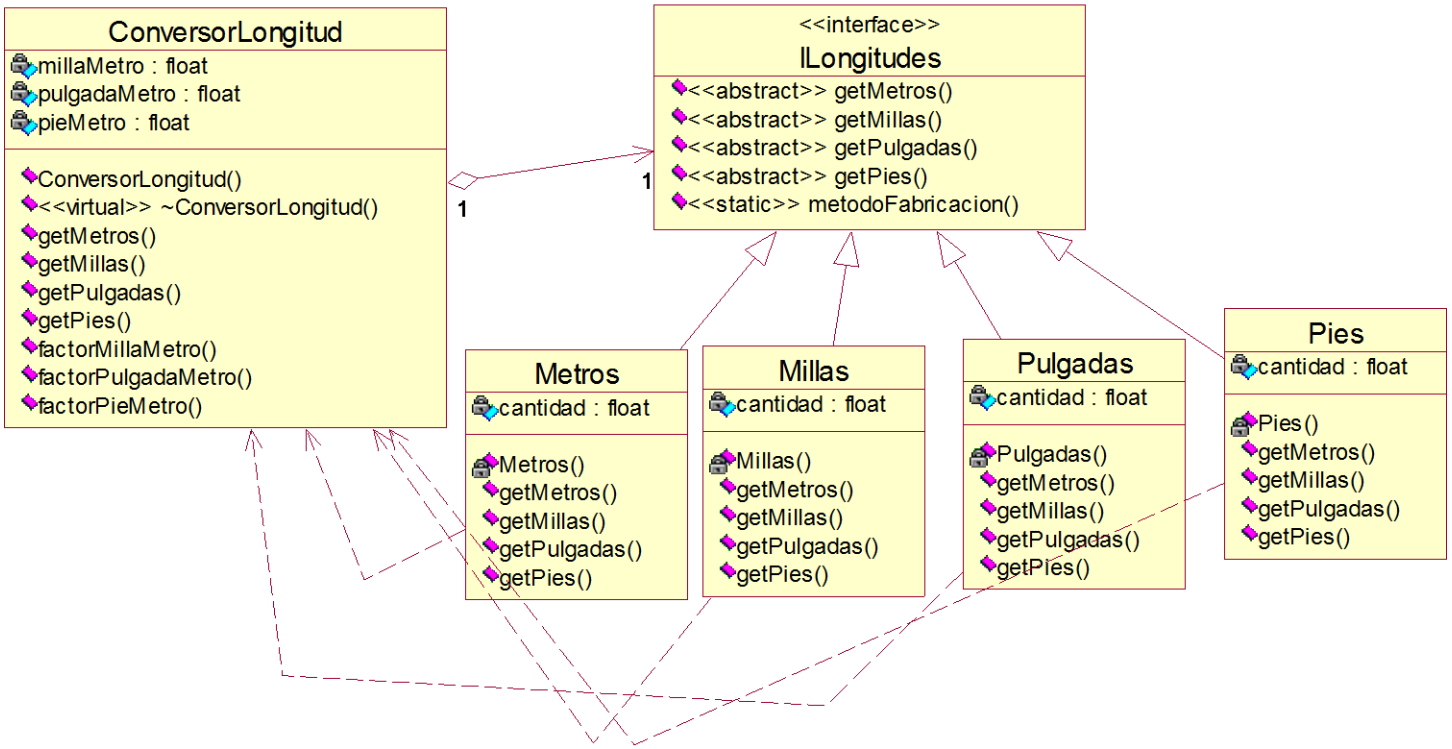
```
void CConversorLongitudesDlg::OnCalcular()
{
        // TODO: Add your control notification handler code here
        UpdateData(TRUE);

        ConversorLongitud elConversor(this->m_tipoLongitud,this->m_longitud);
        this->m_Metros = elConversor.getMetros();
        this->m_Millas = elConversor.getMillas();
        this->m_Pulgadas = elConversor.getPulgadas();
        this->m_Pies = elConversor.getPies();

        UpdateData(FALSE);
}
```



Ejercicio: conversor de longitudes



Ejercicio: conversor de longitudes



Ejercicio: conversor de longitudes

- Los patrones utilizados son: Estrategia(GoF) que incluye Variaciones Protegidas (GRASP), Método de Fabricación (GoF) y Polimorfismo (GRASP)
- El código de test sería:

```
void CConversorLongitudesDlg::OnCalcular()
{
        // TODO: Add your control notification handler code here
        UpdateData(TRUE);

        ConversorLongitud elConversor(this->m_tipoLongitud,this->m_longitud);
        this->m_Metros = elConversor.getMetros();
        this->m_Millas = elConversor.getMillas();
        this->m_Pulgadas = elConversor.getPulgadas();
        this->m_Pies = elConversor.getPies();

        UpdateData(FALSE);
```

Ejercicio: conversor de longitudes

```
typedef enum {METROS,MILLAS,PULGADAS,PIES}      tipoLongitudes;

class ConversorLongitud;
class ILongitudes
{
public:
            virtual float getMetros() = 0;
            virtual float getMillas() = 0;
            virtual float getPulgadas() = 0;
            virtual float getPies() = 0;
            static ILongitudes * metodoFabricacion(tipoLongitudes,float,ConversorLongitud *);
};
-------------------------------------------------------------------------------------------------------------------------
#include "Longitudes.h"

class ConversorLongitud
{
            ILongitudes *plongitudes;
            float millaMetro;
            float pulgadaMetro;
            float pieMetro;

public:
            ConversorLongitud(tipoLongitudes,float);
            virtual ~ConversorLongitud();

            float getMetros() { return plongitudes->getMetros(); }
            float getMillas() { return plongitudes->getMillas(); }
            float getPulgadas() { return plongitudes->getPulgadas(); }
```

Ejercicio: conversor de longitudes

```
#include "ConversorLongitud.h"


class Metros : public ILongitudes
{
          float cantidad; ConversorLongitud *pConversor;
          Metros(float valor,ConversorLongitud *pC):
                    cantidad(valor), pConversor(pC) {}
          friend class ILongitudes;
public:
          float getMetros() { return cantidad;}
          float getMillas() {return cantidad/pConversor->factorMillaMetro();}
          float getPulgadas() {return cantidad/pConversor->factorPulgadaMetro();}
          float getPies() {return cantidad/pConversor->factorPieMetro();}
};
```

Ejercicio de examen

Se desea hacer una aplicación que sirva para calcular las nóminas de una compañía. Al salario base de cada empleado hay que quitarle la retención del IRPF (p. e. 20%) para calcular su salario neto. Como existen diferentes políticas salariales en la empresa, se desea hacer un programa fácilmente extensible a nuevas políticas. De momento se pretende abordar dos de ellas:

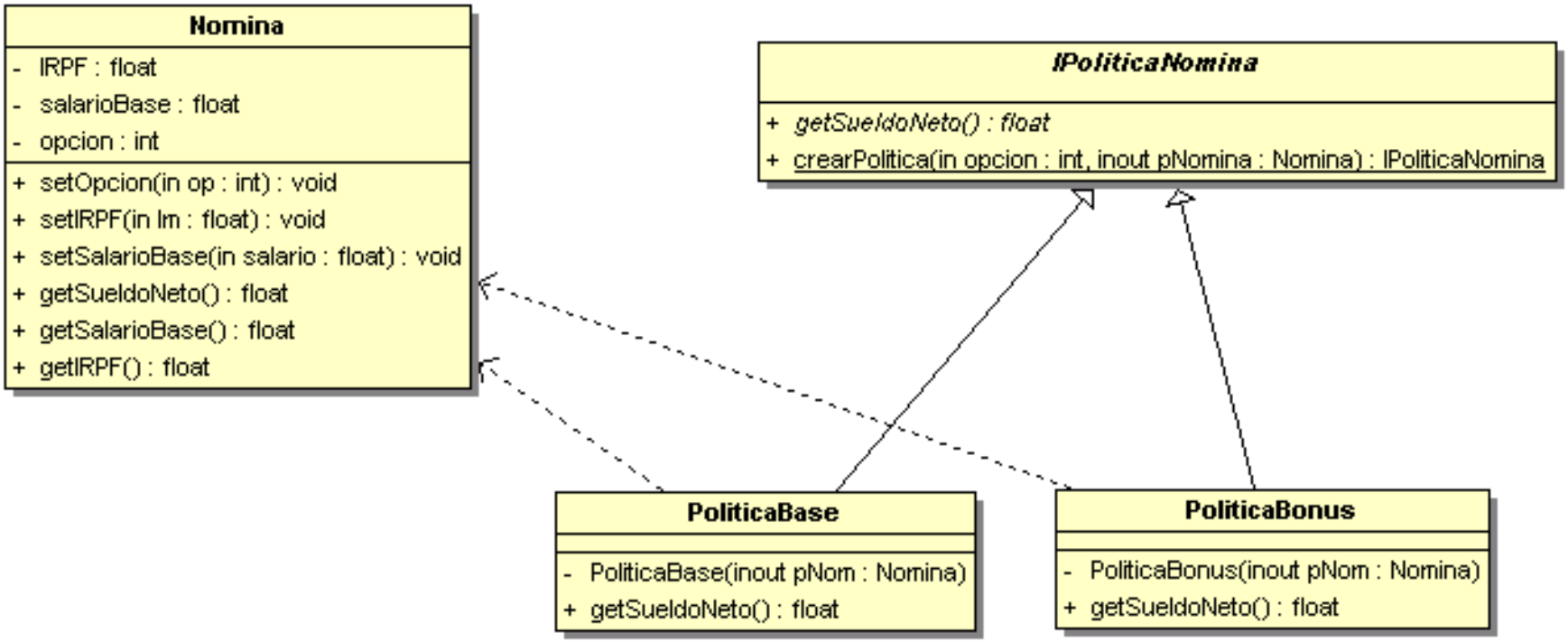
a. El sueldo ordinario
b. El sueldo con bonus, consistente en aumentar el salario base (antes de la retención) un 35%.

Se ha desarrollado el siguiente programa principal. Se pide:

1. Diagrama de Clases de Diseño de una arquitectura que permita una fácil extensión a nuevas políticas. Indicar los patrones utilizados. (5 puntos)
2. Implementación en C++ de la solución. (5 puntos)

```
int main(){
Nomina nomina;
int opcion;
cout<<"1. Nomina ordinaria"<<endl;
cout<<"2. Nomina con bonus"<<endl;
cin>>opcion;
nomina.setOpcion(opcion);

cout<<"IRPF en %: ";
float IRPF;
cin>>IRPF;
nomina.setIRPF(IRPF);

cout<<"Salario base: ";
float salario;
cin>>salario;
nomina.setSalarioBase(salario);
```

Ejercicio de examen

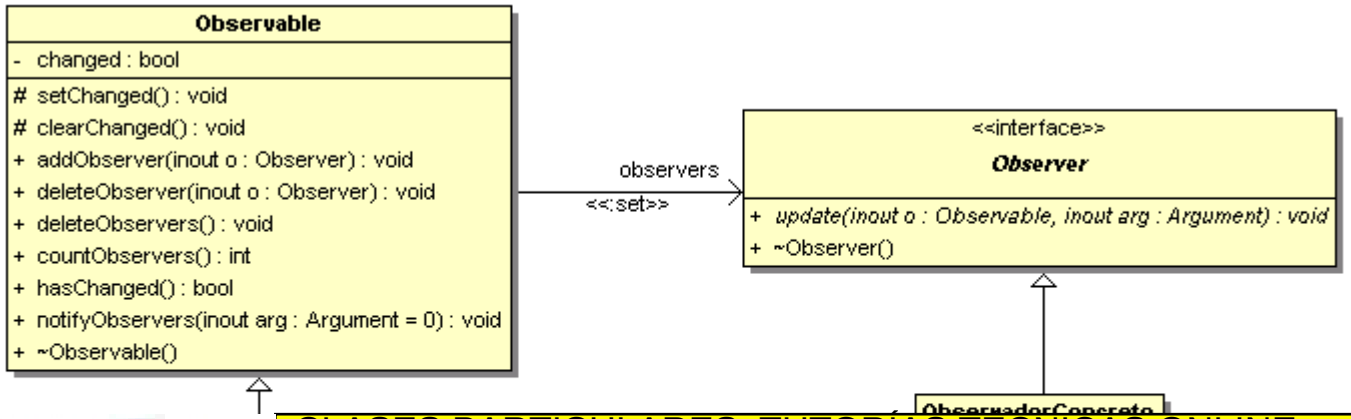


Ejercicio de examen

```
class Nomina;
class IPoliticaNomina{
public:
              virtual float getSueldoNeto() = 0;
              static IPoliticaNomina * crearPolitica(int, Nomina *);
};

class Nomina {
              float IRPF, salarioBase; int opcion;
public:
              void setOpcion(int op) {opcion = op;}
              void setIRPF(float Im) {IRPF = Im;}
              void setSalarioBase(float salario) {salarioBase = salario;}
              float getSueldoNeto() {
                  float resultado = -1;
                  IPoliticaNomina *pPolitica = IPoliticaNomina::crearPolitica(opcion, this);
                  if(pPolitica){
                            resultado = pPolitica -> getSueldoNeto();
                            delete pPolitica;
                  }
                  return (resultado );
              }
              float getSalarioBase() {return salarioBase;}
              float getIRPF() {return IRPF;}
};

class PoliticaBase : public IPoliticaNomina {
              friend IPoliticaNomina;
              Nomina *pNomina;
              PoliticaBase(Nomina * pNom): pNomina(pNom) {}
public:
              float getSueldoNeto(){ return pNomina->getSalarioBase()*(1-(pNomina->getIRPF()/100));}
};
#define BONUS 1.35
```

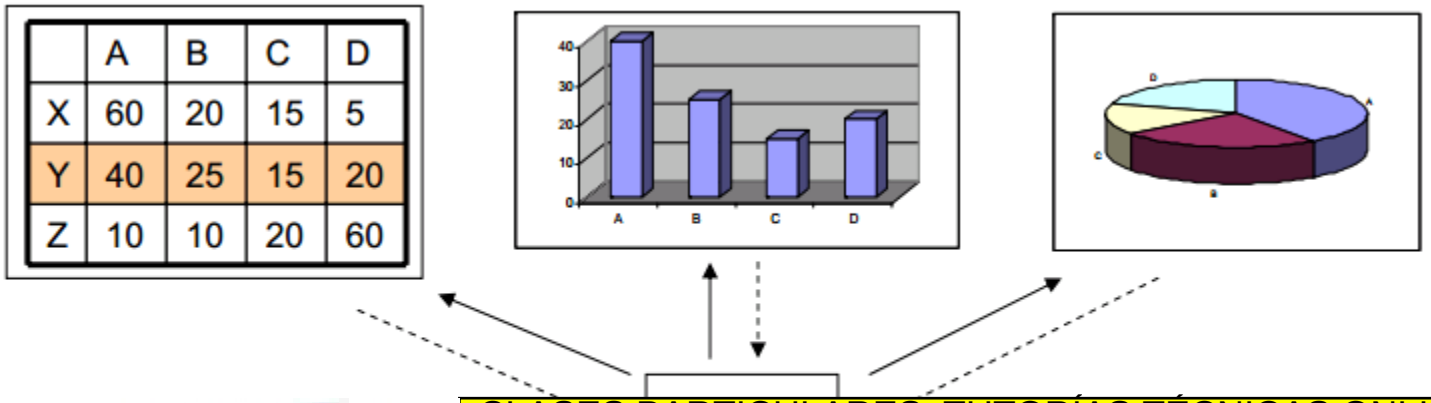
Patrón Observador

- *Problema*: Diferentes tipos de objetos receptores están interesados en el cambio de estado o eventos de un objeto emisor y quieren reaccionar cada uno a su manera cuando el emisor genere un evento. Además, el emisor quiere mantener bajo acoplamiento con los suscriptores ¿Qué hacer?
- *Solución*: Defina una interfaz "subscriptor" u "oyente". Los suscriptores implementan esta interfaz. El emisor dinámicamente puede registrar suscriptores que están interesados en un evento y notificarles cuando ocurre un evento.
- Este patrón define una dependencia de uno-a-muchos



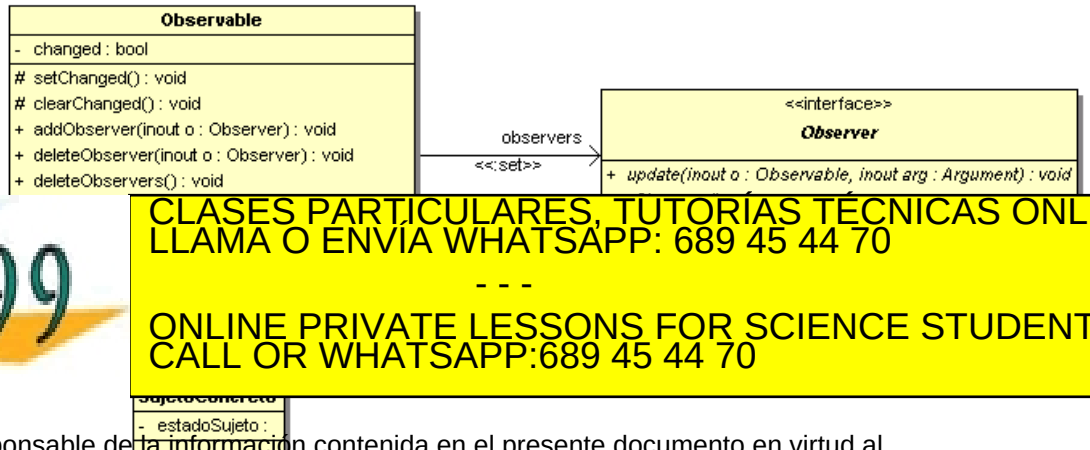
Ejemplo de problemática observador

- Un objeto de hoja de cálculo y un gráfico de barras pueden representar una información contenida en el mismo objeto de datos del dominio.
- La hoja de cálculo y el gráfico de barras no se conocen entre sí, permitiéndose de esta manera reutilizarse, pero gracias a este patrón se comportan como si se conocieran.
- Cuando el usuario cambia la información de la hoja de cálculo, la barra de gráfica refleja los cambios inmediatamente y viceversa.



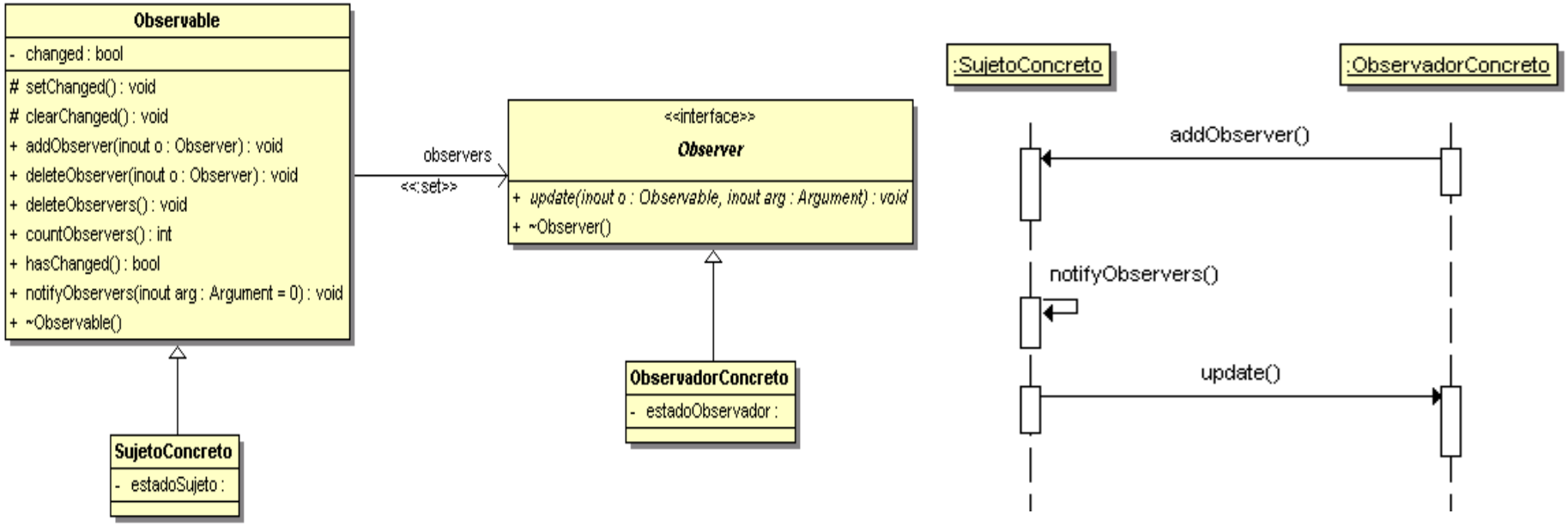
Roles del patrón observador

- *Observable*
  - Un Observable puede ser observado por cualquier número de objetos Observador. Proporciona una interfaz para asignar y quitar objetos Observador.
- *Observador*
  - Define una interfaz para actualizar los objetos que deben ser notificados ante cambios en un sujeto observable.
- *ObservableConcreto*
  - almacena el estado de interés para los objetos ObservadorConcreto. Envía una notificación a sus observadores cuando cambia su estado.
- *ObservadorConcreto*
  - Mantiene una referencia a un objeto ObservableConcreto. Guarda un estado que debería ser consistente con el del sujeto observable.



Roles del patrón observador

"El observable envía notificaciones sin tener que conocer quiénes son sus observadores"

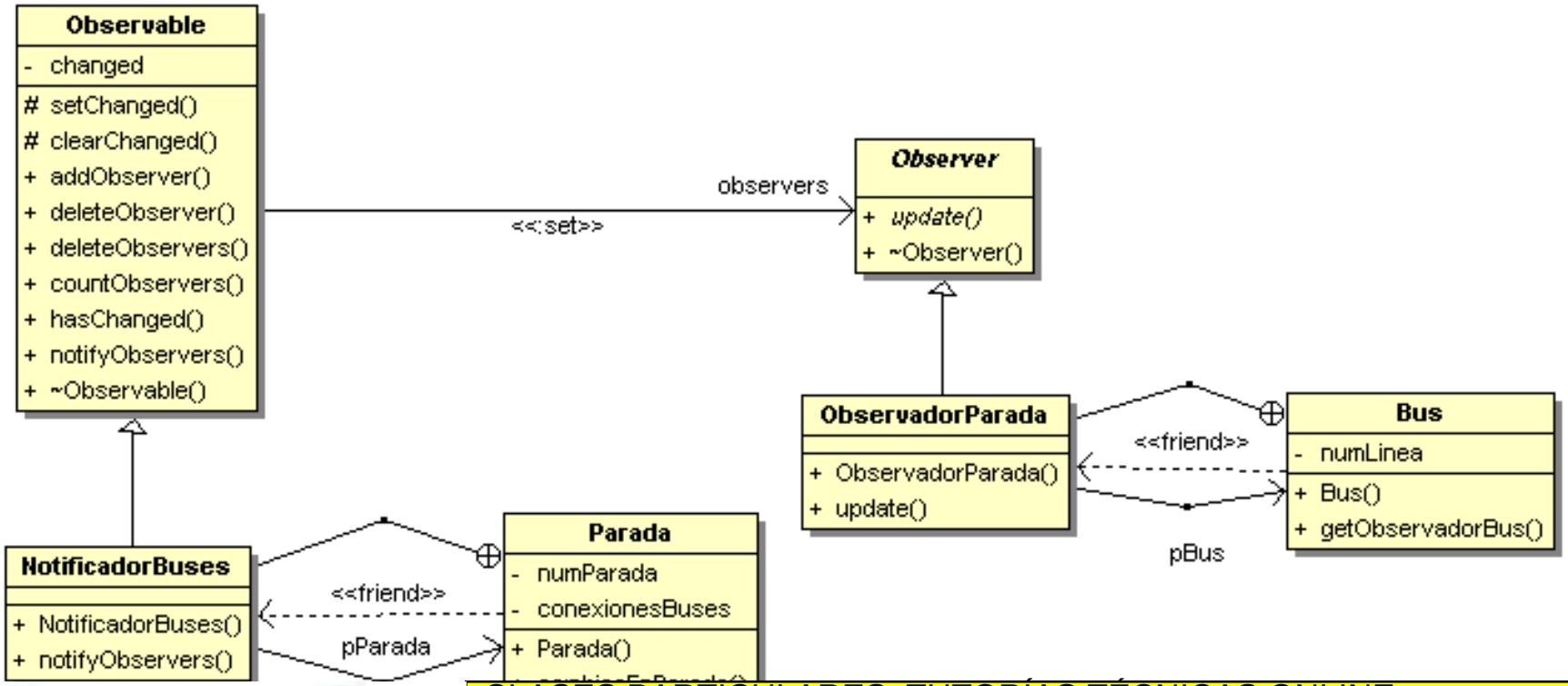


Interfaz de observador y observable

```
class Observable;
class Argument {};
class Observer {
public:
  // Called by the observed object, whenever
  // the observed object is changed:
  virtual void update(Observable* o, Argument* arg) = 0;
  virtual ~Observer() {}
};
```

```
class Observable {
  bool changed;
  std::set<Observer*> observers;
protected:
  virtual void setChanged() { changed = true; }
  virtual void clearChanged() { changed = false; }
public:
  virtual void addObserver(Observer& o) {
    observers.insert(&o);
  }
  virtual void deleteObserver(Observer& o) {
    observers.erase(&o);
  }
  virtual void deleteObservers() {
    observers.clear();
  }
  virtual int countObservers() {
    return observers.size();
  }
  virtual bool hasChanged() { return changed; }
  // If this object has changed, notify all of its observers:
  virtual void notifyObservers(Argument* arg = 0) {
    if(!hasChanged()) return;
    clearChanged(); // Not "changed" anymore
    std::set<Observer*>::iterator it;
```

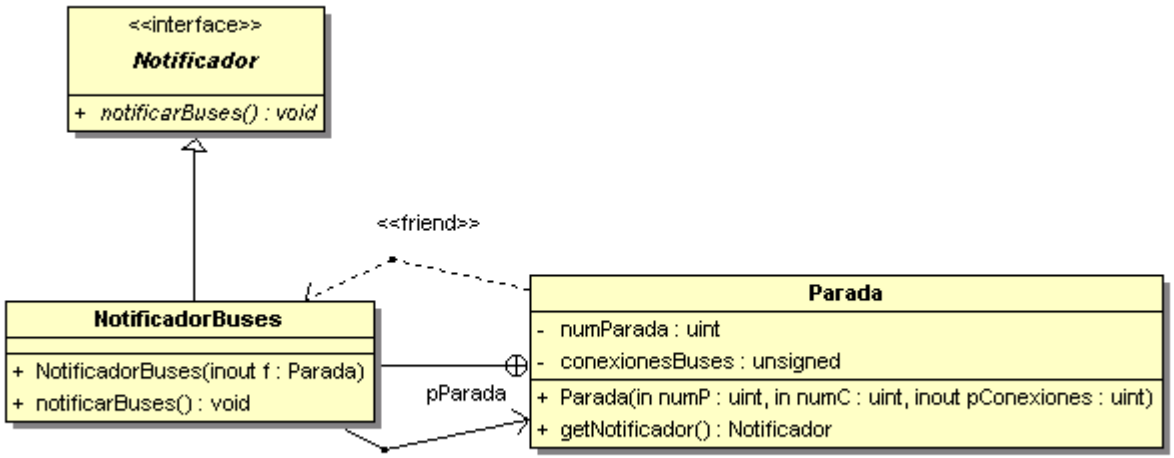
Ejemplo autobús-parada



Las clases internas

- Tanto las clases que están a la escucha como las observables requieren un encapsulamiento que las permitan mantener la propiedad de bajo acoplamiento.
  - No tienen que conocerse las unas a las otras
    - Para evitar el acoplamiento no puede derivarse de las clases de observador y observable
  - Solución: las clases internas.
    - Jerarquía de clases
    - Tiene acceso a los datos de la instancia del objeto que la ha creado.

Ejemplo de clases internas

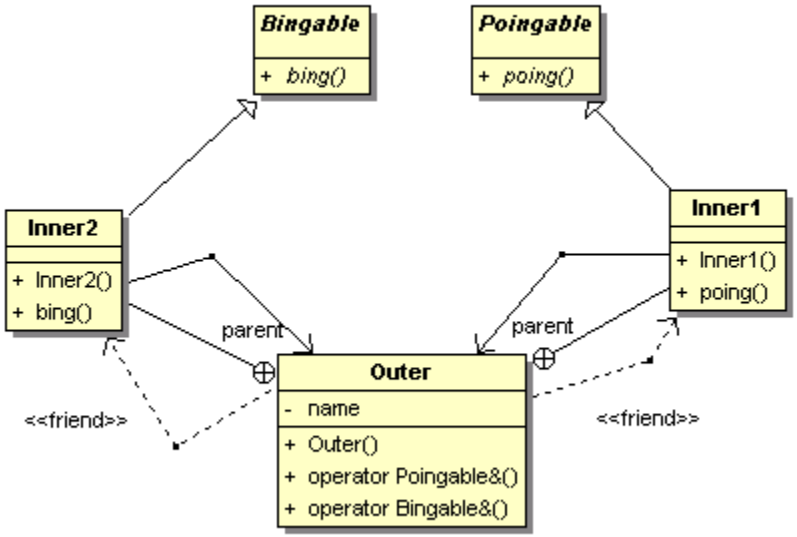



```
int main() {
  unsigned conexionesParada1111[ ] ={60,148,78};
  Parada num1111(1111,3,conexionesParada1111);

  Notificador *pNotificador = num1111.getNotificador();
  pNotificador->notificarBuses();
}
```

Ejemplo de clases internas

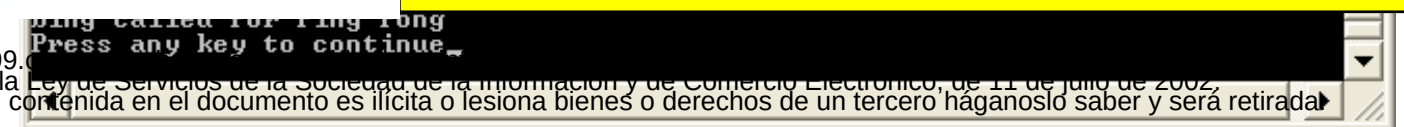
```
class Notificador{
public: virtual void notificarBuses() = 0;
};
class Parada {
  unsigned numParada;
  vector<unsigned> conexionesBuses;
public:
  class NotificadorBuses; // Clase interna:
  friend class Parada::NotificadorBuses;
  class NotificadorBuses : public Notificador {
    Parada* pParada;
    public:
                NotificadorBuses(Parada* f) : pParada(f){}
                void notificarBuses() {
                        cout << "Parada " << pParada->numParada <<". Conexiones: ";
                        for(unsigned i=0;i<pParada->conexionesBuses.size();i++)
                                cout << pParada->conexionesBuses[i] <<" ";
                        cout << endl;
                }

    } elNotificardorLineasBuses;
                                                                                    {
  Notificador* getNotificador() {return &elNotificardorLineasBuses;}
};
```

Ejemplo de clases internas

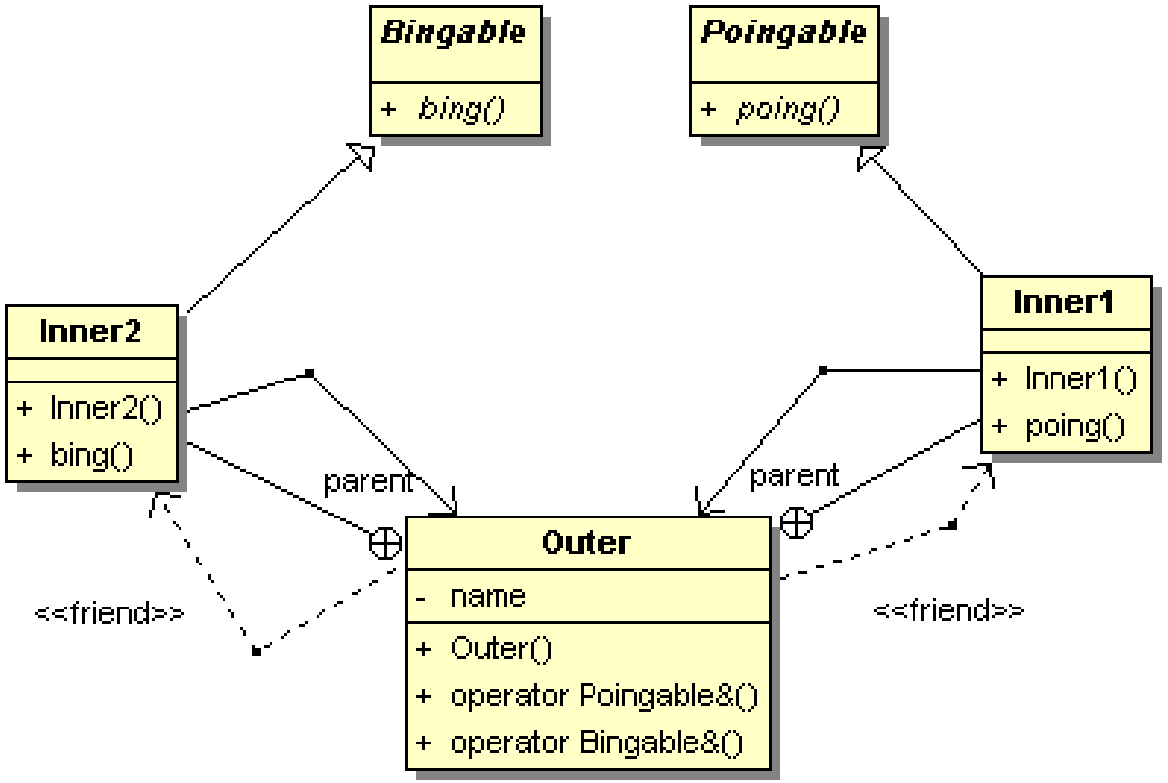



```
class Poingable {
public:
  virtual void poing() = 0;
};

class Bingable {
public:
```

```
void callPoing(Poingable& p) {
  p.poing();
}

void callBing(Bingable& b) {
  b.bing();
}

int main() {
  Outer x("Ping Pong");
  // Like upcasting to multiple base types!:
  callPoing(x);
  callBing(x);
}
```

Ejemplo de clases internas



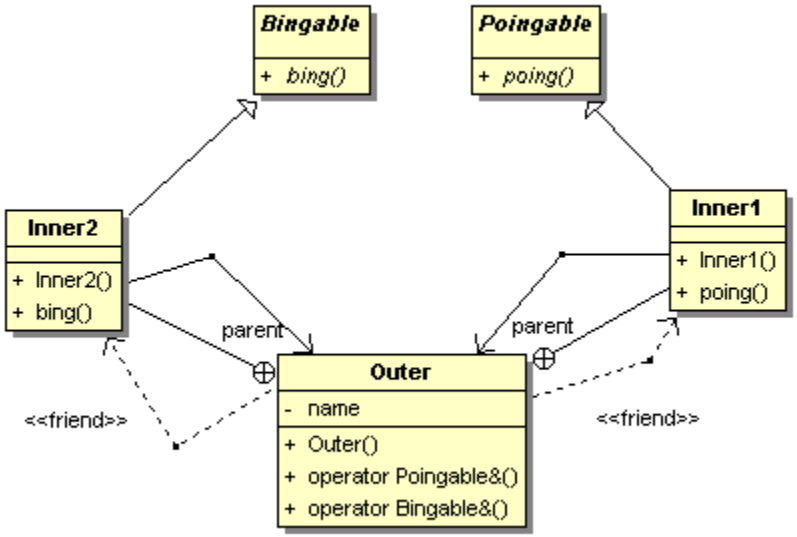
Ejemplo de clases internas

```
class Poingable {
public:
  virtual void poing() = 0;
};

class Bingable {
public:
  virtual void bing() = 0;
};
```

```
class Outer {
  string name;

  class Inner1; // Define one inner class:
  friend class Outer::Inner1;
  class Inner1 : public  Poingable {
    Outer* parent;
  public:
    Inner1(Outer* p) : parent(p) {}
    void poing() {
      cout << "poing called for "
        << parent->name << endl;
      // Accesses data in the outer class object
    }
  } inner1;

  class Inner2; // Define a second inner class:
  friend class Outer::Inner2;
  class Inner2 : public Bingable {
    Outer* parent;
  public:
    Inner2(Outer* p) : parent(p) {}
    void bing() {
      cout << "bing called for "
```

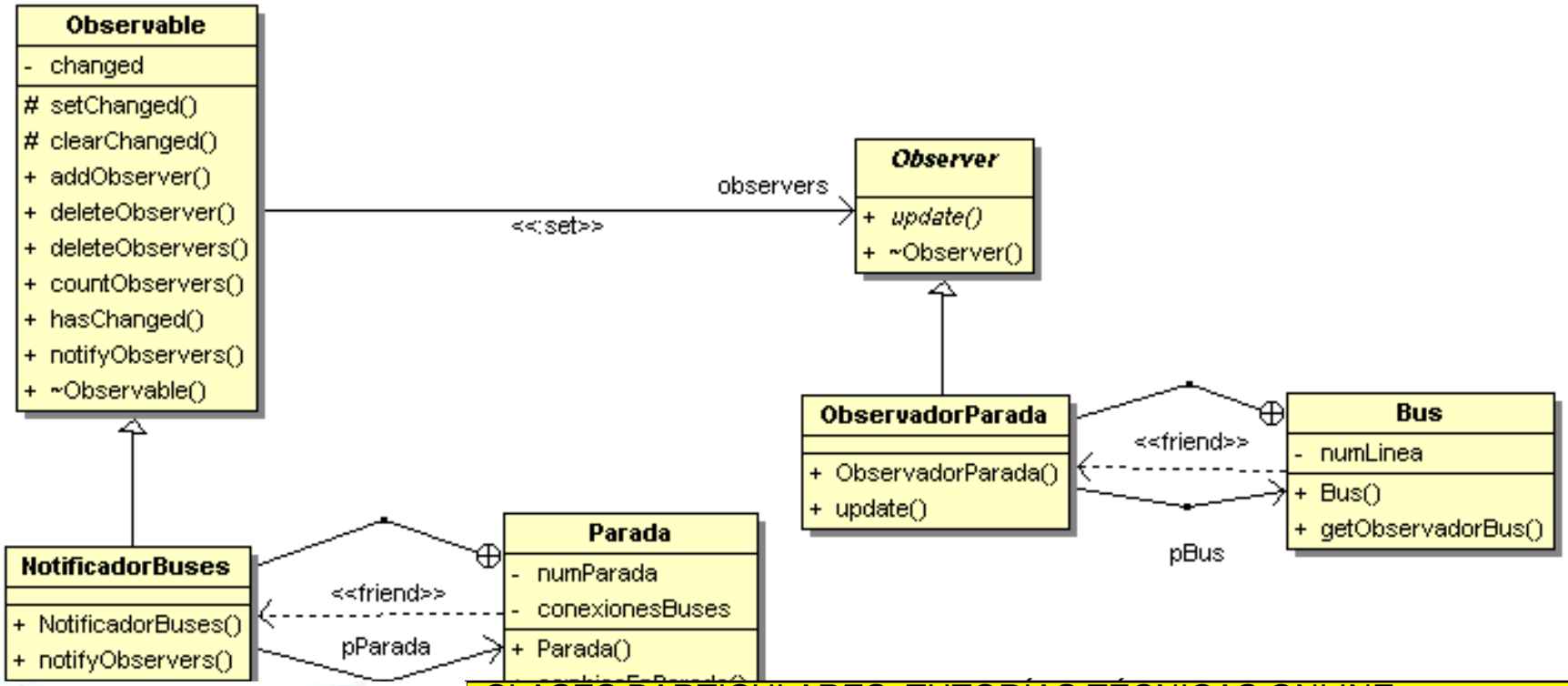
```
public:
  Outer(const string& nm)
  : name(nm), inner1(this), inner2(this) {}
  // Return reference to interfaces
  // implemented by the inner classes:
```

Ejemplo de clases internas

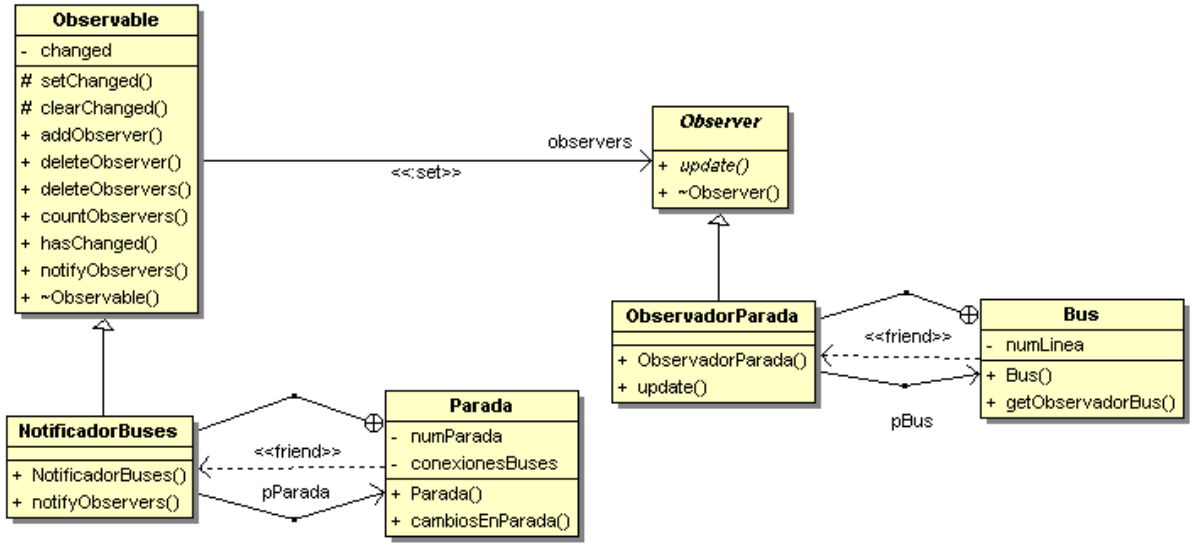



```
void callPoing(Poingable& p) {
  p.poing();
}

void callBing(Bingable& b) {
  b.bing();
}

int main() {
  Outer x("Ping Pong");
  // Like upcasting to multiple base types!:
  callPoing(x);
  callBing(x);
}
```

Ejemplo autobús-parada



Ejemplo autobús-parada




```
int main() {
  unsigned conexionesParada1111[] ={60,148,78};
  Parada num1111(1111,3,conexionesParada1111);
  Bus elBus1(60), elBus2(60), elBus3(148);
  num1111.elNotificardorLineasBuses.addObserver(elBus1.getObservadorBus());
  num1111.elNotificardorLineasBuses.addObserver(elBus2.getObservadorBus());
  num1111.elNotificardorLineasBuses.addObserver(elBus3.getObservadorBus());

  // elBus3 ya no está interesado en esta parada
  num1111.elNotificardorLineasBuses.deleteObserver(elBus2.getObservadorBus());
  // Notificar las líneas de las paradas
}
```